A Security Architecture Based on Trust Management for Pervasive Computing Systems*

Lalana Kagal
Computer Science & Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21244
lkagal1@cs.umbc.edu

Jeffrey Undercoffer
Computer Science & Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21244
junder2@cs.umbc.edu

Filip Perich
Computer Science & Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21244
fperic1@cs.umbc.edu

Anupam Joshi
Computer Science & Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21244
joshi@cs.umbc.edu

Tim Finin
Computer Science & Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21244
finin@cs.umbc.edu

Abstract

Traditionally, stand-alone computers and small networks rely on user authentication and access control to provide security. These physical methods use system-based controls to verify the identity of a person or process, explicitly enabling or restricting the ability to use, change, or view a computer resource. However, these strategies are inadequate for the increased flexibility that distributed networks such as the Internet and ubiquitous/pervasive computing environments require, as these systems lack central control and in addition, their users are not all predetermined. Users in pervasive environments expect to access locally hosted resources and services anytime and anywhere leading to serious security risks and access control problems. We propose a solution based on distributed trust management which involves developing a security policy, assigning credentials to entities, verifying that the credentials conform to the policy, delegating trust to third parties, revoking rights and reasoning about users' access rights. This paper presents an infrastructure that complements existing security features like Public Key Infrastructure (PKI) and Role Based Access Control with *distributed trust management* to provide a highly flexible mode of enforcing security in a pervasive computing environments.

1 Introduction

As computing becomes more pervasive, people expect to access services and information at anytime and anywhere, leading to the development of *ubiquitous/pervasive computing*, of which *SmartSpaces* is a specific example. A SmartSpace environment provides services and resources that users can access using some short range wireless communications such as Bluetooth, IEEE 802.11, or Infrared, via any hand-held device, within a confined space. Vigil is generally applicable to distributed systems, but by making it lightweight and independent of the communication medium, we have geared it towards ubiquitous environments.

For the SmartSpace framework, we extended the C2 [16] architecture, which in turn is based on the Centaurus [10] model. In Centaurus a Client can access the services provided by the nearest Centaurus Service Manager (SM) via some short-range communication. The SM acts as an active proxy by executing services on behalf of any Client that requests them. This minimizes the resource consumption on the Client and also avoids having the services installed on each Client that wishes to use them. Our infrastructure is designed to reduce the load on portable devices and provide a media independent infrastructure and communication protocol for the provision of services.

This paper is organized as follows: Section 2 discusses other research and technologies and briefly compares Vigil to similar projects. Section 3 details the system design and architecture and Section 4 details the working of the simplified Public Key Infrastructure used in Vigil. Section 5 discusses our ongoing work and presents a brief summary of the work presented in this paper.

2 Related Work

Though there are several academic and commercial projects that are aimed at realizing the *SmartSpaces* scenario, most of which have a rather simple security framework and none of which use *distributed trust* as a way to resolve the complex security issues.

Some of the projects that address the *SmartSpaces* scenario are the joint Unisys Corporation/Orange [2] experimental house in Hertford, England, UC Berkeley's Ninja project [6] [7], the University of Washington's Portolano project [4], and Stanford's Interactive Workspaces Project [3]. As demonstrated by the Unisys/Orange project, the concept of SmartHomes is transitioning from the purely academic to industry oriented research. The Unisys/Orange project is an experimental "intelligent" house that responds to voice commands to "dim the lights' or "turn up the volume on the television". In addition to voice, the home owner can interface with the house through a Wireless Application Protocol (WAP) telephone, web browser, or a Personal Digital Assistants (PDA). The In the Centaurus project [10], the main design goal is the development of a framework for building portals to services using various types of mobile devices. Centaurus provides a uniform infrastructure for access to heterogeneous hardware and software components. It uses a language based on XML as the sole data exchange format between

*This work was supported by NSF Awards IIS 9875433 and CCR 0070802, and the Defense Advanced Research Projects Agency under contract F30602-00-2-0 591 AO K528.

the service requester and service provider. This language called Centaurus Capability Markup Language (CCML), provides an extensible and simple content description that enables the creation of a user interface. Vigil uses CCML as its sole form of data exchange. Another important research project is UC Berkeley's Project Ninja [6], [7] which employs Group Controllers, Certificate Authorities, and a hierarchy of Service Discovery Service Servers. However, unlike Vigil, it does not delegate state management to the Services themselves nor does it allow the Service Manager to serve exclusively as a cache. This approach is at a disadvantage because as the complexity of distributed state management increases the fault tolerance of the system decreases. For security and information assurance Ninja utilizes encryption between all entities within the system. This implies a high computational overhead on the endpoints of the communication regardless of whether the endpoint is a PDA, cell phone, or a powerful workstation. Vigil does not make the assumption that the end points are computationally robust and instead relies on a simplified Public Key Infrastructure.

Matt Blaze's PolicyMaker [13] is probably one of the first forays into distributed trust management though the concept has its roots in Pretty Good Privacy (PGP) [18], Simple Public Key Infrastructure (SPKI) [8], and Role Based Access Control (RBAC) [12]. PGP [18] is a simple way of sending secure email using a *web of trust*, without exchanging a key and without a central authority. In PGP, a keyholder (an individual associated with a public/private key pair) learns about the public keys of others through introductions from trusted friends. The largest problem associated with PGP is key distribution and management. SPKI was the first proposed standard for distributed trust management [8]. This solution, though simple and elegant, includes only a rudimentary notion of delegation, which is crucial to the developed of *distributed trust*. PolicyMaker [13] is able to interpret policies and answer questions about access rights. Unfortunately, the development of policy is slightly complicated and not easy for non-programmers to use. This poses quite a problem, as it is generally non-programmers who will define the policies. Role Based Access Control [12] is probably one of the best known methods for access control, where entities are assigned roles, and there are rights associated with each role. Unfortunately, this is difficult for systems where it is not possible to assign roles to all users and foreign users are common. Also it is not possible to change access rights associated with a particular entity without modifying the roles.

We drew on the key points of most of the above-mentioned schemes and designed an infrastructure that uses PKI for authentication and policies to enforce security [9]. A policy contains basic/axiomatic rights, rules for assigning roles, rights associated with roles, rules for delegation, and rules for checking the validity of requests. An entity (service or client) can have many roles in the system and is assigned all rights associated with those roles. Rights can also be delegated to entities or revoked from entities without modifying the roles in any way. Our system will allow an entity in the system to delegate any right that it may have. Whether these delegations are honored depends on the policy. Constraints can be added to both the actual delegation and to the delegatee, tightening control on the rights and permissions. In our model, we use also constraints on *redelegation* that controls whether the permission can be further delegated and to whom it can be delegated. Rights can also be revoked; making rights extremely dynamic. We have found that these features address the required issues of pervasive systems; authentication, delegation and access control, successfully.

3 Architecture

Our system is designed to provide security and access control in distributed systems, and has been optimized to work in *SmartSpaces*, where most of the clients are hand-held devices. Vigil can also be used in wired systems, but the focal point of our research is the security in dynamic, mobile systems. Vigil is designed so that clients can move, attach, detach, and re-attach at any point within the framework.

We have developed Vigil by complementing PKI [5] and Role Based Access Control [12] with trust management. Our work is similar to role-based access control-an approach in which access decisions are based on the roles that individual users have as part of an organization, such as doctor, nurse, manager, or student-in that a user's access rights are computed from its properties. However, our approach uses ontologies that include not just role hierarchies but any properties and constraints expressed in an XML based language including elements of both description logics and declarative rules. For example, there could a rule specifying that if a user in a meeting room is using the projector, he/she is probably the presenter and should be allowed to use the computer too. In this way, rights can be assigned dynamically to users without creating a new role.

There are six functional components within the Vigil architecture: Service Manager, Communication Manager, Certificate Controller, Security Agent, Role Assignment Manager, and Clients (users and services).

The Vigil system is divided into *SmartSpaces*, and each SmartSpace (from now on known as Space) is controlled by one or more *Service Managers*. The *Service Manager* finds matching services for users. It allows users and services to register and then provides brokering between them. The *Communication Manager*, provides a communication gateway between the Service Manager and the entities in the Space. Its sole purpose is to abstract and translate communications protocols and may contain several modules, one for each protocol [10]. The *Certificate Controller* is responsible for generating x.509 digital certificates [5] for entities in the system and for responding to certificate validation queries. There could be more than one Certificate Controller in an organization, but they all share common knowledge. The *Role Assignment Manager* maintains a role list for known entities in the system and a set of rules for role assignment. It responds to initial requests for role assignment in a particular Space. Each Space generally has one Role Assignment Manager to interpret local rules for appointment of roles within that Space. The *Security Agent*, manages the trust in the Space, receives information about new access rights that are conferred on a user and rights that are revoked, and reasons about the current rights of a user. Finally there are users and services that are treated equally as *Clients*.

All messages between the various entities in the Vigil system are in Centaurus Capability Markup Language (CCML) [10], which is an extension of Extensible Markup Language (XML) [17].

3.1 Service Manager

The Service Manager acts as a mediator between the Services and the users. All clients of the system, whether they are services or users, have to register with a Service Manager in the SmartSpace. The Service Manager is responsible for processing Client Registration/De-Registration requests, responding to registered Client requests for a listing of available services, for brokering Subscribe/Un-Subscribe and Command requests from users to services, and for sending service updates to all subscribed users whenever the state of a particular service is modified.

Service Managers are arranged in a tree-like hierarchy and form the core of the Vigil system. Service Managers are identified by their location in the tree or handles. This core can be used to route messages to foreign Service Managers through the hierarchy of Service Managers using the handle to determine where to forward each message.

All Clients depend on their Service Manager to enforce security and to broker requests for services. Consequently, each Client is only concerned with the trust relationship with its Service Manager. Service Managers in the tree establish trust relationships with each other. Consequently, trust between Clients is transitive through the Service Managers.

When a Service Manager starts up, it reads its handle, its parent's handle, its Role Assignment Manager's address, and the Vigil Certificate Controller's address, from a configuration file. The configuration file also specifies the handles of those Service Managers that it can connect to. In our implementation, handles are of the form "*umbc.edu*", "*cs.umbc.edu*", or "*lait.cs.umbc.edu*", etc. Each Service Manager has its own digital certificate and corresponding private key to sign messages.

3.2 Client

All clients must connect to and register with a Service Manager in a Space. During registration, the client transmits its digital certificate, a list of roles which can access it, and a flag, *ShowAll*, indicating whether or not the Service Manager should publish the client's presence to other clients, that do not have permission to access to it. If the client has no services to offer (is solely a user) the list of access roles is empty. During registration, a service can inform the Service Manager the level of security required. This reflects on how often the Service Manager updates access information about the service. To get new access information, the Service Manager queries the Security Agent. The Security Agent will return current information about delegations and revocations to the Service Manager, which updates its knowledge. The knowledge in a Service Manager is periodically erased as a way of allowing quicker propagation of revocations, as after deleting the rights the Service Manager has to ask the Security Agent for new trust information.

On receiving the registration information, the Service Manager verifies the client's certificate, and sends the client a copy of its digital certificate, which the client verifies. This handshaking procedure ensures a trust relationship between the Service Manager and the client. After the client certificate is verified, its certificate is sent to the Role Assignment Module, which decides which roles the client can have, based on rules in the policy. For example, the policy could state that managers from XYZ are developers in ABC. Once the set of roles is decided, the client has all the rights associated with the roles it is assigned. After a client is successfully registered, it is provided with an interface to all the services registered with the Service Manager and that it can access. The client is also shown all the services that it may not be able to access but that have their *ShowAll* flag set. The Service Manager sends an interface to all other Service Managers, that the Service Manager is currently aware of, the the newly registered client. Using the interface sent by the Service Manager, the client can request a service or request a list of services from a remote Service Manager. All requests from clients are authenticated by the Service Manager, which makes sure that the client has the right to access the requested services. The Service Manager queries the Security Agent. The Security Agent checks for role based access rights, prohibitions, permissions and delegations. If the request is authorized, the Service Manager forwarded to the service. A client can also *request permission* to access a service from another client or the service itself. If the client has the ability to delegate rights on the service, and if it is satisfied with the requester's credentials, it delegates this ability to the client for a certain period. While this delegation is valid the client is allowed to access the service. After the delegation expires the client needs to make another request for access.

3.3 Certificate Controller

The Vigil Certificate Controller is used to generate x.509 version 3 digital certificates [14] for entities of the system and verify certificates as well. To get a certificate , an entity sends a certificate request to the Certificate Controller. The entity is sent back a x.509 certificate, signed by the Certificate Controller and the Certificate Controller's self signed certificate, which is used to validate other entities' certificates. These certificates are stored and protected on a client's *smartcard*. An entity could enter a Space with a certificate from another Certificate Authority (different company). This certificate is verified by the Certificate Controller, based on a set of verification rules that specify which companies, Certificate Authorities and foreign entities can be trusted etc.

We use a simplified PKI (please refer to section 4 for more details), which precludes the use of an online repository for certificates, or for a Certificate Revocation List (CRL).

3.4 Role Assignment Manager

As the name suggests, the Role Assignment Manager is responsible for the assignment of roles to entities in a Space. The Role Assignment Manager maintains a list of roles associating entities with roles, and a set of rules for role assignment. These rules specify the credentials required to be in a certain role. When queried with the certificate of an entity, the Role Assignment Manager checks the access control list and the rules for assignment to find the roles of the entity. An entity could have more than one role at a time. For example, an entity could be both a *graduate student* and a *research assistant*. The role of an entity could change over time. Its access rights could also change without any change in role through the delegations of rights.

When the Role Assignment Manager is initialized, it reads its x.509 digital certificate and its PKCS#11 [11] wrapped private key from a secure file and stores it into local memory. It also reads and indexes the ACL file, which contains the roles of all entities within the system, and stores the time stamp of the file.

When the Role Assignment Manager receives a query for an entity's role, it compares the current time stamp on the capability file with the time stamp of the last file read, if they are not equal it re-reads the ACL file. This feature allows roles of entities to change continuously and dynamically.

3.5 Security Agent

The Security Agent is responsible for maintaining security and trust in the Vigil system. It enforces the security policy of the organization or Space. It interprets the policy to provide controlled access to Services and uses distributed trust as a more flexible and easily extensible policy based mechanism. There is generally a global policy associated with the organization and a local policy associated with a SmartSpace. All security agents in the organization will enforce the global policy and will additionally enforce a local policy, which is specific to the Space. A policy includes rules for role assignment, rules for access control, and rules for delegation and revocation.

The Security Agent uses a knowledge base and sophisticated reasoning techniques for security. On initialization, it reads the policy and stores it in a Prolog knowledge base. All requests are translated into Prolog, and the knowledge base is queried. The policy contains *permissions* which are access rights associated with roles, and *prohibitions* which are interpreted as negative access rights. The policy also contains rules for role assignments, access control and delegation. A user has the ability to access a service if the user has not been prohibited from accessing the service by an authorized entity and if it either has the role based access right or if some

authorized entity has delegated this right to it. An entity can only delegate an access right that it has the ability to delegate.

For example, a janitor may not have the right to delegate access to a workstation or a server, but a manager may. So, if Susan, a consultant from XYZ, wants to access a computer in ABC, she asks Mary, a manager from ABC, for permission. If Mary has the ability to delegate, she may or may not decide to delegate this right based on Susan's credentials. If Mary does delegate, Susan will be able to access the computer during the time that the delegation is valid. If Mary decides to leave her job or something similar, all her delegations will be invalid. If Susan now tries to access the computer, she will not be able to because the delegation is invalid. These rules about delegation of access rights, validity of delegations, etc. are part of the policy. There is more inference using these rules, than simple access control lists. It is also more than just using *Role Based Access Control*, because access rights can be delegated, and these delegations are not random, they have to be from an authorized entity to another entity or group of entities, and should follow the security policy.

When a user needs to access a service that it does not have the right to access, it requests another user, who has the right, or the service itself, for the permission to access the Service. If the entity requested does have the permission to delegate the access to the Service, the entity sends a delegate message, signed by its own private key, along with its certificate, to the Security Agent and the requester. The Security Agent checks the roles of the delegator and the delegatee and ensures that the delegator has the right to delegate, and that the delegation follows the security policy. It then adds the permission for the Client to access the Service, but sets a very short period of validity for the permission. Once this period is over, The Security Agent has to reprocess the delegation. This is very useful incase of revoked certificates, delegations or rights. If any one entity in the delegation chain loses the permission, then it is propagated down the chain very quickly, till everyone after the entity loses the ability. Everytime a Service Manager asks about the delegated rights of the client, the Security Agent sends back only valid permissions.

A user can also *revoke* rights that it has delegated by sending the appropriate message to the Security Agent. The Security Agent removes the permission for the delegated entity, and when a Service Manager asks about the delegated entity, it is informed of the revoked right. This causes revocations to progress rapidly through the system.

Figure 1 illustrates the working of the Vigil system.

4 Simplified PKI

In a PKI system, certificates are made available in an on-line repository. Consequently, when a user needs an entity's digital certificate it can be retrieved from such a repository and is valid as long as the certificate chain to the top level CA is verifiable. Similarly, in a typical PKI, the Certificate Authority also provides an on-line Certificate Revocation List (CRL), where inclusion in the CRL indicates that a given certificate is, for one of many possible reasons, invalid. However certificate repositories and Certificate Revocation Lists have a high administrative overhead. This overhead and the accompanying network traffic imposed by certificate acquisition and the signature verification is mitigated in Vigil by its simplified PKI framework.

On registration, all entities have to send their certificate to the Security Agent, which sends back its own certificate. They can verify the exchanged certificates themselves or send it to the local Certificate Controller. The Certificate has a list of CAs that it trusts and some rules for verification of certificates. If certificate is from a trusted CA, or satisfies one of the rules and has not been revoked, it is considered valid. Once verified, all signed messages

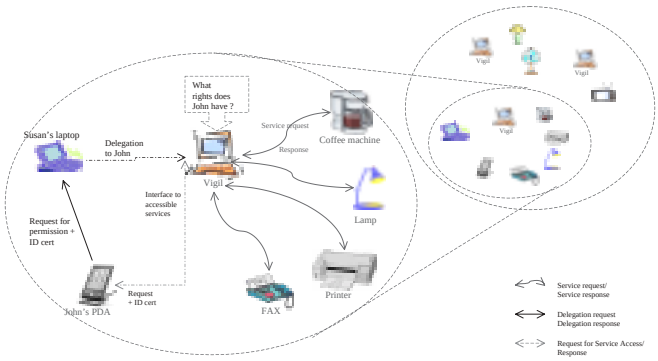

Figure 1: Overview of the Vigil system. There are several SmartSpaces in an organization. In every SmartSpace, a user uses the Vigil framework to gain access to services in that Space. A user can also request permission from another user to access a Service. Though Vigil has been conceptually shown as a central system, it is actually made up of distributed components.

between them can thus be verified. In a similar manner, communication between any two entities in Vigil can be handled securely, by attaching their certificates to the initial message.

Rather than use a CRL to signal a problem with an entity, the information is multicast to all Certificate Controllers in the organization, consequently invalidating the certificate. This precludes the necessity of maintaining a CRL, which must be signed by the Certificate Authority each time it is modified. However the revocations are still globally maintained.

This modified CRL and the fact that the certificates themselves are very small and can be carried easily in any mobile device, makes Vigil more adaptable to SmartSpaces.

5 Conclusion

After developing the initial infrastructure, we are working to improve the security and trust capabilities of Vigil. We are looking at distributed belief as a way for the Security Agent to garner the required trust information. The policy could include rules for belief as well. For example, the Security Agent would believe that Mark had the role of Manager in XYZ, if two registered and trusted clients say that it is true. Currently the trust information is encoded in Prolog, we believe using a semantic language like RDF [15] or DARPA Agent Markup Language (DAML) [1] instead will greatly benefit the system.

Vigil is aptly suited to pervasive environments because it uses a simplified PKI, the certificates are small enough to be carried on a mobile device, the domain is divided into SmartSpaces where policy is enforced at 2 levels, global and local, and distributed trust management is used as a way to handle foreign users and decentralized knowledge. Vigil uses existing security methods along with trust management for more flexibility in authentication and access control in pervasive computing environments.

References

[1] DAML specification. http://www.daml.org/.

[2] Orange and Unisys build the house that listens. http://www.unisys.com/news/releases/2001/feb/02127058.asp.

[3] George Candea and Armando Fox. Using Dynamic Mediation to Integrate COTS Entities in a Ubiquitous Computing Environment. In *Second International Symposium on Handheld and Ubiquitous Computing 2000*, 2000.

[4] Mike Esler, Jeffrey Hightower, Tom Anderson, and Gaetano Borriello. Next Century Challenges: Data-Centric Networking for Invisible Computing. In *Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom-99)*, 1999.

[5] The Internet Engineering Task Force. Public-Key Infrastructure (X.509) (pkix). http://www.ietf.org/html.charters/pkix-charter.html, 2002.

[6] Ian Goldberg, Steven D. Gribble, David Wagner, and Eric A. Brewer. The Ninja Jukebox. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS-99)*, 1999.

[7] Steven D. Gribble et al. The Ninja architecture for robust Internet-scale systems and services. *Computer Networks (Amsterdam, Netherlands: 1999)*, 2001.

[8] IETF. Simple Public Key Infrastructure (spki) Charter. http://www.ietf.org/html.charters/spkicharter.html.

[9] Lalana Kagal, Tim Finin, and Yun Peng. A Framework for Distributed Trust Management. In *Proceedings of IJCAI-01 Workshop on Autonomy, Delegation and Control*, 2001.

[10] Lalana Kagal, Vladimir Korolev, Sasikanth Avancha, Anupam Joshi, Timothy Finin, and Yelena Yesha. Highly Adaptable Infrastructure for Service Discovery and Management in Ubiquitous Computing. *To appear in ACM Wireless Networks (WINET) journal*, 2002.

[11] RSA Laboratories. PKCS 11-Cryptographic Token Interface Standard. 1994.

[12] E. Lupu and M. Sloman. A Policy Based Role Object Model. *1st IEEE International Enterprise Distributed Object Computing Workshop (EDOC'97)*, 1997.

[13] M.Blaze, J.Feigenbaum, and J.Lacy. Decentralized trust management. *IEEE Proceedings of the 17th Symposium*, 1996.

[14] W. Polk D. Solo R. Housley, W. Ford. RFC 2459 Internet X.509 Public Key Infrastructure Certificate and CRL Profile. 1999.

[15] RDF. Resource Description Framework (RDF) Schema Specification. W3C Proposed Recommendation, March 1999, 1999.

[16] Jefferey Undercoffer, Andrej Cedilnik, Filip Perich, Lalana Kagal, and Anupam Joshi. A Secure Infrastructure for Service Discovery and Management in Pervasive Computing. *ACM MONET : The Journal of Special Issues on Mobility of Systems, Users, Data and Computing*, 2002.

[17] W3C. Extensible Markup Language. http://www.w3c.org/XML/.

[18] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995.