iSecurity: A Security Framework for Interactive Workspaces

Yee Jiun Song, Wendy Tobagus, Der Yao Leong, Brad Johanson, Armando Fox
{yeejiun@, tobagus@, deryao@, bjohanso@graphics, fox@cs}.stanford.edu

Stanford University

September 3, 2003

Abstract

Security is an important and open issue in ubiquitous computing. Recent work has focused either on developing models of security suitable for various ubiquitous computing environments, or on the coupling of context-awareness with traditional security mechanisms to provide context-aware security. In this paper, we examine a specific instance of ubiquitous computing — interactive workspaces — and propose a security framework, iSecurity, for this environment. In particular, we investigate iROS-based interactive workspaces and exploit the fact that all communications in these environments have to pass through the Event Heap. We differ from existing work in ubicomp by distributing security policy enforcement to individual client applications while keeping the responsibility of authentication on a central server. This decoupling of authentication and policy enforcement provides a powerful framework that supports a dynamically changing set of applications and users. While security mechanisms incur a performance cost, our system is capable of meeting the human level performance needs of an interactive workspace by offering a round-trip latency of less than 100 ms with normal workloads. iSecurity is fully backwards compatible with existing Event Heap applications, allowing its deployment into iROS-based interactive workspaces with minimal disruption to their daily operation.

1 Introduction

The increasing prevalence of ubiquitous computing environments naturally leads to a greater focus on security issues [1]. In response, recent work has focused on developing security models suitable for various ubiquitous computing environments or marrying traditional security mechanisms to ubiquitous computing technology to enable varying degrees of security.

Most security systems developed for ubiquitous computing environments are based on a central trusted entity that enforces both authentication and security policies [2, 3]. Such designs intrinsically depend on the system having some knowledge of the entire set of services that will be available, and assume that it is desirable for the central server to be responsible for both defining and enforcing security policy. Other systems have focused on utilizing ubiquitous computing technology to enable context-aware security [2, 4].

In this paper, we examine the security requirements of a particular type of ubiquitous environment, iROS-based interactive workspaces [5]. We present a security framework, iSecurity, for iROS-based workspaces, that supports a dynamically changing set of heterogenous service providers and service consumers. The system does not assume any prior knowledge of the set of applications, and thus places no

constraints on them. Yet, it provides a simple and easily understood framework that allows a flexible range of security policies to be defined by each service provider.

The rest of this paper is organized as follows: section 2 gives a brief description of the kinds of environment for which iSecurity is designed, and the existing infrastructure upon which we are building; section 3 articulates the objectives of our design; section 4 discusses the decoupling of authentication and security policy enforcement, which is central to our design; section 5 describes our overall system architecture, and how the different components interact; section 6 describes the implementation details of the system; section 7 describes the operation of the system; section 8 summarizes the results of this paper; section 9 describes future work; finally, section 10 compares our system with related work.

2 Background

The iSecurity system is designed specifically for the sub-domain of ubiquitous environments typified by iROS-based interactive workspaces [6]. The Stanford Interactive Workspaces project explores the augmenting of meeting rooms with technology, creating an environment suitable for collaboration. iROS-based workspace deployments vary significantly, but they all contain large shared displays, and support a myriad of portable devices and applications, interacting to provide users with the services that they need.

An example of such a workspace is the iRoom located in Stanford University [5]. The iRoom has three touch-sensitive large screen displays, each of which is controlled by a dedicated computer. These computers can be controlled via a of keyboard and mouse using the PointRight software [7]. The iRoom also provides wireless LAN support, a digital camera, microphones, and a selection of other interactive devices. Additionally, users are able to manipulate the iRoom computers with PointRight installed on their personal devices. Users are also able to push and pull webpages from one device to another using the MultiBrowse software [8]. The iRoom is typically used during interactive project meetings, where the seamless communication between the iRoom's infrastructure and other personal devices enhances the team's working experience and promotes collaboration.

The underlying infrastructure that allows different devices and applications to work together in iROS-based workspaces is called the Event Heap [9]. The Event Heap is a central message repository residing in each interactive workspace. All applications in the interactive workspace communicate by posting and retrieving messages to and from the Event Heap. All messages are public, and while applications can choose to listen only to messages that meet a certain criteria, there is no way to guarantee that a message be received only by a particular recipient. Each iROS-based interactive workspace has its own Event Heap, and is isolated from other workspaces.

3 Design Goals

3.1 Problem Description

Ubiquitous computing environments such as interactive workspaces have different security objectives from traditional systems. The requirements for traditional security infrastructures can be grouped under four major headings [10]:

1. Confidentiality: Controlling access to information.
2. Integrity: Ensuring that resources and information are exchanged and used in a specified and authorized manner.
3. Availability: Ensuring prompt and continued access to information and resources.
4. Accountability: Keeping track of the access to and usage of information and resources.

While these requirements continue to be important in interactive workspaces, they are insufficient. Kindberg and Fox define two key characteristics of ubiquitous computing systems - physical integration and spontaneous inter-operation, from which they derive two principles - the boundary principle and the volatility principle - which underlie ubiquitous systems [11]. The boundary principle states that ubiquitous computing environments should be divided into regions that have clearly specified scope; the volatility principle states that ubiquitous systems should be designed with the assumption that the set of users, hardware, and software is highly dynamic and unpredictable. These characteristics of ubiquitous systems have implications on security, in the areas of trust management, access control, location authentication, and resource sharing.

More specifically, Johanson characterizes the interactive workspace environment as follows [12]:

1. Bounded environment.
2. Human centered interaction and flexible reconfiguration.
3. Human level performance needs.
4. Heterogeneity of hardware and software.
5. Changing environment.

Traditional security systems do not meet the demands of the above characteristics. Indeed, traditional security principles even seem to be in direct conflict with the transparent, open, and dynamic environment that the interactive workspace supports. As such, we create an augmented security model that integrates the traditional security requirements with the requirements of the interactive workspace environment.

3.2 Scenario

In this section, we present a motivating example for iSecurity. Imagine a three-party negotiation between two competing companies and their potential client. The negotiation takes place in the client's office, which is an interactive workspace. Each company is represented by a team, whose team members bring their own laptops which are connected to the interactive workspace. There are two large display screens in the room, and each team is given permission to use one. The client sets up the environment such that any member of a negotiating team is able to control the large display screen assigned to the team from their own laptops but not the other screen, which is reserved for the opposing team. In addition, only the client has the ability to control both displays.

During the negotiation, both teams utilize a diverse range of applications running in a coordinated fashion - for example, a design change in a CAD program should automatically update a spreadsheet

application running elsewhere the room in order to show the potential impact in financial terms. When the client proposes a potential change, the negotiating teams should be able to effect the change and show the immediate impact the change has on other related parts. In addition, the client has his own spreadsheet program which obtains information from both teams' CAD programs and derives his own estimates, which are privy to him. Despite this sophisticated set up, the negotiating parties' programs should never be able to interfere with the other team's applications. Intra-team communication must be secure and neither team should be able to eavesdrop on the other. In this way, during the negotiation both teams work closely together with themselves without compromising their own positions to the opposing team.

3.3 Objectives

The scenario in the previous section illustrates the fact that a well-designed security model for the interactive workspaces environment needs to have a balance between traditional security principles and the interactivity, flexibility and dynamism required by such an environment. In addition, there is a need to account for the implicit trust relationships which exist among people who meet and interact physically in the interactive workspaces environment.

Combining the objectives of a traditional security model [10] and the characteristics of the interactive workspaces environment [12], we outline the objectives of our system as follows:

1. Integrity: Messages between different parties must be tamper proof, and any message must have an authenticated source.

2. Privacy: All data accesses should be controlled. Messages intended for a specific recipient should only be seen by that recipient.

3. Availability: The system must be able to protect against and recover from a damaging event, thus providing some reasonable guarantees on the access of information and resources.

4. Accountability: In order to prevent and detect security attacks, the system must provide mechanisms for keeping track of all activity. In the case of a security breach, this provides a way of tracking the source of the breach and recovery of lost data.

5. Customizability: The security system for interactive workspaces must be highly customizable and flexible, since the environment is dynamic and consists of a diverse set of heterogenous components.

6. Convenience: The system needs to be simple and unobtrusive, so as not to intrude upon users' activities. The customizability objective compliments the convenience objective as the security systems should be configurable to suit the security needs of the users.

Note that the first four objectives outlined here coincide with those of traditional security systems, while the last two objectives are designed to support the characteristics of the interactive workspaces environment.

4 Authentication and Security Policy

Central to the iSecurity system is the decoupling of authentication and security policy. As noted by Needham and Schroeder [13], the basis of any authenticated communication is a trusted authoritative source of information. This lends to security system designs that are built around a central server, which handles both authentication and the storage and enforcement of security policies. Such a model fits awkwardly into a dynamically changing environment of heterogenous services where the server might not be aware of all the services that are available, and the types of access control they support.

Kindberg and Fox suggest that the use of capabilities fits more naturally into ubiquitous computing environments, because they reduce the need for system-wide configuration and avoid communication with a central server [11]. However, capabilities alone do not meet all the objectives of our security model. As mentioned earlier, accountability is one of the objectives that has to be satisfied by our system. Capabilities do not support traceability [14], and thus fail to meet the accountability objective.

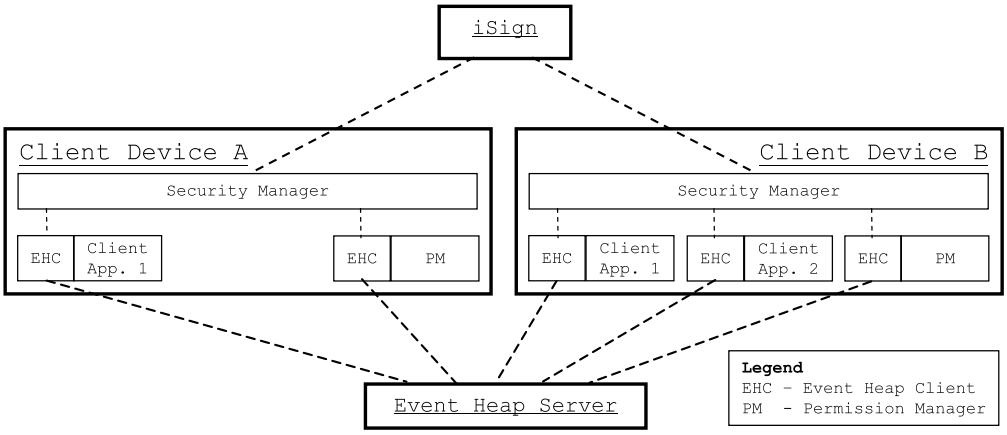
The problem in deciding between a traditional ACL-based system with a trusted central server and one that is based on capabilities reduces to that of deciding where to locate the security mechanisms. In a traditional system, all security enforcement is done on the server. All client service providers have to check with the server before any action is performed. In environments such as interactive workspaces, where the environment is made up of a heterogenous collection of service providers and consumers that is constantly changing, such a design is inadequate. Ideally, one would want to push security mechanisms to the clients, so that the central bottleneck can be avoided. Capabilities partially achieve this objective by pushing out the permission verification mechanism out to the clients, such that clients do not need to contact the server when presented with a capability [15].

Authentication and security policy are two distinct and orthogonal issues. Extending the notion of decentralizing security mechanisms, it follows that only authentication needs to be done at the server. The storing and enforcing of security policies, in the form of ACLs or otherwise, can be distributed to the clients. Not only does this reduce reliance on the central server, it also decouples client security policies from the authentication mechanism, and indeed, from the server's security system entirely. This also isolates the server from the changing environment in which it operates and facilitates the introduction of new clients into the environment.

5 System Design

The design of the iSecurity system is aimed at meeting the objectives mentioned in the previous sections while maintaining backwards compatibility with existing Event Heap applications. Figure 1 shows the components of the iSecurity system. Four separate machines are represented in figure 1. Device A and device B are client machines. These may be permanently placed in the interactive workspace, or may be portable devices, such as laptops or PDAs, that are brought in and out of the interactive workspace. The event heap server runs on a dedicated server machine, and serves only the interactive workspace in which it is located. A separate machine hosts iSign, the trusted certificate authority. A wide range of applications runs on each client machine, and these applications may be consumers or providers of services, or may be both. Each client application communicates with the rest of the system via an Event Heap Client object. A special client application, called the Permission Manager, runs on each client

Figure 1: iSecurity architecture



device. A Security Manager is also installed on each client device.

5.1 iSign

iSign is the trusted certificate authority in the iSecurity system. It serves as a certificate store and provides the basis of secure communication in the iSecurity system. Each user in the system must be registered with iSign. This allows users to retrieve certificates from iSign that enable them to identify themselves to other users in the system. In figure 1, iSign is shown to be running outside of the Event Heap Server machine. This implies that several interactive workspaces are able to share an iSign server, and thus share a user namespace. This allows an organization to have a universal namespace across multiple interactive workspaces, which is a useful feature since one organization can frequently have more than one interactive workspace.

5.2 Security Manager

The Security Manager on each client device is responsible for fetching a certificate from iSign and using it to generate new certificates to each client application. The first time a client application requests a certificate from the Security Manager causes the Security Manager to prompt the human user with an authentication mechanism. The authentication mechanism used can vary depending on the needs of the particular interactive workspace, from a simple login/password challenge to biometric scans if necessary. The input is presented to iSign, which then returns a certificate to the Security Manager if the authentication is successful. Since we only permit one instance of the Security Manager to run on a device, only one user can be logged into a device at any one time.

5.3 Client Application

Each client application running on client devices communicates with the system using the Event Heap Client object. When an application starts, it instantiates an Event Heap Client object, which in turn requests a certificate from the Security Manager. Once it receives the certificate, the Event Heap Client

then uses it to establish a secure direct connection to the Event Heap Server. This allows the Event Heap Server to securely identify the user. Once the connection has been established, the client application can then securely post and retrieve messages to and from the Event Heap Server via the Event Heap Client object's methods. Note that from a client application developer's point of view, communication with the Event Heap Server is almost exactly the same, with or without iSecurity - the only difference being that a secure Event Heap application uses a secure API which is almost identical to the non-secure API that exists in the original Event Heap system.

Each client application is responsible for maintaining its own security policy. Upon receiving a service request, a client application checks the source of the request against its security policy and determines whether or not to fulfill the request. In this way, the Event Heap Server is isolated from the dynamically changing set of services that may be available in the interactive workspace. This also allows client applications complete flexibility in designing their own security policy. For example, it is now possible for client applications to define complex application specific security policies such as "Users are only allowed to turn on the high resolution displays if there are three or more users logged into the system".

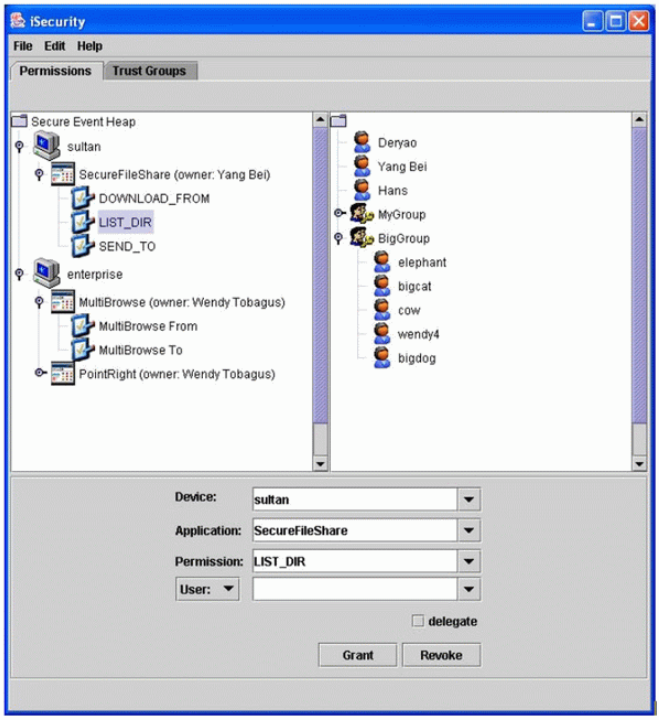
Although each application is responsible for maintaining its own security policy, we observe that many applications will have the same kind of security policy enforcement needs. To ease the development of secure applications, iSecurity provides an ACLManager library that allows client applications to conveniently create and manage access control lists (ACLs). Each entry of an ACL is a `<permission, user>` tuple. The ACLManager allows applications to add/remove entries from the ACLs and to check service requests against the ACLs to determine if a service request is in accordance with the application's security policy, without burdening the application programmers with the responsibility of maintaining the necessary data structures.

5.4 Event Heap Server

All communication between client applications go through the Event Heap Server. The Event Heap Server identifies each client application by the iSign certificate that is presented to it during connection. The Event Heap Server guarantees that messages are delivered only to designated recipients, and also ensures the authenticity of the source of messages. Aside from ensuring the integrity of messages and the authenticity of their senders, the Event Heap Server is also responsible for maintaining trust group information. Trust groups are simply lists of users that can be used in specifying ACLs. They are introduced to simplify security policy definition and management. As mentioned earlier, in addition to `<permission, user>` tuples, ACLs can also contain `<permission, trustgroup>` tuples. Client applications can create/manage trust groups by calling on certain methods of the Event Heap Client object. They can also check trust group memberships in the same way. Trust groups are stored on the Event Heap Server, and client applications make queries of trust group membership by sending special messages to the Event Heap Server.

Although each application manages its own security policy, iSecurity provides a Permission Manager that makes it possible for users to manage permissions of their applications, regardless of the machine they are on, in a single unified interface. Figure 2 shows a screenshot of the Permission Manager. The Permission Manager is a special Event Heap client application. It resides on a client device and provides security policy management capabilities. The Permission Manager broadcasts a request to all

Figure 2: Permission Manager



applications running in the interactive workspace to reply with a list of their ACLs. If an application uses the ACLManager to maintain its security policy, the ACLManager will automatically make an appropriate response. Otherwise, it is up to the application programmer to decide whether or not to respond to such a request. Figure 2 shows a situation where there are two machines in the interactive workspace: `sultan` and `enterprise`. The selected permission list is the `LIST_DIR` list of the SecureFileShare application. The pane on the right shows the list of users and groups who have the `LIST_DIR` permission, which in this case, means that they are able to look at the files that are being shared on `sultan`.

Besides displaying the contents of the ACLs on all relevant applications to the user, the Permission Manager also allows users to manage these permissions, by sending a permission change request to the appropriate application for the user. Again, if the client application uses the ACLManager, these permission change requests will be automatically fulfilled if the requester has the appropriate rights, otherwise it is up to the application programmer to decide whether or not to fulfill the request. Lastly, the Permission Manager allows users to manage create, delete, and modify trust groups.

6 Implementation

We implemented the iSecurity system by augmenting the existing Event Heap software [9]. All the pieces of software were written in Java. In the previous section, we mentioned that iSign runs on a dedicated machine. In our implementation, for convenience, we installed iSign on the same machine that hosts the Event Heap Server. iSign keeps track of all registered users in an internal database. It provides each client device with a certificate that uniquely identifies the human user that is logged into the client device.

This certificate is then used to establish a secure connection to the Event Heap Server in the interactive workspace. All certificates in the iSecurity system are X.509 certificates signed with a 1024 bit key. All secure communication in the system is done over SSL 3.0 connections, using the certificates provided by iSign. Thus, the Event Heap Server is able to identify the human user that is associated with every connection, and hence, every application that is connected to it.

The Event Heap Client object is the core component that applications use to communicate with the system. When a client application starts, it instantiates an Event Heap Client object that requests a certificate from the Security Manager. If this is the first client application to request a cerificate on that machine for that session, the Security Manager prompts the user for a login/password pair and presents it to iSign, to retrieve an iSign certificate. This is the only authentication mechanism currently implemented. The Security Manager then uses the iSign certificate to certify the client application. A certificate chain consisting of the client application certificate and the iSign certificate is then used by the client application to connect to the Server. In our implementation, certificates are invalidated when the user logs out from iSecurity. If a user forgets to log out, the certificates expire after a pre-defined time period. In our implementation, this is set to be twenty-four hours.

As mentioned in the Section 5, each client application is responsible for maintaining its own security policy. However, we provide an ACLManager library which provides basic security policy management functionality. This allows the client application to create and manage access control lists, and automatically queries for memberships in the ACLs.

The Permission Manager is implemented as an event heap client application. It sends special system messages to other event heap client applications to allow users to monitor/manage the security policies of all the applications running in an interactive workspace from a unified interface. Note that these system messages are themselves sent securely, and protected by the same security mechanisms as all messages in the iSecurity system. In order for applications to work with the Permission Manager, they have to listen to these system messages and respond accordingly. For applications that use the ACLManager library, this is handled automatically.

The Event Heap Server is the hub of all activity taking place in the interactive workspace. As mentioned earlier, in the Event Heap infrastructure, all messages are posted to and retrieved from the Event Heap Server. We took the current implementation of the server as detailed in [5] and extended it to cope with the security requirements.

In particular, these are the key changes that we made to the existing implementation of the Event Heap Server:

1. Added the ability to maintain and store Trust Group information.
2. Communications with secure clients are all done through SSL 3.0 connections, and secure connections only accepted upon presentation of valid certificates.
3. Events sent by secure clients are stamped with the sender's identity as provided by the certificate.
4. Before secure events are retrieved by anyone, the identity of the user retrieving the events is verified to ensure that he has the right to retrieve them. If not, the events will not be returned to him. The verification process might involve checking the trust group information if the event is targeted to a particular trust group as oppose to a particular user.

These changes allows the iSecurity system to make the following guarantees to the application programmer:

1. If a recipient is specified, a message will be delivered only to that recipient.
2. The sender of messages, as indicated by the source field of messages, is guaranteed to be an authenticated user of the system.
3. Messages are tamper proof.
4. Messages cannot be eavesdropped upon.
5. Messages are not susceptible to replay attacks.

Note that the last three guarantees are a result of iSecurity's use of SSL [16].

7 Operation

In this section we will present a walkthrough of the events occurring in the system for a typical interactive session using a simplified scenario. There are two users, Alice and Bob, and each of them brought their own laptops into the interactive room. There is one room computer controlling a large screen display in the room. Alice is a registered user of the room and already has access to certain resources in the room, while Bob is her guest and has no access to any resource in the room.

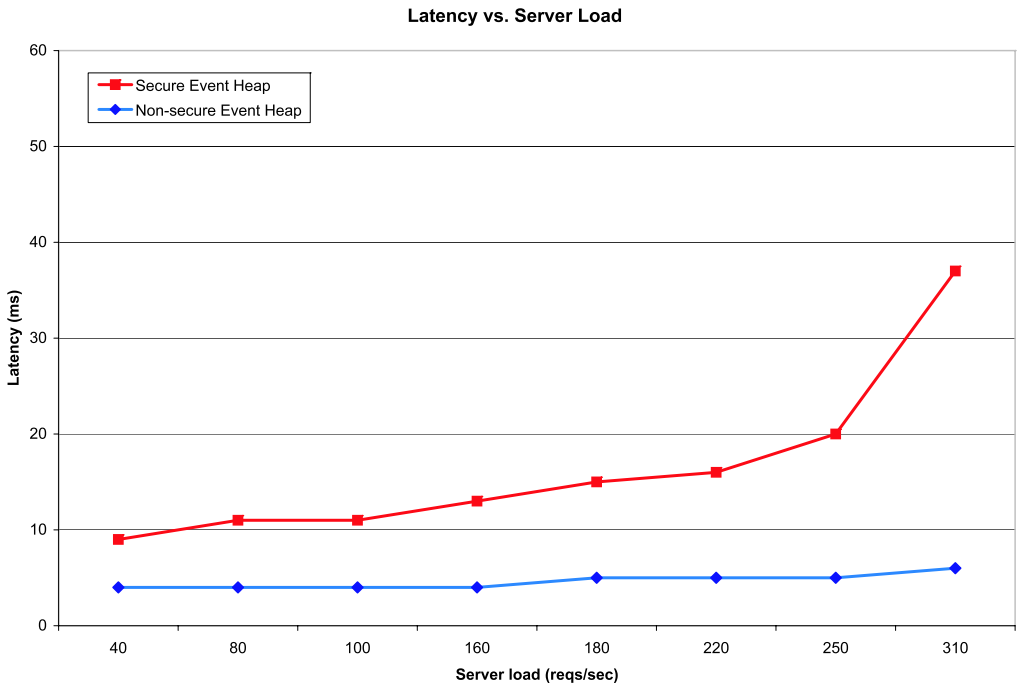
The first thing that both users do is authenticate themselves to their own laptops. The authentication mechanism used in our present implementation is a simple login/password challenge, but other types of mechanisms like smart card or biometric scans can be implemented. This step assumes that the users have already been previously registered with iSign.

During the course of the meeting, when Bob tries to use his laptop's mouse and keyboard to control the room computer using PointRight [7], he fails. When the PointRight service running on the room computer receives Bob's attempt at access, it checks Bob's identity against all the entries in its access control list. PointRight finds that Bob is not authorized to use the room computer, and rejects his request.

Bob then requests Alice to delegate to him the authority to access the PointRight on the room computer. Alice launches the Permission Manager on her laptop and uses it to send instructions to the room computer to add Bob's name to its access control list. The PointRight service on the room computer receives the instructions from Alice, verifies that Alice has the right to delegate such authority, and executes the instructions accordingly by adding Bob's name to its access control list. The next time Bob tries to access the room computer's PointRight service, his request is granted, and he is able to control the machine from his laptop.

Due to space limitations, we do not go through the more complex example presented in 3.2, but the concepts are similar, and we leave it to the reader to extend this simple example to more complex situations.

Figure 3: Latency under varying workload



8 Evaluation

8.1 Performance

One of the requirements of the interactive workspace is human level performance [12]. To measure the impact of iSecurity on the performance of the Event Heap, we measured the latency of messages under varying server loads. Since the Event Heap was designed to handle only the load within a single physical location, and the number of human users and devices within such a space is limited, we aim only to achieve latencies of less than 100 ms with a workload in the order of hundreds of events per second. For the duration of the tests, the server was running on a P4 1.5GHz computer with 128MB of RAM. The server load is distributed among 20 clients, and latency is measured from the time a client posts a message until the time it retrieves the same message. All 20 clients are running on one P3 1Ghz machine with 256MB of RAM. Readings are taken after the system achieves steady state, and we plot the message latency versus the server load, as measured by the requests per second being handled by the server.

Figure 3 shows that although iSecurity does cause a significant performance hit on the system, the latency of system remains well below 100 ms under normal workloads. However, the graph shows that as the workload increases, the performance of the secure system degrades much faster than that of the original system. It is possible that the performance dip is due to client side saturation of having to maintain 20 active JVMs and SSL connections on a single machine, and we intend to repeat the experiment using multiple client machines for the camera-ready version of this paper.

A more serious performance hit is encountered during the connection between the client application and the server. The time needed to set up the SSL connection is approximately 4-5 seconds, which is

an order of magnitude larger than the time needed for normal TCP connection used in the unsecured version. Fortunately this overhead is only incurred once each time an application connects to the event heap server.

8.2 Ease of Client Application Development

In order to assess the functionality and usability of iSecurity, we developed a secure application using the iSecurity framework and ported an existing application. The application we developed was Secure File Share, a secure file sharing application that allows users in an interactive workspace to conveniently transfer files from one device onto another. This application makes use of the ACLManager to manage three permission lists: `LIST_DIR`, `SEND_TO`, and `DOWNLOAD_FROM`. These permissions allow users on one device to list the files being shared on a particular device, send files to that device, and download files from that device, respectively. Since the ACLManager is used, the Permission Manager is automatically able to display Secure File Share's permission lists, and allow the user to use the unified graphical user interface, as shown in figure 2, to manage its permissions. The development of Secure File Share took a single graduate student three days. This demonstrates the ease of developing secure applications for the iSecurity framework.

The iSecurity system is fully backwards compatible with existing applications of the interactive workspaces project. We have deployed iSecurity in the iRoom, a prototype interactive workspace, and verified that all existing applications function correctly. However, we do not in general allow communication between existing insecure applications and secure applications. The only exception is where secure applications explicitly request access to the old insecure function calls, to post and receive insecure messages. One of the goals of our implementation was to make it as easy as possible to port existing applications over to the iSecurity system. To illustrate this, we ported an existing application, the Event Logger, to use the iSecurity system. The Event Logger is a simple yet flexible application that allows anyone to selectively log relevant messages passing through the Event Heap to a file. A user can specify the types of messages to log, and which particular fields of the messages to log. We adapted the Event Logger application to run as a secure application on iSecurity by modifying only *nine lines* of code. This demonstrates the ease of porting existing application to the new security infrastructure. The iSecurity-enabled Event Logger allows users to log only events for which they have permissions to see.

9 Future Work

Several applications exist in the Interactive Workspaces project that have yet to be modified to take advantage of the iSecurity system. In particular, we plan to port PointRight and MultiBrowse [7, 8], two of the most used applications in the iRoom, to take advantage of iSecurity features. Having a suite of secure applications will then allow us to test iSecurity in other real world deployments of the iROS-based workspaces. We plan to test and evaluate the iSecurity system in the interactive workspaces deployments on various locations on our campus. This will allow us to study how users interact with the security system, and whether a security system such as iSecurity interferes significantly with users' ability to be productive in an interactive workspace.

In our design of iSecurity, we have sacrificed performance for simplicity. Our performance tests

have shown that iSecurity performs adequately under typical workloads. We hope to re-evaluate the performance of the system under more intensive workloads with multiple secure applications running simultaneously. Performance enhancements in the form of caching can be added to the system if proven necessary. In particular, trust group information which is being stored on the event heap server can be cached on client devices to significantly reduce the amount of network traffic in the system.

10 Related Work

Undercoffer et al's work in Centaurus2's security system [17] is comparable to iSecurity. Centaurus2 differs from iSecurity in that the Centaurus2 ubiquitous computing system uses a hierarchy of service managers to enforce security, access control, and broker request for services. As such, security managers need to be aware of and understand the security policies of the clients. The Centaurus2 system restricts applications to a rigid set of access rights to make this possible. iSecurity avoids this problem by moving the responsibility of security policy enforcement to the client applications, so that the system need not be aware of individual client applications' security policies.

Cerberus is a ubiquitous security system for Gaia, a middleware infrastruture that focuses on supporting the development of applications for active spaces [2, 18]. Cerberus focuses on context awareness and uses a central inference engine to enforce security policies. Cerberus' inference engine provides a powerful and flexible way of defining security policies that exceeds iSecurity's approach. However, this approach requires that the inference engine be aware of the security policies of all services in the system. This is a significantly different approach from iSecurity and is not suitable for the kind of dynamically changing environment for which iSecurity is designed.

Covington et al designed a context-aware security architecture for ubiquitous environments that specifically addresses the needs of a residential computing infrastructure [3]. In this system, security policy is defined by a generalized policy definition language [19] and enforced by a central Authorization Service.

In [4], Robinson and Beigl propose a theoretical model for security in context-aware environments which is not identity-based but rather, based on the overall context of the environment. Enforcement of security policies is based on the present context of the environment, so a user might be able to do certain thing during certain times and not others. Environmental resources are governed by administrator, while local entities are responsible for managing local policies.

Langheinrich's work on privacy awareness in ubicomp environments [20] focuses on privacy. Rather then a system that makes security and privacy guarantees, Langheinrich argues that the perfect protection of privacy is impossible and proposes a system that allows data collectors to announce and implement data collection policies, and provide users with the means of knowing the kinds of data that are being collected about them.

11 Conclusion

In this paper, we studied the security needs of a particular type of ubiquitous computing environment, iROS-based interactive workspaces, and articulated objectives that any security system for such an environment should aim to have. Interactive workspaces supports a dynamically changing set of users and ap-

plications, and thus presents a significant security system design challenge. We designed and implemented iSecurity, a security framework for iROS-based workspaces, that provides the basic security mechanisms needed by application programmers to develop secure applications. By decoupling authentication from security policy enforcement, iSecurity isolates the central server from the changing set of applications that operate in the system, while allowing the server to provide authentication services to clients. Security policy enforcement is distributed to individual client applications, allowing different client applications to have vastly different security policies. We demonstrated that iSecurity performs sufficiently well to meet human level performance requirements. We illustrated the ease of application development through the development of the Secure File Share application, as well as the adaptation of the Event Logger. We have deployed the iSecurity system in the iRoom, a prototype interactive workspace environment, and demonstrated its backwards compatibility with existing applications. We believe that iSecurity provides a simple yet powerful platform for secure application development in interactive workspaces, and hope to ascertain users' acceptance of the system in future work.

12 Acknowledgements

This work started as part of a joint course between Stanford University and the Swedish Royal Institute of Technology (KTH). We would like to thank Hans Andersson, Sherif Yousef, and Yang Bei of KTH for their effort in the initial development of the iSecurity system.

References

[1] Marc Langheinrich. Privacy by design – principles of privacy-aware ubiquitous systems. In G.D. Abowd, B. Brumitt, and S. Shafer, editors, *Ubicomp 2001 Proceedings*, volume 2201 of *Lecture Notes in Computer Science*, pages 273–291. Springer, 2001.

[2] Jalal Al-Muhtadi, Anand Ranganathan, Roy Campbell, and M. Dennis Mickunas. Cerberus: A context-aware security scheme for smart spaces.

[3] M.J. Covington, P. Fogla, Zhiyuan Zhan, and M. Ahamad. A context-aware security architecture for emerging applications. In *Eighteenth Annual Computer Security Applications Conference, 9-13 Dec. 2002, Las Vegas, NV, USA*, pages p.249–58. Los Alamitos, CA, USA : IEEE Comput. Soc, 2002.

[4] Philip Robinson and Michael Beigl. Trust context spaces: An infrastructure for pervasive security in context-aware environments. In *Proceedings of the First International Conference on Security in Pervasive Computing*, 2003.

[5] B. Johanson, A. Fox, and T. Winograd. The interactive workspaces project: experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, 1(2):67 – 74, April 2002.

[6] Shankar R. Ponnekanti, Brad Johanson, Emre Kiciman, and Armando Fox. Portability, extensibility and robustness in iros. In *First IEEE International Conference on Pervasive Computing*, 2003.

[7] Brad Johanson, Greg Hutchins, Terry Winograd, and Maureen Stone. Pointright: experience with flexible input redirection in interactive workspaces. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 227–234. ACM Press, 2002.

[8] Brad Johanson, Shankar R. Ponnekanti, Caesar Sengupta, and Armando Fox. Multibrowsing: Moving web content across multiple displays. In *Technical Note in UBICOMP 2001*, 2001.

[9] Brad Johanson and Armando Fox. The event heap: a coordination infrastructure for interactive workspaces. In *Fourth IEEE Workshop on Mobile Computing Systems and Applications, 20-21 June 2002, Callicoon, NY, USA*, pages p.83–93. Los Alamitos, CA, USA : IEEE Comput. Soc, 2002, 2002.

[10] National Research Council. *Computers at risk: safe computing in the information age.* National Academy Press, 1991.

[11] T. Kindberg and A. Fox. System software for ubiquitous computing. *IEEE Pervasive Computing*, 1(1):70 – 81, January 2002.

[12] Bradley Earl Johanson. *Application coordination infrastructure for ubiquitous computing rooms.* PhD thesis, Stanford University, 2002.

[13] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[14] Li Gong. A secure identity-based capability system. In *IEEE Symposium on Security and Privacy*, pages 56–65, 1989.

[15] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz. An architecture for a secure service discovery service. In *Mobile Computing and Networking*, pages 24–35, 1999.

[16] David Wagner and Bruce Schneier. Analysis of the ssl 3.0 protocol. In *The Second USENIX Workshop on Electronic Commerce*, pages 29–40, 1996.

[17] J. Undercoffer, F. Perich, A. Cedilnik, L. Kagal, and A. Joshi. A secure infrastructure for service discovery and access in pervasive computing. *Mobile Networks and Applications*, 8(2):113 – 25, 2003.

[18] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74 – 83, October 2002.

[19] Matthew J. Moyer and Mustaque Ahamad. Generalized role based access control. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS), Mesa, Arizona, USA*, 2001.

[20] Marc Langheinrich. A privacy awareness system for ubiquitous computing environments. In *4th International Conference on Ubiquitous Computing*, pages 237–245, 2002.