A Middleware Architecture for Securing Ubiquitous Computing Cyber Infrastructures

Raquel Hill, Jalal Al-Muhtadi, Roy Campbell, Apu Kapadia, Prasad Naldurg, Anand Ranganathan
Department of Computer Science, University of Illinois, Urbana-Champaign
(rlhill@uiuc.edu, almuhtad@cs.uiuc.edu , rhc@uiuc.edu, akapadia@uiuc.edu, naldurg@cs.uiuc.edu,
ranganat@students.uiuc.edu)

Abstract

Ubiquitous computing is revolutionizing the way applications, users, resources, and physical spaces interact. In this paper we address securing cyber infrastructures for ubiquitous computing environments, like smart buildings and campuses. Our emphasis here is to construct a middleware-based critical cyber infrastructure (CCI) that encompasses heterogeneous components and binds networks, processors, and devices with mechanisms, protocols and services to offer reliable, fault-tolerant, available, and secure operations. Existing CCI implementations create statically configured, confined networked subsystems, which are isolated from the public Internet, and are context insensitive. This leads to multiple subsystems that are incompatible and incapable of interoperating, thus making operations, management, and trust difficult.

In this paper, we propose Hestia: Heterogeneous Survivable Trusted Information-assurance Architecture and describe how it addresses the problem of securing critical information services in large-scale ubiquitous computing environments. Hestia is a novel programmable middleware solution, implemented as a network of Middleboxes. These Middleboxes form protective layers that isolate critical cyber-infrastructure services and mediate authorized access to the services in our system. The Middleboxes provide a programmable distributed object-oriented framework that enables us to integrate security, privacy, and reliability mechanisms into service access interfaces and implementations.

1. Introduction

Ubiquitous computing allows the coupling of the physical world to the information world. Ubiquitous environments organize networked computer devices into a distributed system that cooperates and coordinates its activities with its users. We envision that ubiquitous computing will soon extend beyond the boundaries of "prototype" experiments and encompass larger areas, enabling smart building, smart campuses, and smart fleets [1]. However, security, privacy, and fault-tolerance are the major hurdles for real-life deployment of the technology on a wide-scale.

Users of today's computing and information systems expect these systems to be available even when under attack, to perform their tasks in a timely manner, and to provide accurate results consistently. The problem of securing critical cyber infrastructure (CCI) is exaggerated by smart buildings that must bind networks, processors and devices with policies, mechanisms, protocols and services to offer survivable and secure operations, while providing better management and finer integration between the heterogeneous components. Moreover, a secure critical infrastructure for smart buildings is essential, because many of the services that a building provide is critical to supporting its inhabitants, including surveillance systems, HVAC, lights, door locks, etc.

Many of the issues we highlight and address in this paper are motivated by real-world security, privacy and survivability challenges that we face in the new Siebel

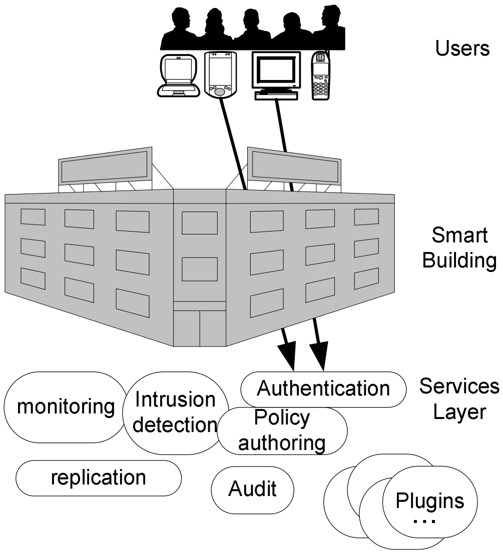

Figure 1: Hestia Overview

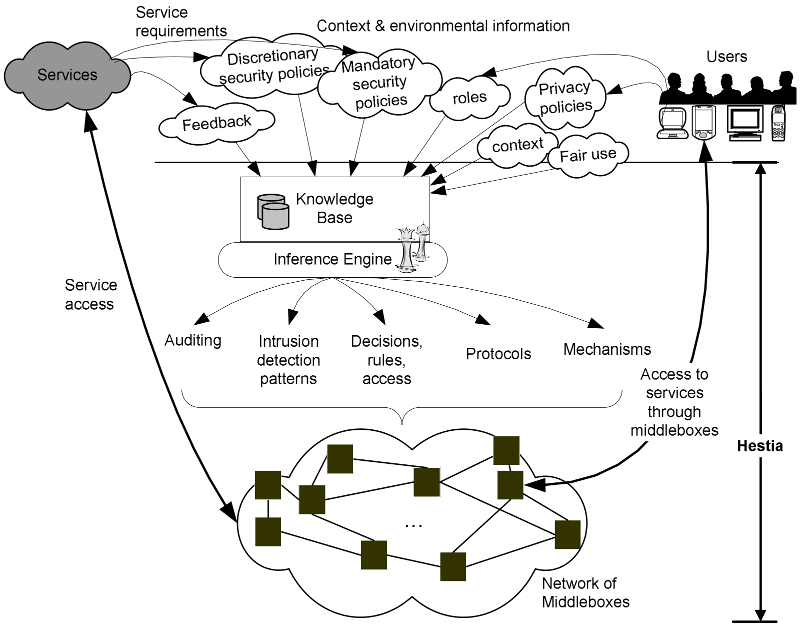

Figure 2: Hestia Middleware

Center for Computer Science. The Siebel Center is a "smart building," and showcases state-of-the-art computing and communications infrastructure in its offices, meeting rooms and classrooms. Digital locks, heating, cooling, and lighting controllers, video cameras and other digital sensors and actuators are deployed throughout the building, and will be accessible via a private network. Faculty, students and visitors to our building need reliable access to computing, communication, and information services that they are authorized to use. The building also needs to be secure against accidental software or hardware failures or malicious attacks. Since the Siebel Center is an open academic environment, draconian access controls are not feasible; however, the security and survivability of the critical infrastructure within the building is a key consideration. In addition, the heterogeneous components and subsystems of the building need to be integrated to enable a greater level of interoperability.

We propose *Hestia* a middleware for providing a secure layer for critical cyber infrastructures, like smart buildings and other large-scale ubiquitous computing environments. *Hestia* partitions the network into two trust domains: an application-domain network for clients, and a protected network consisting of Middleboxes and heterogeneous services. This partitioning is enforced at the perimeter of the *Hestia* layer using network-level firewalls or NAT boxes. A Middlebox is a node that contains some instantiation of security, privacy and/or load balancing mechanisms. The Middleboxes network, as shown in Figure 2, acts as a cluster of reconfigurable computing and communication nodes. The Middleboxes provide a programmable distributed-object interface that enables service owners and administrators of our system to exercise fine-grained control over network service usage, deployment, management, and control. They also provide appropriate filtering mechanisms to enforce customizable anonymity, confidentiality and privacy concerns. *The* service-domain includes a set of networked services and proxies encapsulated by routers and firewalls and an application-domain network. In the application-domain, building services are exposed by the *Hestia* discovery protocol as application service interfaces of Middlebox proxies that are mapped across routers and firewalls. This novel distributed-object framework uses a network of "Middleboxes" to integrate security, privacy and reliability concerns into the service domain. The *Hestia* service-domain supports service composition, survivability, distributed trust management and control. The system offers distributed deployment of mechanisms, such as access controls, anonymity, replication, load balancing, auditing, and intrusion detection.

In the following sections, we define the *Hestia* middleware architecture, and summarize the contributions of a *Hestia*-enabled security infrastructure. We explain our architecture in more detail with the help of an example. We examine the task of controlling access using swipe-cards to different parts of our smart building that relies on a networked Lock-server. All doors in our building are augmented with e-locks that open to authenticated users who are authorized by the policies. The swipe-card detector sends a message over the network that causes a lookup in a door-lock access database associated with the Lock-server. If the user is authorized, the door opens in response to a reply-message from this server. We wish to extend this mechanism to provide additional services. For example, to assist people with disabilities, we offer them UbiSense location technology [2]. UbiSense employs special tags that transmit ultra-wide radio bands. The tags can be associated with a particular user. Tag-detectors can detect the presence and location of a user within six inches within the building. Some services can be programmed to use the UbiSense system to identify and authorize disabled users waiting to enter a door and open the door through the door-lock mechanisms.

The door-lock server is an important CCI service in this context. It is also a single-point of failure, and can be the target of various attacks. Since all messages are sent in plaintext, the *Hestia* layer can provide the required encryption support to protect the confidentiality of the information exchanged and preserve the privacy of individuals. In addition, access to the door-lock service itself has to be fault-tolerant and the system should be able to balance the load dynamically when requests become a bottleneck to server access. No client in the system should know where the server is located to prevent DoS attacks.

2. Architecture

Our architecture is divided into a *service domain* and an *application domain.* The service domain provides users and services with a set of mechanisms for fault tolerance, quality of service, and privacy through a network of Middleboxes. Because a smart building environment is dynamic and context-aware, it must capture and act upon many factors, like context information, role hierarchies, security policies, building plans, risk factors, and service dependencies. Managing all these factors individually is difficult. Therefore, the service domain contains a Knowledge Base and an Inference Engine. The Knowledge-base is a repository for the context information, role hierarchies, security policies, etc. The Inference Engine uses the information in the Knowledge Base and composes requirements and mechanisms for satisfying these requirements, generates service graphs and applies the necessary policy decisions (access control, privacy, etc) to the service graphs. Additionally, the Inference Engine must generate auditing mechanisms for the service and intrusion detection monitoring for detecting malicious activity. Figure 2 illustrates how user and service policies are used in Hestia to derive the appropriate security mechanisms.

Hestia makes extensive use of Role Based Access Control (RBAC). Access to resources will typically be granted based on the user's role along with contextual information. The Inference Engine also contains a secure feedback component. When a user is denied access to a resource, the user must be provided with useful feedback on how access can be gained (e.g., should the user come back at a later time, or should the user obtain additional credentials?). However, unconstrained feedback may reveal too much information about the system's policies. In [3] we describe *Know*, a mechanism for providing useful feedback while honoring the privacy of the system's policies. Such a component is very important in a ubiquitous environment where several users interact with a plethora of devices.

The application domain provides users (based on their role, for example) with an abstraction of what services are available and a set of interfaces to interact with the service domain. The application domain provides users with discovery and lookup services. Services are accessed through the service domain. Application domain services (discovery, lookup, etc.) are also subject to security constraints, so that users will be able to view and access services based on their roles.

For example, the door lock service is accessed through the service domain via a proxy that runs on a Middlebox. This service will appear in discovery and lookup services in the application domain. After discovering the location of the doorlock service proxy, users can interact with the proxy in the service domain. Other services in the Siebel Center will include a location service that tracks users' movements while respecting privacy, heating and cooling, sprinkler systems, etc. All these services need suitable protection mechanisms.

The network of Middleboxes provides mechanisms that include load balancing, fault tolerance, routing, anonymity, security and quality of service. Each service is provided by a specific layer within our Middleware. We briefly describe these layers.

2.1 Load Balance Layer (LBL)

This layer distributes service processes among participating Middleboxes in a probabilistic manner. The main goal of the load balancing service is to ensure that no Middlebox becomes overloaded. The load balancer is given a resource requirement profile for each service that is to be hosted. This information should include CPU, disk space, and bandwidth requirements. At this layer, process migration occurs when the load of the Middleboxes needs to be redistributed. Migration of services will need to be done securely. For example a service like Kerberos that stores private keys may require a secure transfer to another Middlebox. Furthermore, access to migrated proxies can be maintained for connected users though a mobile-IP approach. Messages to the previous “home” of the proxy are forwarded to the new home.

Addressing: When a service is to be hosted by a Middlebox, the load balancer is given a pseudonym for the service. It then creates an object reference for the service. After creating the object reference, the load balancer registers the object reference and the pseudonym with the Name Space. To resolve a service, an end-user presents the pseudonym to the Name Server and is returned the corresponding object reference or handle.

Denial of Service protection: Through distribution of load, the Middleboxes are able to handle larger aggregate loads, thus increasing DoS resilience of the network. Additionally, services can specify usage constraints as part of their requirements. This layer maintains the aggregate load for proxies of a service and keeps it within the maximum allowable load, thereby preventing DoS attacks on the actual service.

For example, since the doorlock service is used heavily within a building, its proxies may attract a considerable amount of load. The LBL layer, in such cases, could migrate proxies to ensure that no Middlebox is overloaded. Furthermore, if users try to overload the service by making repeated requests for access, the load balancing layer will ensure that the aggregate request bandwidth to the doorlock database is kept within acceptable thresholds.

2.2 Fault Tolerance Layer (FTL)

While Hestia may provide basic fault tolerance for services (through replication for example), this layer maintains availability of services by replicating the proxies for a service. If proxies are attacked (e.g., DoS), then new proxies for the service can be instantiated on the Middleboxes. Hence this layer provides fault tolerance through availability of services through proxies. Services are deployed along with replication requirements for their proxies. Proxies can be replicated for quick recovery using active or passive replication mechanisms. With active replication, the replica is maintained in the same state as the original proxy. Passive replication uses periodic checkpointing of consistent states. For example, the doorlock service can deploy several proxies through the LBL. This ensures that the doorlock service can be accessed through several proxies, increasing its availability and fault tolerance. If a proxy fails, its active (or passive) replica can be used as a substitute. Fault tolerance service may require a more sophisticated fault-tolerant Middlebox access schemes.

2.3 Anonymity Layer (AL)

This layer obfuscates the identity and/or location of the client, the Middleboxes, or both. Onion Routing [4], Crowds [5] and Mist [6] routing concepts at this layer provide anonymity.

We provide a solution based on Mist, where users or services establish routes through Middleboxes while keeping their locations private. Handles that maintain forward and reverse paths for packets establish a route though Middleboxes. Each router only knows the previous and next hops. These routes eventually end at some Middlebox, which serves as a point of communication for the entity. In Mist, this Middlebox is called the Lighthouse for that entity. Traffic for that entity is directed to its Lighthouse, which then forwards the data down the established path. Since communication takes place through Lighthouses, the actual location of an entity is not revealed. Hence this decouples a user’s identity from its location, giving the user location privacy or location anonymity. Users can optionally choose to not reveal their identities, thereby achieving location and identity anonymity. This layer provides anonymity services for users and services.

- Anonymity for Users: Since users must constantly interact with services in ubiquitous computing environments, it becomes feasible for the system to track a user’s movements. For example, the administrator of a service can mine service logs and infer the location of users. Providing anonymity to users ensures that their locations are kept secret while allowing them to interact with services through their Lighthouses.
- Anonymity for Services: While critical services may be deployed through Middleboxes, we would like to keep the location of these services private. This would prevent malicious parties and even insiders from identifying the specific machines on which they can mount attacks. Such attacks could cripple the targeted service. Hence the anonymizing layer of the Middleboxes provides this

functionality. For example, the ubiquitous environment might store a plethora of information pertaining to users. This includes personal data, as well as usage statistics of users. Storage of such data is a source of threat to a user's privacy. Data storage can also be anonymized by storing the data in undisclosed locations. While data are accessible though the service's Lighthouse, its location is not known. This prevents malicious parties from denying access to data, since they do not know which machines they can mount attacks on. Furthermore, the data can be replicated to increase its availability, and resilience to denial of service attacks. This can be accomplished by the Fault Tolerance layer.

Continuing with our example, the location of the doorlock service may be kept secret through this layer. This is important since the doorlock service is crucial to the operation of a building, and exposing its location would make it vulnerable to attackers.

2.4 Quality Layer (QL)

This layer provides users with an interface for specifying the level of quality that they would like to receive. Quality may be specified with regard to data transmission metrics or quality of service (QoS) (e.g. delay, bandwidth or packet loss rates). Quality may also be specified with regard to the level of security or quality of Protection (QoP) that a client requires, in terms of confidentiality, integrity, anonymity, etc. For example, services can request higher grade encryption and authentication for users connecting with the service. Users may also request routes from the QoS layer with QoP requirements such as bandwidth distribution that is immune to traffic analysis. The doorlock service, for example, may demand a certain level of authentication (QoP) for access to the locks. It may also request QoS parameters such as low latency for quick response times.

2.5 Secure Services Layer (SL)

This layer uses Middleboxes to provide access control, confidentiality, and integrity for services. Services can upload access policies into the SL, which can then be combined with the system policies to control access to the services. The doorlock service may request the use of encryption and signatures to ensure the privacy and integrity of messages to the service.

Functionally, the Middlebox-network in Figure 2 acts as a cluster of reconfigurable computing and communication nodes. Clients can only access services in our system via the Hestia layer. Servers create and install proxies on the Middlebox network either proactively or reactively in response to requests from clients. These proxies provide a restricted view of service interfaces, corresponding to the authorizations of the requesting clients, and may actually implement the server's logic and perform computation and serve data, thereby offloading the server's computation dynamically. The proxies also act as filters to provably enforce confidentiality and privacy concerns. In addition, these proxies can be replicated for load balancing and fault-tolerance, or form a customized network among themselves to provide QoS or VPN-style routing. The instantiation of proxy objects on Middleboxes is driven by the policy specifications, and different functional and non-functional requirements can be composed to create customized service objects on-demand. By defining the interfaces and rules for composition correctly, different protocols and functions can be composed on top of each other using standard object composition techniques to provide differentiated services.

The proxy objects in our architecture are not persistent and maintain little global state information. Ideally we should be able to migrate and restart proxies on any Middlebox in our network with little overhead. To accomplish this, the proxies are carefully designed to implement soft-state protocols. Even when a proxy is attacked and compromised, the damage is contained, and a new proxy can be restarted on a different Middlebox with little effort. Proxies have little knowledge of other services and systems. This aspect of our design allows us to build truly survivable network services that can continue to provide service guarantees and degrade gracefully under attack, on top of the existing best-effort network service model.

3. Implementation

We are implementing the network of middleboxes as an overlay network over TCP/IP. We choose Prolog to act as a simple inference engine. We use CORBA [7] as the major backbone for communication in our distributed system. CORBA offers several services that are instrumental in implementing some of the needed functionality, including the support for atomic transaction, persistent objects, and platform independence. Many CORBA implementations are heavyweight and may not be appropriate for implementing an overlay network. Therefore, we are experimenting with the Universally Interoperable Core (UIC), which provides a lightweight, high-performance implementation of CORBA [8]. Every layer in our architecture has a broker that provides a CORBA IDL interface that allows services and users to access the functionality provided by that layer. To demonstrate the system, we are in the process of developing several services for controlling different functionality in the smart building including locking/unlocking of doors, controlling lights, configuring HVAC systems, as well as other applications that utilize the smart building environment as a whole, like services that unlock doors and turn on lights automatically for disabled people as they move in the environment.

4. Related Work

Several middleware platforms for distributed programming introduce meta-programming extensions that provide functionality that is similar to Middleboxes. CORBA provides portable interceptors [9], which allow developers to extend and control the behavior of the ORB. However, interceptors have limited capabilities, and can only reside on the client or server sides, whereas Middleboxes are envisioned to run on intermediate machines. Java RMI [10] introduced RMI stubs for distributed objects. The stub is a remote reference to a distributed object in which all method calls are merely forwarded to the target object. JINI [11] and some CORBA implementations, like TAO [12], support “smart proxies.” A smart proxy is similar to a stub, but can provide additional features like results caching, failover, and custom protocols to communicate back to the target object. Microsoft .NET Remoting [13] supports a construct similar to smart proxy (RealProxy and TransparentProxy). Middleboxes provide functionality that is similar to smart proxies. However, unlike typical smart proxy frameworks, where a single proxy is created per client, Hestia supports many-to-many relationships between clients and Middleboxes on one hand and between services and MiddleBoxes on another. For example, a Middlebox can choose to forward a client’s request to one of many active services in order to achieve load balancing. For fault tolerance, Hestia supports the dynamic creation of Middleboxes on the fly, as well as dynamic bindings between Middleboxes and services as they become available. Furthermore, several mediators may be needed to provide sufficient services. For example, anonymity may require communication channels to traverse several Middleboxes in sequence to provide a better level of concealment.

5. Conclusion

We believe that *Hestia* will open new frontiers in the design, development and deployment of trusted CCI for buildings. The unique Middlebox architecture provides a programmable distributed object-oriented framework that is inherently survivable. It enables us to integrate security, privacy, and reliability mechanisms into service access interfaces and implementations. *Hestia* will allow services to not just execute, but execute with desired security, availability, quality of service and load-balancing properties.

Our proposed research will have a significant impact on the adoption of CCI by buildings as diverse as hospitals, airports, offices, laboratories and power plants. It will demonstrate that smart building services can be deployed without compromising privacy or critical safety, security and survivability properties. It will encourage new industries for applications that exploit smart buildings: ubiquitous services for users with disabilities; safety and rescue operations; sophisticated anti-theft approaches; building guide and navigational services; personal safety; communication, collaboration, education services; people control as in airport personnel management/restricted area access control subsystems, and passenger information subsystems.

6. References

[1] J. Al-Muhtadi, S. Chetan, and A. Ranganathan, "Super Spaces: A Middleware for Large-Scale Pervasive Computing Environments," presented at Middleware Support for Pervasive Computing Workshop, in conjunction with PerCom, Orlando, FL, 2004.

[2] UbiSense, "Local position system and sentient computing." http://www.ubisense.net/.

[3] A. Kapadia, G. Sampemane, and R. H. Campbell, "Know Why Your Access Was Denied: Regulating Feedback for Usable Security," Technical Report: UIUCDCS-R-2004-2406/UILU-ENG-2004-1708, 2004.

[4] M. Reed, P. Syverson, and D. Goldschlag, "Anonymous Connections and Onion Routing," *IEEE Journal on Selected Areas in Communication, Special Issue on Copyright and Privacy Protection*, 1998.

[5] M. Reiter and A. D. Rubin, "Crowds: Anonymity for Web Transactions," *ACM Transactions on Information and System Security (TISSEC)*, vol. 1, 1998.

[6] J. Al-Muhtadi, R. Campbell, A. Kapadia, D. Mickunas, and S. Yi, "Routing Through the Mist: Privacy Preserving Communication in Ubiquitous Computing Environments," presented at International Conference of Distributed Computing Systems (ICDCS 2002), Vienna, Austria, 2002.

[7] OMG, "CORBA, Architecture and Specification," Common Object Request Broker Architecture (CORBA) 1998.

[8] M. Roman, F. Kon, and R. H. Campbell, "Reflective Middleware: From Your Desktop to Your Hand," *IEEE Distributed Systems Online. Special Issue on Reflective Middleware*, 2001.

[9] OMG, "CORBA 3.0.3 Specification, formal/2004-03-01." available at http://www.omg.org/cgi-bin/doc?formal/04-03-01, 2004.

[10] Sun Microsystems Inc., "Java Remote Method Invocation (Java RMI)." available at http://java.sun.com/products/jdk/rmi/.

[11] J.Waldo, "The Jini architecture for network-centric computing," *Communications of the ACM*, vol. 42, pp. 76–82, 1999.

[12] D. C. Schmidt, D. L. Levine, and S. Mungee, "The Design and Performance of Real-Time Object Request Brokers," *Computer Communications*, vol. 21, pp. 294-324, 1998.

[13] Microsoft Corporation, "Microsoft .NET Remoting: A Technical Overview." available at http://msdn.microsoft.com/library/.