Improving Multi-Tier Security Using Redundant Authentication*

Jodie P. Boyer
jpboyer@uiuc.edu

Ragib Hasan
rhasan@uiuc.edu

Lars E. Olson
leolson1@uiuc.edu

Nikita Borisov†
nikita@uiuc.edu

Carl A. Gunter
seclab.uiuc.edu/cgunter

David Raila
raila@uiuc.edu

Department of Computer Science
†Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801

ABSTRACT

Multi-tier web server systems are used in many important contexts and their security is a major cause of concern. Such systems can exploit strategies like least privilege to make lower tiers more secure in the presence of compromised higher tiers. In this paper, we investigate an extension of this technique in which higher tiers are required to provide evidence of the authentication of principals when they make requests of lower tiers. This concept, which we call *redundant authentication*, enables lower tiers to provide security guarantees that improve significantly over current least privilege strategies. We validate this technique by applying it to a practical Building Automation System (BAS) application, where we explore the use of redundant authentication in conjunction with an authentication proxy to enable interoperation with existing enterprise authentication services.

Categories and Subject Descriptors: D.4.6 [Security and Protection], J.7 [Computers in other systems], K.6.5 [Security and Protection]:*Authentication, Physical Security*

General Terms: Security

Keywords: Authentication, Building Automation Systems

1. INTRODUCTION

Multi-tier web server systems are of great importance in e-commerce and enterprise information applications, and are gaining importance in many other contexts. Motives such as extortion, identity theft and other types of fraud have made these systems highly attractive as targets for attackers, so their security is a cause for rising concern. Such systems split functionality between "tiers" such as a web server, application server, and database. To improve security of the system as a whole it is desirable to limit the privileges of higher tiers when they utilize the services of lower ones. A typical strategy is to use the principle of least privilege to limit a higher tier's access to only the lower-tier functions required for its mission. However, the operation of a higher tier typically requires broad privileges in the lower tier. For example, an application server that needs access to a database may be given access to all records on the database even if, at any given time, it should only be allowed to access the records of a limited collection of principals who have authenticated themselves to the application server and made requests that require access to the limited view of the database.

In this paper we explore a strategy we call *redundant authentication* that can be used to limit the privileges of higher tiers based on a requirement that they produce non-repudiable evidence that principals on whose behalf they operate have recently authenticated themselves to the system. This greatly limits the scope of a compromise in many cases, including the case study we use to validate the concept. Our case study consists of two applications that integrate enterprise authentication with the security functions of a Building Automation System (BAS) using redundant authentication. The first application allows principals to delegate access to rooms and the second allows them to open doors remotely. Both applications are implemented using a higher-tier web application server that controls the lower-tier BAS server. In particular, we construct a middle-tier control system gateway that enforces attenuation of privilege to limit actions that the application server performs for specific principals. With this protection, only the resources of principals that have recently authenticated a request to the application server can be affected. Redundant authentication assures that certain basic control system privileges are respected by the application server, again assuring a limit on the behavior of a compromised application server.

The paper offers three contributions. The first is the idea of redundant authentication as a strategy for robust multi-tier security. The second is a pair of BAS applications to validate this approach in a practical context. The third is a strategy for implementing redundant authentication in practical contexts using an *authentication proxy*, which is an isolated process that translates enterprise authentications into

*This paper is to appear in CSAW 2007 in Fairfax, Virginia

CSAW'07, November 2, 2007, Fairfax, Virginia, USA.

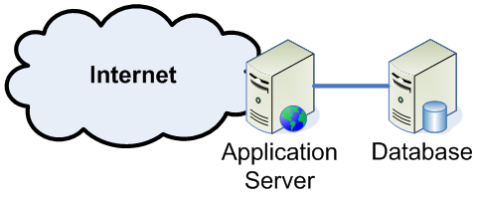

Figure 1: Multi-Tier Web Server System

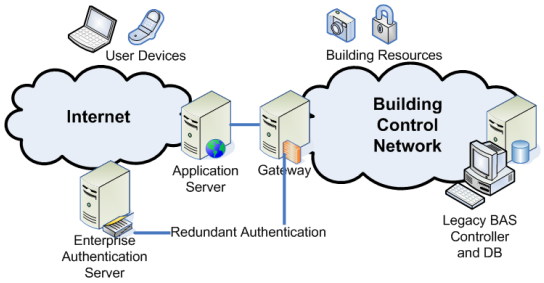

Figure 2: User Accessible BAS

non-repudiable credentials that can be used for redundant authentication. We applied this strategy to BAS applications to implement them utilizing the University of Illinois enterprise authentication servers.

The paper is organized into six sections. In the following section, we discuss the general concept of redundant authentication, including possible applications and some of the challenges to its use. In Section 3 we discuss application requirements for a BAS case study and how they may relate to multi-tier security. In Section 4 we consider the design and implementation of our prototype using redundant authentication with an authentication proxy. Section 5 provides analysis and related work. Section 6 concludes.

2. ARCHITECTURE

Web server systems are often organized as a collection of tiers that divide responsibilities between specialized software components. In a typical case, a system is organized as an application server that is attached to the Internet and communicates with back-end components such as a database. Figure 1 illustrates this organization. When Internet users require service, they authenticate to the application server, which determines their authorizations and uses these to modify the database according to the privileges of the principal that represents the user on the system. Although the application server is acting on behalf of a single principal in this instant, it requires general access to the database because it may act on behalf of many different principals. This is worrisome in the face of threats like identity theft because an attacker who manages to compromise the application server may be able to execute a wide range of actions relative to the database, such as querying for the names and Social Security numbers of all principals in the system.

Ideally one would like to limit the application server to act on behalf of a specific client, perhaps by deriving a view for that client and restricting it to that view, but to be effective, there must be a way to enforce such a restriction on the application server. One way to do this is to insist that the application server declare that its actions are on behalf of a specific principal, for whom it produces evidence that the request requires access to the database. This is possible in some circumstances and would have the effect that the application server is unable to gain access to large amounts of sensitive data. It may still compromise resources of principals that are validly authenticated to the system, but in many applications this will be only a small portion of the system resources as a whole. To accomplish this, we examine the idea of redundantly authenticating a principal by demanding proof of authentication at the lower tier and enforcing restricted privileges according to the specified principal.

Redundant authentication is a possible protection measure for many types of multi-tier systems. To show how it can be used in practice and to explore some of its challenges, consider a typical BAS integration scenario depicted in Figure 2. Here a user on the Internet (or enterprise network) obtains a credential from the enterprise server to authenticate himself and obtain authorization for actions like opening a door using the BAS control network. It is typical that such control networks provide modest security and, given the importance of the resources at issue, they are protected behind an application server and gateway. The architecture is comprises of three tiers, with the BAS controller itself representing the bottom tier. The middle tier, the gateway, provides support for interoperability with the control system and adds security by separating the complex software on the application server from the vulnerable control network. Redundant authentication is part of this protection. If, for instance, the gateway insists that requests from the application server come from a recently authenticated user, then a compromised application server is unable to execute a command such as "open all doors". Indeed, the most it could do is to open doors that could have been opened by recently authenticated principals. This significantly mitigates risk in the system. Referencing Figure 2, the protocol operates as follows. A user authenticates himself to the application server, which determines authorization for requests. The server processes the authorized request by passing the request to the gateway, along with the redundant authentication token. The gateway enforces policy restrictions using redundant authentication and information within the legacy BAS controller and decides if the action is authorized. The gateway then effects the necessary actions using the protocol of the BAS, which carries out actions with the appropriate building resources. In order to make an accurate decision, the gateway must check that the authentication came from a trusted source, which means that any authentication claims must provide non-repudiation. Although there is some cost in treating the authentication redundantly, the advantage is that complex application-specific logic is not carried out on a machine attached to the building control network and its actions are limited by the general policy on the gateway.

To support redundant authentication as discussed above, the authentication material passed from the user to the application must be non-repudiable; i.e. the application server should be able to forward a token to the gateway that convinces it that a valid user has authenticated. Most enterprise systems do not provide non-repudiable credentials, so it is necessary to insert a proxy authentication system to provide this function. This enables the gateway to communicate directly with the application server and facilitates

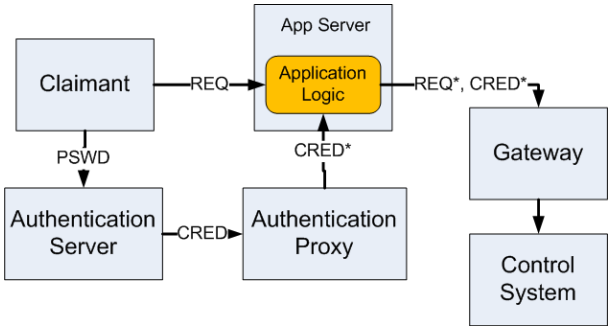

Figure 3: Proxy Authentication

interoperability with client claimants. Figure 3 illustrates the concept. The claimant provides a request REQ to the appropriate application logic on the application server and uses a password PSWD to authenticate to the authentication server, which then provides a credential CRED for the authentication proxy. The authentication proxy creates a non-repudiable credential CRED*, which it provides to the application server. These are supplied to the gateway in a high-level protocol where the client request is expressed in a command REQ*. The gateway enforces limitation of privilege and translates this high-level command into a command that the control system understands. In this way the existing enterprise authentication server supports the redundant authentication in a way that is transparent to the client.

An alternative to redundant authentication would be for the client to authenticate to the gateway directly. For example, the application server could bundle the request REQ* and pass it to the client to give to the gateway, along with the (repudiable) credential CRED. This method also prevents a compromised application server from acting arbitrarily maliciously. However, it presents a much larger attack surface for the gateway, since now any user may send any request to it and thus exploit potential vulnerabilities. With redundant authentication, only the application server ever needs to communicate with the gateway, and thus both the application server and the gateway must be compromised in order to gain unrestricted access to the control system. In addition, with redundant authentication, the application server may implement a different authorization policy than the gateway. The gateway uses a simple policy, such as attenuation of privilege. The application policy may be much richer and implement organizational constraints, such as forbidding delegation of access to and remote unlocking of sensitive rooms. As long as the application server is not compromised, both policies operate in concert, and in case of compromise, the gateway policy mitigates the damage that can occur.

3. APPLICATION REQUIREMENTS

As discussed in the previous section, we focus our efforts on building a redundant authentication system for a building automation system. In this section we present our applications, threats, threat model, and discuss how we plan to mitigate the threats to our system.

While building automation systems are responsible for controlling many different functions, our applications focus specifically on building security systems. We propose two applications for a building automation system. First, we propose an application that provides the ability to delegate room access to other principals in the building (called delegated access), and second, we propose an application that gives principals in the building the ability to unlock doors for which they have been granted access via a web service (called mobile access).

The delegated access system allows building managers to assign rights to room managers that in turn manage the access list for a room. As an example, imagine that Joe runs the lab in room B and needs to give his student Sally access to the room. In a typical system, Joe contacts the building manager, Bill, who gives Sally access to the room, followed by a notification to Sally to tell her that her access rights have been updated. If the system had delegated access rights, Bill would give Joe the rights to manage the access list himself. This way, Joe can simply log into the delegated access system and add Sally to the access list. This addition would be audited and, once the change has been made, Sally, Joe, and Bill would receive e-mail notification of the change and Sally would have access to the room.

The mobile access application allows users to connect to the system either over the web or with an application on their cell phone and request that their door be unlocked. Currently, if a user, Fran, forgets her key she must visit the building manager who will issue her a physical temporary key. With the mobile access scenario, Fran could simply use a web service to request that her door be opened, either by web access or by a small application on her cell phone that would send a command to open her door. Actions through the mobile access system should be audited for suspicious behavior to ensure that they do not enable new types of attacks on the building or other assets.

Because these applications deal directly with the building automation system, it is important to consider the threats. The major concern associated with a security breakdown in one of these systems is that it may leave the building automation system vulnerable. Attackers could launch serious attacks such as modifying access to rooms in the building or unlocking arbitrary doors in the building. Both of these threats would allow them to mount a physical attack on a building, during which they could steal equipment, access sensitive files, *etc.* Another example is a denial of service attack, where attackers could remove some or all access rights to rooms in the building, rendering the building unusable by its occupants. However, attacks on the building are not limited to attacks on the building security system. If attackers gain access to the building, they may also be able to change the status of other systems controlled by the BAS, such as environmental controls or security monitoring systems.

A least privilege system would be able to mitigate threats to other building functions, since neither of our applications needs to access them (although we are currently exploring other applications that allow users to control room temperatures and make use of the video monitoring system). However, least privilege cannot address threats to the building security functions, and thus we use redundant authentication as a core strategy to mitigate them. In particular, it is important to specify the type of policy that the gateway applies to requests. In our system, the gateway applies an attenuation of privilege policy that rejects requests unless the entity making the request has been issued access within the BAS to the room on which they are performing an ac-

tion. This means that principals can only delegate access to rooms to which they have access, and the mobile access system will not give principals more access then they already have. An important feature of this policy is that it uses information that is accessible to the gateway independently of the application. Additionally, it provides the application server the ability to enforce special purpose access policies, such as maintaining a set of principals and delegation rights.

We assume that attackers may be able to compromise the application server, but not the authentication proxy or the gateway. This is a reasonable assumption because the authentication proxy and gateway are small, special purpose, isolated implementations and can therefore conform to stringent security requirements. As a result, we assume that the gateway does not fully trust the application server. Attackers might gain the ability to replay cached authentication certificates with requests to the gateway within a certain time period. However, our policy enforces the constraint that attackers are only as powerful as recently authenticated individuals.

As a case study, we have developed a system for delegated access and mobile access for the Siebel Center for Computer Science at the University of Illinois. The Siebel Center is a 225,000 square foot office building that houses the UIUC Computer Science Department that was built in 2004 and uses the Andover Continuum System for its BAS. Additionally, we integrate our system with two available authentication systems: Bluestem [9], a Kerberos-based authentication system developed at the University of Illinois, and Microsoft's Active Directory system [11]. We present the details of this case study in the following section.

4. IMPLEMENTATION

Figure 4 shows the architecture of the prototype we implemented to satisfy the requirements of Section 3. An arbitrary client uses the system by directly contacting the application server, which is described in Section 4.1, and issues a command such as "unlock the door to room 101." If the application server determines that the client has not yet authenticated, or if it decides that enough time has elapsed that the client needs to re-authenticate, it redirects the client to the authentication proxy, described in Section 4.2. The authentication proxy may use a pre-existing enterprise Authentication Server, such as UIUC Bluestem or Active Directory. If the client successfully logs in, then the proxy issues a cryptographic token to the client. This token is passed back to the application server, which maintains its own policy logic. If the command issued from the client is allowed by the policy, based on the client's identity as confirmed by the authentication proxy, then the application server sends the command and the cryptographic token to the gateway over a dedicated link.

The gateway, described in Section 4.3, is a trusted server that enforces the building system's access control policy logic, which is independent from the particular application that issued the command. The policy data is stored in a database within the building control network, and can be queried using standard SQL. If the building system's policy logic is satisfied, then the command is translated into the OPC standard and sent to the controller. The building network, including the SQL server and the controller, is a legacy system provided by Andover Controls.

We have implemented two applications to demonstrate this architecture. These implementations are described in Section 4.4.

4.1 Application Server

The application server utilizes an XML-based interface to the system using the oBIX XML language for operating mechanical and electrical control systems in buildings [4]. Using an XML specification enables other oBIX-based applications to have a common communication vocabulary. In recent versions, the purpose of oBIX has been generalized to include any type of embedded software systems. The committee specification defines a general object model and a set of operations on these objects. In theory, any kind of object could be represented in this model; however, we choose to use other more appropriate XML specifications for objects such as user principals and authorization tokens.

In oBIX, objects are referenced via a named URI and accessed with the `read` or `write` operations. Commands that are more complex than a read or write, such as modifying an access control list, can also be performed with the `invoke` operation. In our prototype, each door is assigned a URI and contains two boolean sub-objects[1] called `unlocked` and `open`, representing respectively whether the door is unlocked, and whether the door is open. Principals are granted the privilege to write to the `unlocked` sub-object, but the `open` sub-object is read-only. This is simply a design decision reflecting the capabilities of the doors in our prototype. One could easily imagine a system in which doors can also be opened automatically.

Additionally, the prototype provides access control to each area of the building. Each area has doors assigned to it. This way, if an area of the building has two doors, a principal only needs to be granted access once to the area, rather than once for each door into the area. These area objects are also assigned URIs and can be accessed with a `read` oBIX operation, which returns the doors assigned to an area. These values are read-only, since doors and areas are fixed locations in a building; thus, the `write` oBIX operation is meaningless for area objects.

However, a more interesting operation is `invoke`, for which we define two possible commands: `grant_access` and `revoke_access`. We wish to allow delegation of granting access to rooms to the "owners" of a room. Ownership of rooms and delegation is not currently supported by the BAS, and therefore must be implemented and enforced by the application itself.

The access control lists for each area are stored in a database in our prototype system. In order to respect this security policy, we also require principals to identify themselves. The authentication process is described in Section 4.2. The application server receives a security token from the authentication proxy. This token is encoded as a PKCS#7 signed statement [16], using syntax as described in [10]. This signed statement can be verified against the authentication proxy's public key, and is included in any requests sent by the application server to the gateway.

4.2 Authentication Proxy

Our architecture employs a proxy authentication server in order to abstract the details of the enterprise authenti-

[1]The term sub-object is used in the model description of the committee specification [4], we simply reuse their vocabulary.

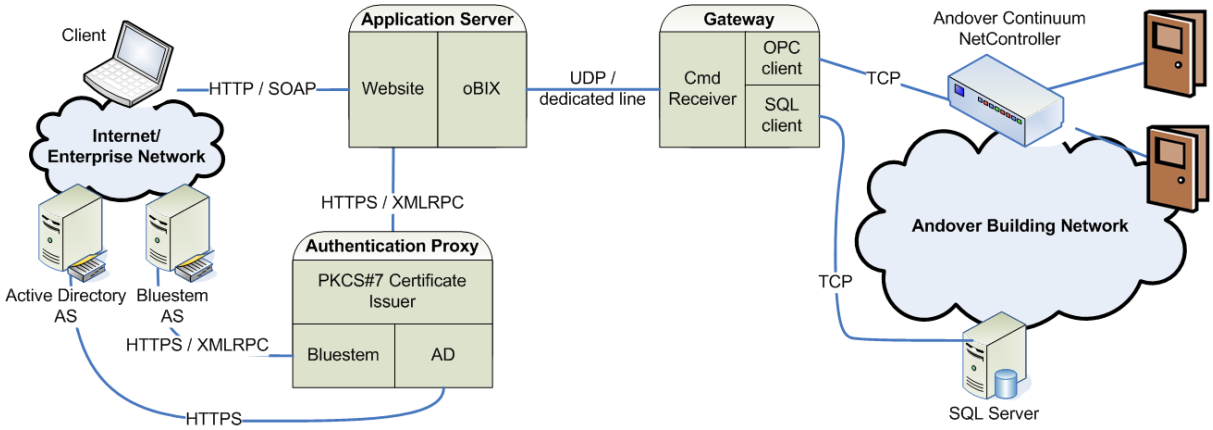

Figure 4: Prototype Architecture

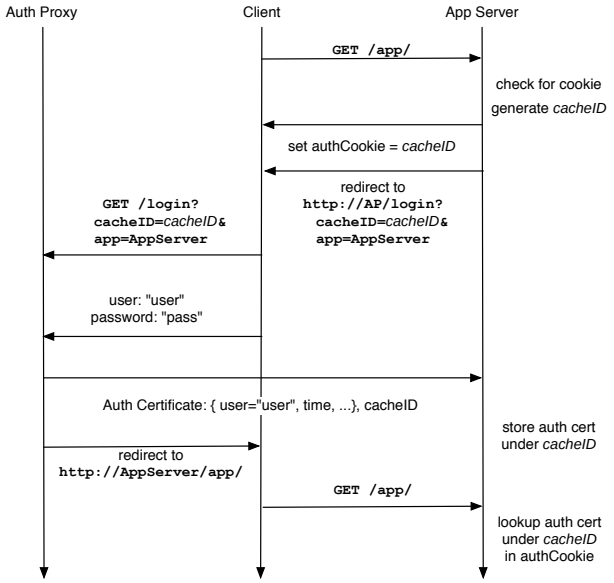

Figure 5: Authentication Protocol

cation system from other components of the system and to provide a non-repudiable authentication credential that can be used in redundant authentication. The authentication server itself is a trusted component of the system and we expect that it should receive similar protection measures as (and perhaps be colocated with) the enterprise authentication server.

Our design of the protocol for interactions with the authentication proxy is based on the UIUC Bluestem protocol [9], an authentication service for web applications deployed on the University of Illinois campus. The protocol is a three-way interaction between the client, the application server, and the authentication proxy, shown in Figure 5.

1. When the client first contacts the application server, the application server checks for the presence of an authentication cookie. If this cookie is absent, it generates a cache ID and sets it in a cookie at the client.

2. The application server redirects the client to a web page on the authentication proxy, passing as parameters the cache ID and the name of the application being requested. The application server asks the client to log in using the enterprise authentication credentials.

3. Assuming the enterprise authentication is successful, the authentication proxy generates an authentication credential, digitally signed using the private key of the AP, and then sends it directly to the application server, coupled with the cache ID. The application server verifies the signature and stores the credential in its authentication cache.

4. The authentication proxy redirects the client back to the application server web page. The application server notices the presence of a cookie and uses the cache ID to look up the presence of an authentication credential in the cache. If it is present, then the authentication is considered successful; otherwise, it returns to step 1.

The cache ID serves as a secret index into the application server authentication cache, and an attacker who guesses it would be able to exploit the legitimate client's credentials. Therefore, the ID is generated as a random 128-bit number and is kept secret during communications between the client, application server, and the authentication proxy by way of using TLS to protect the individual connections.

Both the application server and the gateway maintain a trustworthy copy of the public key of the authentication proxy, so they can both verify the signature on the authentication credential. The credential includes the following fields: a) the current time, b) the enterprise authentication method used (we currently support two, Microsoft Active Directory [11] and UIUC Bluestem), c) the enterprise ID of the client, and d) the application service that had requested the authentication. The last component is present to ensure that another application or service that uses the authentication proxy cannot use the credentials it obtains from users to authenticate with the BAS application server or the gateway behind it. This concern also justifies sending the credential directly to the application server from the proxy, since a BAS-destined credential will only be sent to the BAS server.

In addition to allowing redundant authentication, the authentication proxy offers the advantage of encapsulating the

enterprise authentication system details from the application, allowing them to be more easily switched or upgraded. The non-repudiable authentication credential can also help make audit logs more trustworthy, an application that we plan to explore in the future.

4.3 Gateway

The gateway is responsible for marshalling requests from the application to the BAS. Unlike the application server, the gateway is considered a trusted component and is the last line of defense against attackers to the system.

The gateway receives requests across a dedicated connection with the application server. These requests have a very structured format which allows the gateway to carefully check the integrity of the commands. Requests include the ID of the requester, the room on which an action is to take place, and arguments necessary to complete the request, such as the name of the principal to whom access to a room should be granted. Additionally, the request includes an authentication token signed by the authentication proxy. When the gateway processes requests, it checks the token to ensure both that it is fresh and that it is from a trusted source. Following these integrity checks, the gateway then enforces its specialized authorization policy by determining if the user has access to the specified room within the BAS. The gateway also makes sure that the token it receives is fresh. Currently, it rejects tokens older then 10 minutes. The gateway then processes the command on the BAS if the policy constraints are met.

The gateway interfaces with the BAS in two ways. First, it accesses the legacy database used by the BAS. Specifically, the database provides information about principals' access rights, as well as a log of recent accesses to rooms in the building. Second, the gateway uses an OPC [12] interface to operate the BAS. The OPC interface allows the gateway to control objects in the building as well as determine the current state of objects in the building. In our applications, the OPC interface is used specifically to unlock doors and return the current state of doors in the building, because door state information is not stored in the building database.

4.4 Applications

To provide a practical implementation of our architecture, we built a prototype application for door access and delegation in the Siebel Center for Computer Science. The web server was implemented in C# and ASP.net, and runs on Microsoft IIS.

When the user visits the website, the web server verifies that the user is logged in. If not, the user is redirected to the Authentication Proxy for login and returned to the web server. There the user is presented with the main menu, where she can choose to unlock doors or delegate access (grant/revoke). To unlock a door, the user provides a door ID and initiates an unlock request. The web server forwards the request to the oBIX interface, which in turn sends the request to the gateway. The result is returned to the web server via the oBIX interface. A screenshot of the unlock page is shown in Figure 6 in the appendix.

The delegation application allows a user to grant room access to a another user. The application enforces the policy that each area is assigned to an area owner, and only the area owner may delegate access privileges to other users.[2] The application also allows the user to revoke access permissions she had granted in the past. To assist the user, a list of delegated accesses previously granted by the user is displayed. This information is kept locally in a database. A screenshot of the grant/revoke access page is shown in Figure 7 in the appendix.

In designing the web application prototype, we considered the tradeoffs between usability and security. We decided to favor higher security in the prototype, at the expense of some usability features. For example, it might be convenient to show all doors accessible to the user in the unlock dialog. However, we decided not to list the accessible doors for two reasons: first, an intruder who breaks into the system by impersonating another user should not learn the access information, and second, keeping accessibility information consistent with the BAS would require database synchronization between the BAS and the application database. So, the web server only displays minimal information as a response to the request. When the user makes a request to unlock a door, it only tells her whether the request succeeded or not, without providing a reason. Similarly, in the grant/revoke access page, we do not display information about areas that can be delegated by the current user. We only show the delegation history of area accesses granted by the user, and this information is protected by a secret PIN. Thus, a malicious intruder who succeeds in breaking into the system and impersonating a user can only derive door access and ownership information through brute-force trial and error.

5. ANALYSIS

In this section, we present a risk analysis of our approach of multi-level security with redundant authentication and compare our architecture with several alternative techniques that also aim to minimize damage resulting from mistakes in the application or malicious attacks.

5.1 Risk Analysis

Our architecture introduces several new components, with complex interactions between them; potentially, each component could be a target for attack. However, the component breakdown is designed to compartmentalize any faults or attacks and to simplify the design and implementation of the security-critical components.

In particular, the application server is the most vulnerable component of our architecture: it performs sophisticated interactions with the client, presenting a large attack surface, and it contains complex and evolving application logic that can be a source of vulnerabilities. Of course, we expect that standard precautions for protecting the application server will help prevent routine break-ins, but some attacks may nevertheless succeed. Therefore, our design explicitly limits the impact of any faults in the application server: as long as the gateway and authentication proxy are both secure, even a fully compromised authentication proxy may only execute actions on behalf of recently authenticated users.

The authentication proxy and the gateway are both trustworthy components, but the design of both is relatively simple. We expect the authentication proxy can be secured in a similar way to the enterprise authentication system, and

[2]This type of policy suffices for our particular application, and is much easier to implement than allowing multiple area administrators. More complicated models such as the relational database access control model (initially proposed in [7]) could be implemented, if desired.

because the logic of the proxy is small and unchanging, we expect that vulnerabilities will be rare. And even a full compromise of the authentication proxy is tempered by the gateway since an attacker will not be able to, in our example, access the HVAC or other building functionality. The risks of vulnerabilities on the gateway are mitigated by the fact that it exports a narrow, static interface for interacting with the outside world, and this interface is only available to the application server.

5.2 Related Work

A common approach to mitigate vulnerabilities is sandboxing. Sandboxing restricts an application's interaction with its environment according to some policy. The policy is usually designed with the principle of least privilege in mind, so that the sandbox policy allows an application to perform only those operations needed for it to function. Sandboxing has been used extensively on the system call interface for an operating system [1, 6, 14, 13, 5] and in execution environments such as Java [8], but we could imagine connecting the application server directly to the BAS server with a sandbox limiting the types of commands that are allowed. The sandbox would limit the scope of damage that an application could cause, but still permits more access than necessary. For example, the sandbox would prevent a door opening application from modfiying the HVAC settings in a building, but it would not restrict the compromised application from unlocking arbitrary doors, unlike our design with redundant authentication.

An alternative to sandboxing is privilege separation [15]. This technique relies on splitting an application into two components: a privileged one and an unprivileged one, with a communication pathway between the two. This technique has been successfully shown to contain faults and attacks on the unprivileged components and prevent them from damaging the privileged process. In a sense, privilege separation can be seen as technique for splitting an application into multiple tiers, either manually or automatically [2].

Privilege separation has three disadvantages as compared with our approach. First, each new application must be separated into two components; therefore, with N applications, there will be N separate privileged components installed on the system; whereas our architecture defines reusable interfaces and components. Second, separation must be applied judiciously to improve the security of the system; for example, two trivial partitioning schemes — moving all the functionality either into the privileged or the unprivileged component and making the other a simple shim — produces no security benefit. Finally, separation may make it impossible to simultaneously satisfy the goals of keeping the privileged component simple and tolerating errors in the unprivileged component. For example, in the delegate access application, authentication and authorization of delegation commands should be done within the privileged component, otherwise there are no meaningful restrictions on what a compromised unprivileged component may do. However, this means introducing complicated logic to maintain an auxiliary policy and store the associated state inside a privileged component, contradicting the goal of simplicity.

Intrusion detection is a third technique that may be used to mitigate damage. For example, an intrusion detection system on a building automation system might look for sequences of commands that look suspicious (e.g. many sequential door unlock commands) and alert building security. Intrusion detection, however, tends to have a limited view; in particular, it would not have the redundant authentication information that is used to make authorization decisions in the gateway. Also, intrusion detection is more frequently used to look for anomalies or particular attacks, rather than enforce policy constraints, as the gateway does. However, intrusion detection systems can interoperate well with our proposed architecture: an intrusion detection system might receive audit records from the gateway that carry with them non-repudiable authentication, and thus be in a better position to detect and respond to failures. In future work, we intend to explore the issue of intrusion detection in tiered security architectures in general and building automation systems in particular.

The redundant authentication token issued by the authentication proxy can be seen as a capability authorizing the application to act on the behalf of the authenticated user. This functionality could be implemented by a real capability using an operating system such as EROS [17]. The Asbestos OS [3] can also use labeling of event processes to implement similar functionality. We chose to use a commodity operating system for our implementation for ease of deployment and to make the critical components simpler and easier to analyze.

An alternate approach to redundant authentication would be to follow an approach for single sign on, like that suggested by the Liberty Alliance [18]. In this case, the authentication proxy would provide the necessary wrapper to be an approved identity provider and both the application server and proxy would be service providers. As with redundant authentication, the application server would ask the authentication proxy to authenticate the user and would be returned a token proving the authentication. Unlike our proposed process, though, the gateway would also ask the authentication proxy directly to authenticate the user. However, this would require the gateway to have a communication channel with the authentication proxy. Despite the fact that the authentication proxy is a trusted component, the additional channel of communication with the gateway makes it more vulnerable and complex than we believe is strictly necessary.

6. CONCLUSIONS

We presented the concept of redundant authentication as a method to secure multi-tier web systems. When acting on behalf of a user, higher-tiers are required to provide non-repudiable evidence that the user authenticated recently to lower-tiers. Using this evidence, the lower-tier system is able to enforce its own security policies based on the currently authenticated user. The policy can greatly limit the power of an attacker on a compromised higher-tier system.

In order to validate redundant authentication, we presented a pair of applications that make use of information stored in a Building Automation System (BAS). Because the BAS is a sensitive system, we developed a lower-tier system, called the gateway, that is responsible for interfacing with the BAS. The gateway redundantly authenticates the users so that it can enforce its own security policy. In developing these applications for the Siebel Center, we developed an authentication proxy, which interfaces with enterprise authentication systems and provides the applications with non-repudiable tokens which are then used by the gateway

to redundantly authenticate the user.

Acknowledgements

This work was supported in part by NSF CNS05-5170 CNS05-09268 CNS05-24695, ONR N00014-04-1-0562 N00014-02-1-0715, DHS 2006-CS-001-000001 and a grant from the MacArthur Foundation. The views expressed are those of the authors only. Ragib Hasan is supported by NSF Award number 0331707 and 0331690.

7. REFERENCES

[1] A. Acharya and M. Raje. MAPbox: Using parameterized behavior classes to confine untrusted applications. In *USENIX Security Symposium*, 2000.

[2] D. Brumley and D. Song. Privtrans: Automatically partitioning programs for privilege separation. In *USENIX Security Sympoisum*, Aug. 2004.

[3] P. Efstathopoulos, M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazieres, F. Kaashoek, and R. Morris. Labels and event processes in the Asbestos operating system. In *Symposium on Operating Systems Principles*, 2005.

[4] P. Ehrlich and T. Considine (Chairs). Open Building Information Exchange (oBIX) version 1.0. OASIS Committee Specification, December 2006. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=obix`.

[5] T. Garfinkel, B. Pfaff, and M. Rosenblum. Ostia: A delegating architecture for secure system call interposition. In *Network and Distributed System Security Symposium*, 2004.

[6] I. Goldberg, D. Wagner, R. Thomas, and E. Brewer. A secure environment for untrusted helper applications. In *USENIX Security Symposium*, July 1996.

[7] P. P. Griffiths and B. W. Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems (TODS)*, 1(3):242–255, September 1976.

[8] Java. http://java.sun.com/.

[9] E. Kubaitis. Bluestem overview. Web Page, August 2000. `https://www-s4.uiuc.edu/bluestem-notes/`.

[10] K. Lawrence and C. Kaler (Chairs). Web Services Security (WS-Security) X.509 Certificate Token profile 1.1. OASIS Standard Specification, February 2006. `http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-x509TokenProfile.pdf`.

[11] Microsoft. Active directory overview. Web Page, Janurary 2005. `http://technet2.microsoft.com/windowsserver/en/library/7c981583-cf41-4e6c-b1f6-5b8863475ede1033.mspx?mfr=true`.

[12] OPC Task Force. OPC overview. OPC White Paper, October 1998. `http://www.opcfoundation.org/DownloadFile.aspx/General/OPC\%20Overview\%201.00.pdf?RI=1`.

[13] D. S. Peterson, M. Bishop, and R. Pandey. A flexible containment mechanism for executing untrusted code. In *USENIX Security Symposium*, Aug. 2002.

[14] N. Provos. Improving host security with system call policies. In *USENIX Security Symposium*, Aug. 2003.

[15] N. Provos, M. Friedl, and P. Honeyman. Preventing privilege escalation. In *USENIX Security Symposium*, Washington, DC, Aug. 2003.

[16] RSA Laboratories. Public-key cryptography standards (PKCS) #7: Cryptographic message syntax standard version 1.6. RSA Laboratories Technical Note, May 1997. `http://www.rsa.com/rsalabs/node.asp?id=2129`.

[17] J. S. Shapiro, J. M. Smith, and D. J. Farber. EROS: A fast capability system. In *Symposium on Operating Systems Principles*, 1999.

[18] T. Wason, S. Cantor, J. Hodges, J. Kemp, and P. Thompson. Liberty ID-FF architecture overview, 2005.

APPENDIX

A. APPLICATION SCREENSHOTS

Figure 6: Unlock Door page of the web interface.

Figure 7: Grant/Revoke Access page of the web interface.