IMPROVING THE SECURITY IN INTERCONNECTING BUILDING AUTOMATION SYSTEMS TO OUTSIDE NETWORKS

BY

HEE DONG JUNG

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Adviser:

Professor Carl A. Gunter

Abstract

As control systems are becoming more complex and capable with much functionality, it requires more efforts not only to maintain correct operations but also to protect them from various threats. Security of the control network which connects entities in the system and serves as a path for information transfer between them is a major cause of concern. Operators of the control systems have taken a conservative way to provide a protection to the network where it is simply isolated from other systems and networks that could introduce access channels. Even though the isolation provides a great protection, it limits management efficiency and expandability of the system. Solving the problem of providing interconnectivity as well as sufficient protection to the control network is not trivial.

Existing work proposed a solution where they applied a multi-tier web server system to the control system in the effort to provide better connectivity and introduced a concept of redundant authentication to mitigate risks to the system. In this architecture, a front end system that accepts requests from users is required to provide a non-repudiable credential of the requesting user when it passes the request to a back end proxy that has access privilege on the control system. This limits malicious actions that could be performed by the compromised front end system. It, however, forces every recently authenticated user to share the vulnerability in the case of the compromised front end system due to a requirement that clients should remain unmodified.

In this thesis, we suggest a new solution with a client program to overcome the above limitation and provide a better protection. Installation of the client program is required in order to access the control system from the outside network. With this architecture, users who have chosen to opt out by not installing the client program are safe from the risk introduced by other users who have chosen to install the program and use the service. Non-repudiable credentials are still required with every request to the control system hence containing the possible actions of the compromised front end system on the control system. We validate our strategy on Building Automation System (BAS) testbed with a practical application which allows users to unlock doors of the building.

To My Lord, my Savior, Jesus Christ, and to my loving family.

Acknowledgments

This thesis would not have been possible without the support of many people. Many thanks to my adviser, Carl A. Gunter, who always supported me in many ways and guided me through the research. Thanks to Jodie P. Boyer for setting the basis for this research, Ragib Hasan and Lars E. Olson for helping me on various aspects on completing the research. I also want to thank my lab mates, parents, and many friends for their support. Last but not the least, thanks to Eun Sun Kuk for her strong support and love.

Table of Contents

List of Figures . . . . . **vi**

List of Abbreviations . . . . . **vii**

Chapter 1 Introduction **1**

Chapter 2 Related Works **8**
2.1 General architecture of BAS 8
2.2 Background on BAS vendors, capabilities, and standards 9
2.3 Basic perimeter protection and their application to the control systems 11
2.4 Multi-tier systems and its application to BAS 12

Chapter 3 Design **16**

Chapter 4 Application and Threat Model **21**

Chapter 5 Implementation **25**
5.1 Client Program 27
5.2 Application Server 29
5.3 Gateway 31

Chapter 6 Analysis **33**
6.1 Risk Analysis 33
6.2 Possible Alternative Approaches 35

Chapter 7 Conclusion **37**

References **39**

List of Figures

1.1 The Siebel Center for Computer Science at the University of Illinois 2
1.2 Temperature Controller 3
1.3 Swipe Card and Door Lock 4
1.4 Digital Security Camera 4

2.1 Typical Building Control System 8
2.2 Typical Multi-tier System Architecture 12
2.3 User Accessible Building Automation System with Redundant Authentication 14

3.1 User Accessible Building Automation System with Client Program 18

5.1 Building Automation System Test Bed 25
5.2 Prototype Implementation 26
5.3 Client Program User Interface 28
5.4 Client Program Signing Window 28

List of Abbreviations

SCADA	Supervisory Control and Data Acquisition.
ICS	Industrial Control Systems.
VPN	Virtual Private Network.
BAS	Building Automation System.
HVAC	Heating, Ventilating and Air Conditioning.
OPC	OLE for Process Control.
SPC	Standard Project Committee.
ASHRAE	American Society of Heating, Refrigerating and Air-conditioning Engineers.
ANSI	American National Standards Institute.
oBIX	Open Building Information eXchange.
OASIS	Organization for the Advancement of Structured Information Standards.
CPNI	Centre for the Protection Of National Infrastructure.

Chapter 1

Introduction

Networked computers are widely used for large control systems which manage and regulate physical processes. Supervisory Control and Data Acquisition (SCADA) systems that control and monitor processes like electrical power transmission and distribution are one example and Industrial Control Systems (ICS) which monitor manufacturing and production processes in a factory are another. Security and reliability of these networked computer control systems are highly important due to the value of the resources that the systems control and possible disastrous consequences of their malfunction. For example, 2003 blackout which occurred throughout parts of Northeastern and Midwestern United States and Ontario, Canada illustrates possible damages on the society when the security is compromised [1].

A common way to provide protection to such systems is by isolating the control system network which connects entities in the system and serves as a path for information transfer between them from other less secure public networks including the Internet. This isolation, often called an air gap, provides the protection by limiting access to the control network. Only authorized operators with physical access to the system have access to the control system network. Therefore, it requires human operators to present at the physical location and manually manage the control system making the task inconvenient and inefficient. Operators often make compromises to perform their job more conveniently and efficiently. Typical compromises are remote management by Virtual Private Network (VPN) connection to the control systems and multi-homed systems that connect to both the control and the public networks. Above compromises, however, introduce new vulnerabilities to the system and if

used without care, they can be used as means to compromised the whole system. Since VPN connections have known vulnerabilities [13, 18], the operators must address them before they use VPN. Also, multi-homed management console has access channels from outside networks which broaden attack surface to the control system. Compromised VPN and multi-homed management console can undermine the security of the control system and may lead to devastating results.

In addition to making the control system management inconvenient and inefficient, the isolation prevents support for various applications. In case of Building Automation System (BAS) which controls the functions of a building including door locks, video surveillance, and the Heating, Ventilating and Air Conditioning (HVAC), the isolation limits useful applications. For example, temperature control using web service, unlocking doors through web pages and granting access to rooms without manual configuration of the control systems. Therefore, it is desirable to provide a different security mechanism that can provide convenience and support for applications as well as a great protection to the control systems.

Figure 1.1: The Siebel Center for Computer Science at the University of Illinois

Figure 1.2: Temperature Controller

To illustrate the above state in more detail, we present a case study of one example of the networked computers control systems. The Siebel Center, Computer Science Department building at the University of Illinois at Urbana-Champaign, is an example of BAS. Figure 1.1 shows the outside look of the building. It is a 225,000 square foot building that was built in 2004. The Siebel Center uses Andover Continuum from Advanced Control Corporation for its BAS management. As described above, the control system of the Siebel Center manages and regulates various functions in the building. These functions include HVAC system, door lock system, and surveillance system. Figure 1.2 shows a temperature controller on a wall. This device enables users to manually set temperature of a room. This controller enables users to manually set temperature of a room. The operator of the building has privilege to set high and low limits on the controller so that the users cannot intentionally or unintentionally set the temperature to an inappropriate degree. The Siebel Center uses swipe cards to open doors to rooms as shown in figure 1.3. When the card is swiped, information on the card is sent to the control system to verify whether it has privilege to open the door. To add a new access privilege or to remove access privilege, the operator must make the change

Figure 1.3: Swipe Card and Door Lock

Figure 1.4: Digital Security Camera

to the management system manually. Moreover, since requests for these changes usually come through enterprise network or the Internet, the operator must transfer the data to the control system manually. Therefore, it is onerous for the operators if a lot of changes occur. Another important feature of BAS is video surveillance with security cameras. The Siebel center has several digital security cameras at strategic locations covering building exits and hallways. Figure 1.4 shows one of them. Although they are not always monitored by human, audit records can be used in case of theft.

Even though the Siebel Center already has many functions, some interesting and useful functions are not possible due to the isolation of the control network. A software doorbell function could be one example. Since there is no physical doorbell for each room in Siebel Center, a person without access privilege to the room must knock on the door until someone comes out. It becomes a problem when the room is big and the door is far from where people

stay. If there were connection between the building control network and the Internet, we can think of a smart phone application or a web site which allows a user to input a room number. It can then connect to the control system and finds out people who recently swiped into the room and send some type of alerts to answer the door. Another example is opening doors through web services or smart phone applications. Surprisingly many people forget to take their cards when leaving the room and could not get in. Sometimes people leave their cards at home. It is very useful to have the door opening web service or application in these situations. The application can allow people to enter a door name with some type of identification and then send that information to the control system for verification using the connection between the control network and the Internet. The control system will open the door after the verification. (An advanced example of such an application is the Grey authorization system [6, 7].) However, these new functions that are possible from the interconnectivity must not jeopardize the security of the control system. Vulnerabilities introduced by the connection to the outside networks have to be addressed beforehand.

Multi-tier client-server system which is widely used in e-commerce and enterprise information applications can be applied to the control systems in order to provide the convenience and the support for useful applications. But simply applying the multi-tier system to the control system may introduce new vulnerabilities. Existing work [8] suggested a way to improve security in applying the multi-tier systems. Their architecture consists of three tiers and functionality is distributed among them. Higher tier or front end accepts connections from clients and middle tier acts as a proxy and connects the higher tier to lower tier or back end. The control system itself is the lower tier. Their system enforces the principle of least privilege on the higher tier such that the higher tier can only access the lower tier's functions and data that are required for its mission. This is achieved by demanding a non-repudiable credential of a client at the middle tier. The client first authenticates to the front end and redundantly authenticates itself to a special proxy which provides the non-repudiable credential to the front end. The front end sends the request from the client with the non-repudiable credential to the middle tier. The middle tier or the proxy only passes the request from the

front end to the back end when it comes with the non-repudiable credential of the requesting client. The principle of least privilege is enforced since the higher tier's actions are limited by the non-repudiable credential, whereas higher tier in normal multi-tier system usually has unlimited access. Their approach, therefore, contains possible malicious actions of the higher tier if it is compromised.

Even though this strategy achieves the goal of providing certain level of protection in providing the interconnectivity to the control system, it forces every recently authenticated user in the system to share the vulnerability introduced by the new architecture. This inherits from a requirement that the client should remain unchanged. Because of the requirement, their architecture relies on a special proxy to generate the non-repudiable credentials that are not tied with every request. Instead, the credential is tied with each client and could be used with any requests from the same client until it is expired. Therefore, if the higher tier gets compromised, it can use the unexpired credentials from the recently authenticated users to send bogus requests to the middle tier pretending them to be coming from the legitimate clients.

In this thesis, we introduce a new architecture that provides more protection with the help of a client program. Users who want to use the new system must install the client program. Those users who decide not to install the program cannot use the system but they are not exposed to new vulnerabilities introduced to the users who have chosen to install the program and use the system. Our new system also requires a non-repudiable credential with every access of the higher tier to the lower tier to limit the possible actions. In our architecture, however, the client program provides the non-repudiable credential and it is tightly tied with every request. Therefore, it is not possible for the compromised higher tier to use the credentials to create new fake requests. The most it can do is to reuse recently executed requests which are probably not very useful.

We applied our strategy to our BAS test bed for validation. In particular, we implemented an application which allows users to send an unlock door request to a higher tier application server which serves as a front end to the BAS control networks. A middle tier gateway

proxy enforces least privilege by requiring the non-repudiable credential with the request from the higher tier application server. The client program uses public key cryptography digital signature to sign the request and to provide the non-repudiable credential. The middle tier gateway proxy first verifies the credential and passes the request to the control system or rejects the request depending on the verification status. We were able to open a door using our application.

This thesis consists of 6 chapters. Next Chapter explains backgrounds and related works. In Chapter 3, we introduce our new strategy by describing the architectural design of the system. Chapter 4 discusses the application, its requirements and threat model. In Chapter 5, we describe the actual implementation of the system. We then analyze our strategy in Chapter 6. Finally we conclude in Chapter 7.

Chapter 2

Related Works

In this Chapter, we will first describe general architecture of Building Automation System (BAS) and then give some background on current BAS vendors, capabilities, and standards. Next, we will briefly talk about ways to control access to a network from potentially vulnerable or malicious computers and efforts to apply them to the control networks. Lastly, we will explain the work by Boyer et al. [8] in more detail since it is closely related to this work.

2.1 General architecture of BAS

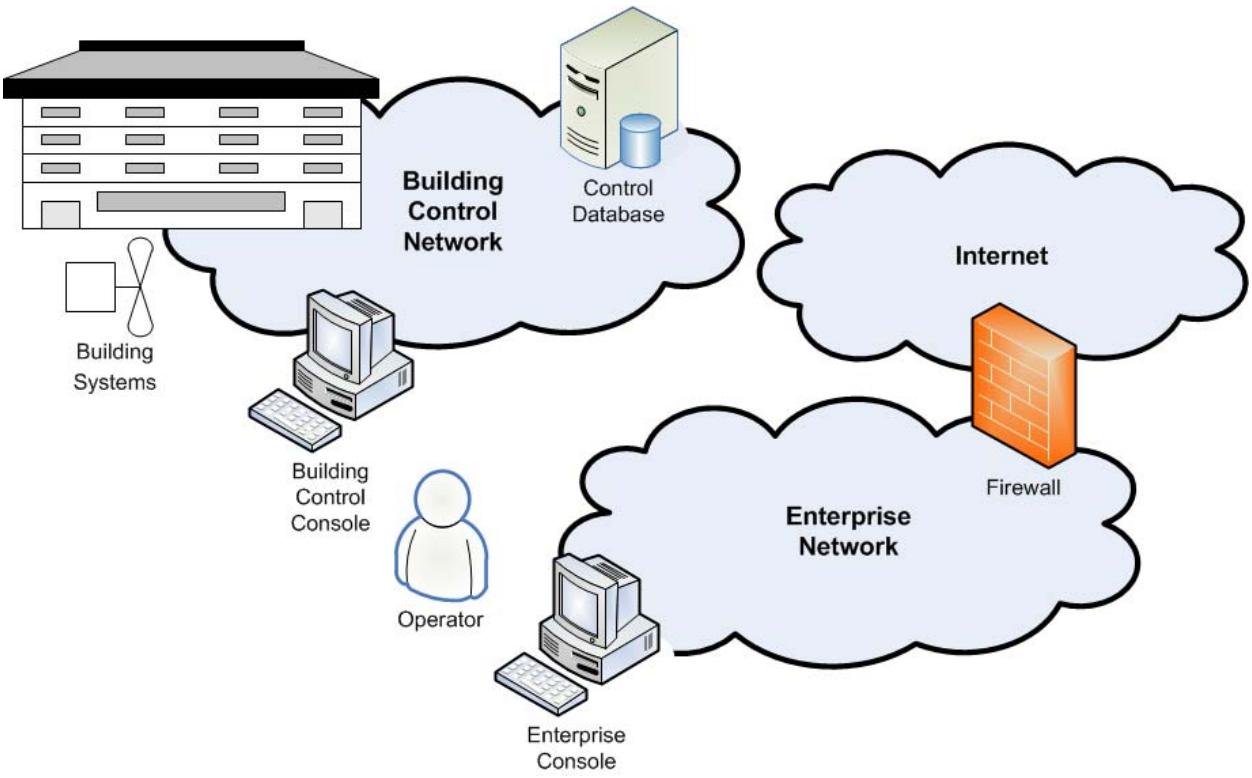

Figure 2.1: Typical Building Control System

BAS is an example of the networked computers control systems that manages functionalities of a building. These functionalities include lighting control, HVAC system, door lock/unlock management, and alarm system. Since compromise of these functionalities can bring severe damages including unlocking of doors and turning off the alarm system, the building control network is usually isolated from other networks. Figure 2.1 shows a general architecture of the building control system. Building control network is separated from enterprise network and the Internet. A control console machine which has applications to manage the building functionalities is attached to the control network. Control database contains information regarding who has access to which rooms and current temperature settings of each room. The enterprise network connects computers used by people who work in the building and provides access to the Internet with some protection measure like firewall. A building operator's job is to monitor and control the BAS and keep the database up to date.

Even though this isolation of the control network provides a great protection, it makes the operators to conduct their job manually because they need to use the specific control console. For instance, when there is a request to adjust temperature of the building or to grant an access to a room to someone, the operator must sit on the control console to make the change. These requests usually come to the operator via e-mail or some kind of web request services. However, this architecture does not allow them to be processed efficiently and automatically since the control network is isolated. In this thesis, we suggest a new architecture that not only alleviates the burden on building control system management due to the isolation of the control network but also effectively mitigates risks to the architecture.

2.2 Background on BAS vendors, capabilities, and standards

Some of the major BAS vendors are: Siemens Building Technologies, Honeywell Building Control Systems, Johnson Controls, and Schneider Electric, formerly TAC. Recently, big companies like IBM Corporation and Cisco Systems, Inc. also joined the BAS industry.

They all provide some kind of building management systems that allow the operators to have more control and easier access to all building systems. Capabilities of these solutions seem to include secure, extensible, and open systems. However, looking at them closely reveals that those approaches are primitive. For example, some seems to offer SSL connections to the control networks but limiting the use to the operators where others offer web access but without open API. Although some vendors are working on more programmable and better networked systems, most of the vendors are more concerned on the security and reliability of BAS. Therefore, current BAS solutions take a conservative state and building owners and the operators seem to accept it.

BACnet, LonTalk, Modbus, oBIX, and OLE for Process Control (OPC) are main industry standards and communications protocols for BAS. Solutions from some vendors have support for multiple protocols. For example, APOGEE Building Automation solution from Siemens natively support BACnet but it also provide integration options for OPC, Modbus, and LonTalk. Proprietary protocols are also used by others. Now let us look at some of these protocols more closely.

BACnet is one of the main standard protocols. The development of BACnet began in 1987 by American Society of Heating, Refrigerating and Air-conditioning Engineers (ASHRAE). It became ASHRAE and American National Standards Institute (ANSI) Standard in 1995. It was also standardized by ISO in 2003. The BACnet protocol specifies both how to represent data on the network and the services that are used to transfer data from one BACnet node to another. BACnet defines all data on the network in terms of "objects", "properties", and "services". Objects represent physical inputs, outputs, and logical grouping of points that perform some function. These objects have prescribed properties and they are monitored and controlled through these properties. The BAS uses defined services to access a property of an object or to request an action from an object [2].

LonTalk is another major standard protocol used in BAS besides BACnet. It was developed by a company called Echelon Corporation and accepted as ANSI Standard in 1999. Subsequently, the communications protocol has been accepted as ISO/IEC global standard,

European standard, and Chinese standard. The Lontalk protocol implements all seven layers of the OSI network model. It is implemented with a mixture of hardware and firmware on a silicon chip. This is to eliminate any possibility of modification to the protocol. At first, an Echelon Corporation-designed IC chip was required to implement the LonTalk protocol. However, the protocol became available for general purpose processors in 1999 [3, 10].

The oBIX (Open Building Information eXchange) standard is web service-based interfaces to building control systems. It is an open standard developed around XML to support web services and service oriented architecture. It enables communications between mechanical and electrical control systems in buildings and enterprise applications. It is developed and managed by the Organization for the Advancement of Structured Information Standards (OASIS) oBIX Technical Committee. In oBIX, all information and BAS entities are represented as objects in XML. Each object consists of attributes or valid XML elements. These valid XML elements also have attributes. Objects are queried and modified using web services. A distinguishing aspect of oBIX is support for a publish/subscribe service on objects which allows the objects to be accessed by a defined polling rate in a client-server based system.

2.3 Basic perimeter protection and their application to the control systems

A common practice to provide a protection to a network from outside access is to place a machine at entry points to the network. The machine performs a security control on every access from the outside. There are various types of this machine. The most common one is a filtering firewall. It inspects packets coming into the network and decides whether to allow the packets based on a set of rules. It can also restrict the ports that could not be used to access the network. Another type is one that serves as an endpoint of VPN. It uses authentication to deny access to unauthorized users and decrypts packets which were encrypted to prevent eavesdropping. This machine can also be a proxy server which evaluates

requests from clients before they reach the actual server. It is similar to the filtering firewall but rather than filtering at the packet layer, it works at application layer. Therefore, the proxy server must have detailed knowledge about the applications to evaluate the requests.

There have been efforts to apply these methods on the isolated control networks. For example, Tofino Industrial Security Solution provides firewall protection to the control systems. Rules can be defined to specify which network devices are allowed to communicate and what protocols they may use. In March 2006, American Gas Association released a report [5] on Cryptographic Protection of SCADA Communications. Although the main purpose of the report is to provide a guideline for voluntary implementation of a comprehensive cyber security posture, VPN is suggested for encryption and authentication of SCADA communications. A recent report [11] from Centre for the Protection of National Infrastructure (CPNI) and the Department of Homeland Security recommended good security practices in supporting remote access to industrial control systems. Among the many guidelines, it recommends the use of firewalls and VPNs.

2.4 Multi-tier systems and its application to BAS

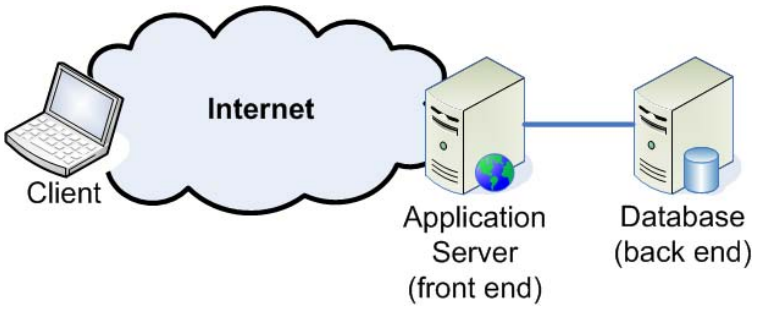

Figure 2.2: Typical Multi-tier System Architecture

In the effort to provide both interconnectivity and protection to BAS, Boyer et al. [8] introduced an architecture that employs multi-tier client-server architecture to BAS. Multi-tier architecture is widely used in e-commerce and enterprise information applications. It separates service into difference modules to provide a flexibility and reusability. For example,

an application that allows access to personal bank accounts through the Internet can be broken into user interface, application process logic, and back end system such as a database. Each of these modules can be modified without the change of the other and they can be reused in similar services as well. Figure 2.2 illustrates this architecture. Clients use user interface such as web browsers to request service to an application server. The application server, known as a higher tier or a front end, is in charge of authenticating the clients and determining whether the requests are authorized for the clients. If the requests are legitimate, the application server forwards the request to the database, a lower tier or a back end. And the database serves the requests by retrieving or updating the relevant information. In this architecture, the application server acts as the proxy server described above and provides the protection to the system behind it. However, if the application server is compromised, the system behind it is exposed to a serious threat. Because the front end passes requests to the back end on behalf of many different clients, it usually has general access to the back end. Therefore, the compromised front end can send malicious requests to the back end pretending them to be coming from legitimate users. The back end does not have means to identify whether those requests are legitimate or fake. In case of e-commerce, the compromised application server can retrieve personal information like credit card numbers and passwords from the database.

Authors have introduced a new architecture that can mitigate the effect of the compromised front end in the multi-tier systems. Figure 2.3 illustrates the architecture on typical BAS. As we can see from the figure, it has another proxy between the front end application server and the back end control system. It is called gateway and its job is to enforce the principle of least privilege on the application server to restrict its access to the back end control system. In other words, it forces the application server to access only the functions and data that are necessary to serve the client's request from the back end control system. Redundant authentication is their mechanism to enforce the principle of least privilege. The gateway demands proof of authentication with every request from the application server and restricts the access privileges according to the specified principal. Each user must authenti-

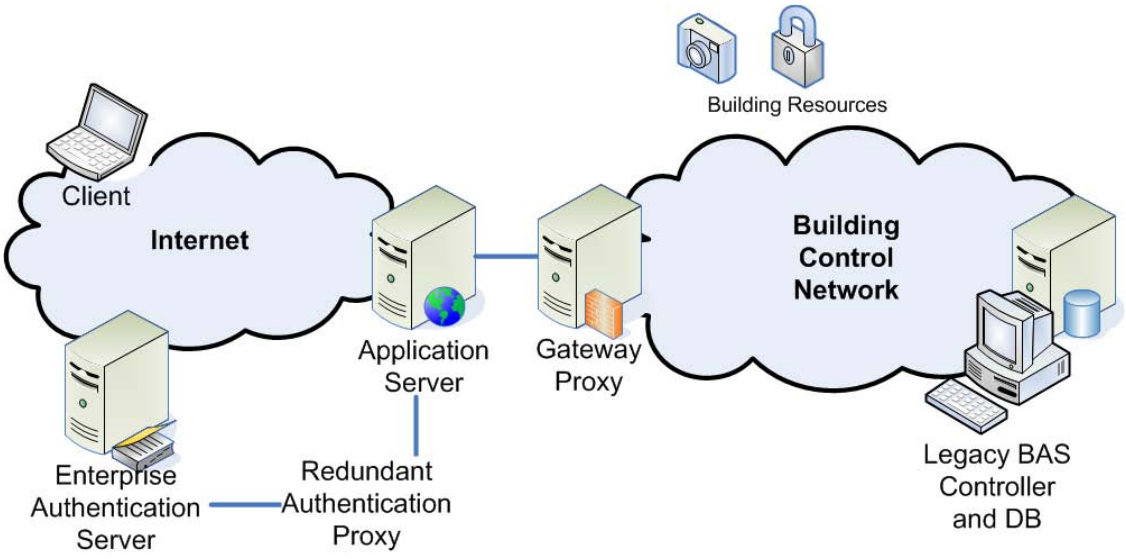

Figure 2.3: User Accessible Building Automation System with Redundant Authentication

cate themselves to the front end application server and the application server must show the proof of authentication to the gateway hence it is called the redundant authentication. In order to prevent the application server from forging the proof, they incorporated enterprise authentication with a proxy server to provide the proof such a way that the application server cannot tamper with the proof. Therefore, the compromised application server cannot request a command like "open all doors" without the proof of the authorized personal. This significantly improves the protection to the multi-tier system. In addition, this gateway is also capable of performing the application layer firewall by enforcing a set of rules on each application. These rules can be simple as whitelists and blacklist or more complex.

The role of the authentication proof is critical in this architecture. It must be non-repudiable. In other words, when the application server passes a request with this credential, it must be able to convince the gateway that it is from the user who made the request. Since the most enterprise systems do not support non-repudiable credentials, they used another proxy called authentication proxy to translate the enterprise authentication credentials to non-repudiable credentials. When a user requests a service to the application server, the user is also asked to authenticate to the enterprise authentication server. The enterprise authentication then provides the authentication credential to the authentication proxy which

creates a non-repudiable credential based on the enterprise credential. The authentication proxy returns this non-repudiable credential to the application server so that the application server can send both the request and the credential to the gateway. The gateway limits the privilege of the application server based on the credential provided.

This architecture shows a way to provide user accessibility to the control systems with an effective protective mechanism. The authors have validated the architecture by applying it to the BAS. However, we have found some limitations of the architecture and we suggest a new architecture that overcomes those limitations and provides an even better protection with same user accessibility to the control systems.

Chapter 3

Design

One of the limitations of the work by Boyer et al. [8] is that if the application server gets compromised, all the recently authenticated users are under the risk. A lot of users may concurrently send requests to the application server and to pass the requests to the gateway, the application server needs non-repudiable credentials from all those users. The credentials must go through the application server because there are no other channels to the gateway. Once an attacker takes over the application server, the attacker can use the credentials that are not expired and send false requests on behalf of recently authenticated users. Setting the expiration time short might help since it leaves not much time for the attacker to use those credentials. However, if it is too short, the users have to authenticate repeatedly causing inconvenience. Therefore, it is hard to find the perfect expiration duration. Although their work provides better protection than a simple multi-tier system where the compromised application server has full privilege over the back end system, the attacker, in the worst case, might be able to do almost everything with a combination of various privileges of different users.

In addition, their architecture relies on the enterprise authentication to conduct user authentication and provide credentials to the authentication proxy for generating non-repudiable credentials. It implies that the enterprise authentication server and the proxy must be trusted. Otherwise, the gateway cannot rely on the non-repudiable credentials created by those two components. Moreover, they assume that the gateway is trusted as well. They claim that since the application proxy and the gateway are small, special purpose and isolated implementations, they can conform to stringent security requirements. Although it

is generally true, we believe that it is always better to have less number of trusted components.

In the effort to overcome above limitations, we introduce a new architecture that provides better protection to the control systems in the case of the compromised application server and that allows interconnectivity from outside networks. Consequently, our design provides convenience to the control system management and support for useful applications with increased protection. We chose to use client slide program to achieve the above goal. More specifically, we require a client program which serves the purpose of providing the non-repudiable credentials that were used in the existing work. We still utilize the non-repudiable credential to enforce the principle of least privilege on the higher tier. However, in our case, the client program provides this non-repudiable credential instead of the enterprise authentication server and the authentication proxy. Moreover, in our case, this non-repudiable credential is tightly coupled with not only the user making a request but the request itself. Therefore, even though the application server passes the credential with the request to the gateway, the compromised application server cannot use the credential to perform any other actions than the exact request which the credential is tied to. Consequently, the scope of possible actions of the compromised application server is greatly reduced from access privileges of recently authenticated users to the recent requests. Moreover, our architecture does not require the enterprise authentication server and the authentication proxy hence reducing the number of trusted components of the system.

Our architecture could be applied in almost any multi-tier systems where client are capable of installing the client side program. To give better understanding of how it could be applied in real systems, we illustrate our architecture integrated with a typical BAS in Figure 3.1. As the figure shows, it consists of three tiers. The application server is the top-tier or the front end, the gateway proxy forms the middle-tier and the legacy BAS controller and database represents the bottom-tier or the back end. There is only one access channel from the outside to the building control network which is from the application server to the gateway proxy. Users use the client program to send requests to the application server

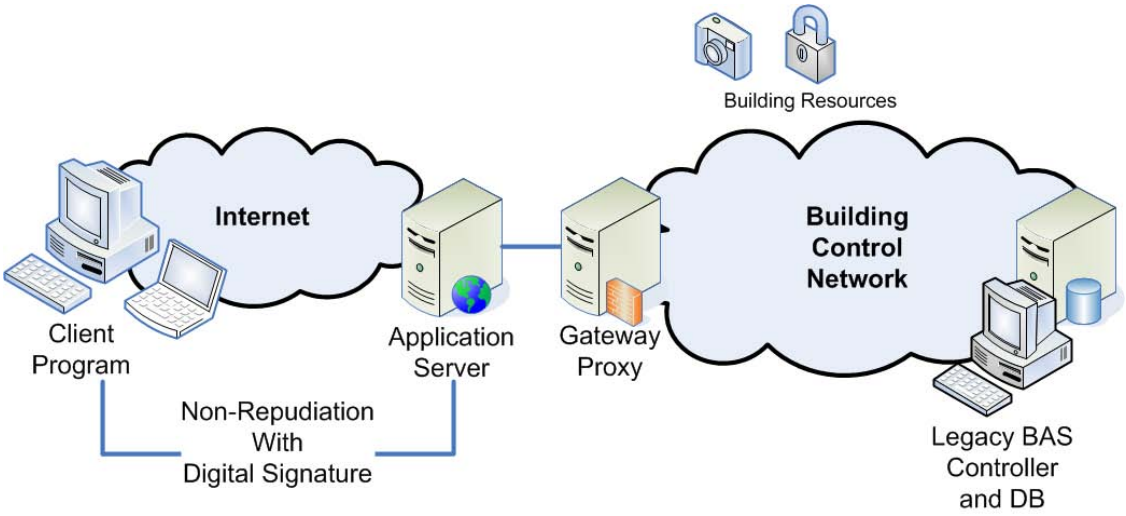

Figure 3.1: User Accessible Building Automation System with Client Program

over the Internet or enterprise network. As mentioned above, the enterprise authentication server and the authentication proxy are not needed in our architecture. However, the gateway still requires the non-repudiable credential to enforce the principle of least privilege on the application server. The client program provides this credential using digital signature scheme. Every request from the application server to the gateway is digitally signed by the client program. Therefore, it is not possible for the compromised application server to reuse the credential to make bogus requests, whereas it is the case in the existing work. In fact, the most the compromised application server can do is to re-send the recent requests before they expire. They are probably of little use for the attacker since the clients would not send requests like "open all doors of the building" or "turn all security cameras off." With the help of the client program we significantly mitigate the risk to the control system, even more than the existing work.

We now describe the process in detail. First of all, a user who wants to use the new service must install the client program which allows the user to connect to the application server. When the user executes the program, it asks for information that is required to make a request to the control system. After the information is obtained, the program connects to the application server and sends the request over the Internet or the enterprise network. Upon receiving the request, the application server translates the client request to a low level

query that is understood by the gateway and the control system. This low level query is then sent back to the client for verification. The client program reads the query and displays it to the user in a user readable form so that the user can verify that the query is the exact translation of the request. This is necessary to prevent the compromised application server from sending a fake query back to the client for a signature and using it to achieve a malicious goal. For example, when a user asks to open a door to his/her office, the compromised application server can change it to open an entrance door and send back to the user for the signature. If the client program does not show the returned request, the user will blindly sign the request thinking it will open the office door but the entrance door will be opened instead to allow an intruder. After the user verifies the query and digitally signs it to tie the query with the user's non-repudiable credential, the signed query is sent back to the application server. It then passes the signed query to the gateway. The gateway first checks validity of the digital signature and if valid, it checks whether the current request is authorized to the associated user with its policy. If the signature is valid and the request is authorized, the gateway executes the query by sending a message to the control system. Otherwise the gateway rejects the request.

One advantage of our system is that users have a choice to decide whether to use the service or not. It is important since only those users who chose to use the service share vulnerability introduced by the interconnectivity. In other words, users who opt out of the service do not risk anything in the case of the compromised application server or the client machine. Users can opt out simply by not installing the client program. Compromised client machine cannot connect and send requests to the application server without the client program. Moreover, the compromised application server cannot exploit clients who do not have the program installed. The existing work, however, forces every user to share the vulnerability introduced by the architecture. This is because one of their requirements is to keep the client unmodified. We decided to relax the requirement since most client devices including laptops or cell phones are capable of installing a program. By relaxing this requirement, we could limit the scope of the risk only to those users who opt in to

use the service and enhance the security of the control system in case of the compromised application server.

Chapter 4

Application and Threat Model

It is important to analyze threats to a new system and describe the threat model. However, we will first describe an application that we developed for the implementation of our system to help better understanding of the threats. After we discuss the threats and the threat model, we will show how our system effectively mitigates the threats.

Even though our system could be applied to different control systems, we focused on the building automation system for our implementation. The application we developed illustrates how the interconnectivity from our design provides efficiency to the building management and supports useful applications. Our application allows users of the building to unlock doors for which they have been granted access via a program installed on their machines including laptops and smart phones. Normally, they open doors with their identification cards. If they have been granted a privilege to open the door then they can unlock the door with their cards. If they do not the privilege, the door stays locked. People sometimes forget to take their cards with them and thus cannot get in to their offices. In that case, the building operator or manager either makes a temporary access card for them or use the master card to open the door for them. This takes away much time from the operator. However, our application allows the operator to save the time spend on these occasions by allowing the users to open the doors by themselves. However, it does not allow them to open doors that they do not already have privilege to open. The access privilege of each user is usually the same when using our application. But it is also possible to limit the privilege if necessary with the gateway policy. In addition, we require that every access through the application to the control system must be audited. It should be analyzed to ensure that there are no

new security violations to the building system using the new service. It helps us to deal with new attacks to the control system promptly.

Here is an example scenario which shows why the interconnectivity helps. A user, Alice comes out of her office to go to a meeting. After the meeting she finds out that she left her identification card in the office. The only way to get in is to get a temporary card from building operations center because she uses the office alone. Depending on the busyness of the center, it might take up to an hour to get the temporary card. However, if she could use our application, she can go in to her office less than a minute. All she needs to do is execute the program to send request to the building control system to open the door to her office and wait few seconds. If there were no interconnectivity between the building control network and the public network, both Alice and the building operator could not have saved their time.

Because the application provides a direct connection to the building control system, it is important to consider possible attacks and discuss the countermeasures. The most important concern in providing the interconnectivity to the building control system is that it may leave the system vulnerable. In other words, the interconnectivity might be used as a channel for attacking the control system. In the worst case, attackers might be able to gain complete control over the building control system. However, it is more likely that the attackers exploit the application and use it in their favor. In that case, the attackers could launch attacks like opening doors to the building entrances or critical rooms. These might lead to physical attacks to the building including stealing of expensive equipment and files which have sensitive information. Another possible threat is a denial of service attack. The attackers could launch the denial of service attack on the application server which accepts connections from users and make the service unavailable. In addition, it might be possible for the attackers to remove all access privileges from all doors of the building, rendering the building not accessible. Moreover, if the attackers were able to gain the control over the building control system, they could manipulate other functionalities of the building automation system. For example, they could set the temperature of the building

at a certain degree and disallow any changes, making the building unusable or they could disable security cameras.

The multi-tier architecture with the enforcement of the principle of least privilege on the application server would be able to mitigate the threat of the attackers taking complete control over the building control system. Our system only allows door unlocking requests to pass through the gateway proxy, hence it is not feasible for the attackers to send commands like disabling the security cameras or changing the temperature and make them work. Unless there is a serious bug in the application, it is not possible for the attackers to manipulate other building functionalities. However, the enforcement of the principle of least privilege cannot address the threats to the door unlocking system. To mitigate the threats to the door unlocking system, we use the non-repudiable credentials that ties users with the requests that they make. This allows the gateway to enforce a policy that rejects unlocking requests when the user making the request has not been granted access within the BAS to the door on which they are performing an action. In other words, the gateway checks whether the user making the request is allowed to unlock the same door physically using their identification card. This means that our application does not give more access privileges to the users than they already have. Therefore, the attackers must acquire the non-repudiable credentials of many different users in order to unlock arbitrary doors. Or they must acquire the credentials of high privilege users to open doors to critical rooms like the building control center. Even though our architecture do not provide countermeasures for the denial of service attack on the application server, many existing solutions including filtering based on profiling [28, 27, 19], rate limiting based on Reverse Turing Tests [22, 17], and payment based defenses [21, 30, 24, 16, 29] can be applied to our system.

Our architecture has three main components: the client program, the application server, and the gateway proxy. Among these components, we assume that the client program and the application server could be compromised by the attackers but not the gateway proxy. This is a reasonable assumption because the gateway proxy is a small, special purpose program which can comply with strict security requirements. In addition, the gateway is

isolated from the outside networks and the only path is from the application server which the gateway does not fully trust. However, it is relatively easier to compromise the client program and the application server since they are exposed to the public networks. Because of the use of the non-repudiable credentials with every request to the control system in our architecture, the possible actions that the compromised application server can perform are very limited. All it can do is to replay the recently used requests until the credentials expire. Because our application does not give more access privilege to individuals than the privilege that they already have within the BAS, the compromised client program is only as powerful as the individual associated with the program. Even though it is dangerous if a client program of an individual with high access privilege is compromised, the gateway can enforce an attenuation of privilege policy to reduce the privilege of the individual when using the application. For example, the policy can specify which doors can be opened using our application. In this case, the gateway can reject any requests to open doors to important places.

Chapter 5

Implementation

Figure 5.1: Building Automation System Test Bed

We implemented a prototype of our system on a building automation system test bed which is a small scale version of the actual BAS used for the Siebel Center for Computer Science at the University of Illinois. This prototype implementation shows the practicality of our design. Figure 5.1 shows the test bed located in Illinois Security Lab. This test bed

allows us to simulate a building with two doors and up to two areas. Each area could have more than one door associated with it and people who have access to an area can open all the doors associated with the area. One door of the test bed has a swipe card reader and the other door has a proximity card reader. The swipe card reader is shown in the bottom middle of the figure. This test bed can handle any number of users, although we have registered four users for our implementation. The test bed is in a stand-alone mode, meaning that there is only one management console connected to the BAS. The computer on the left of the figure is the management console and runs the Andover Continuum System. One limitation of the test bed is that it can only simulate the access control functions of the Andover Continuum System, but this is enough for our prototype implementation.

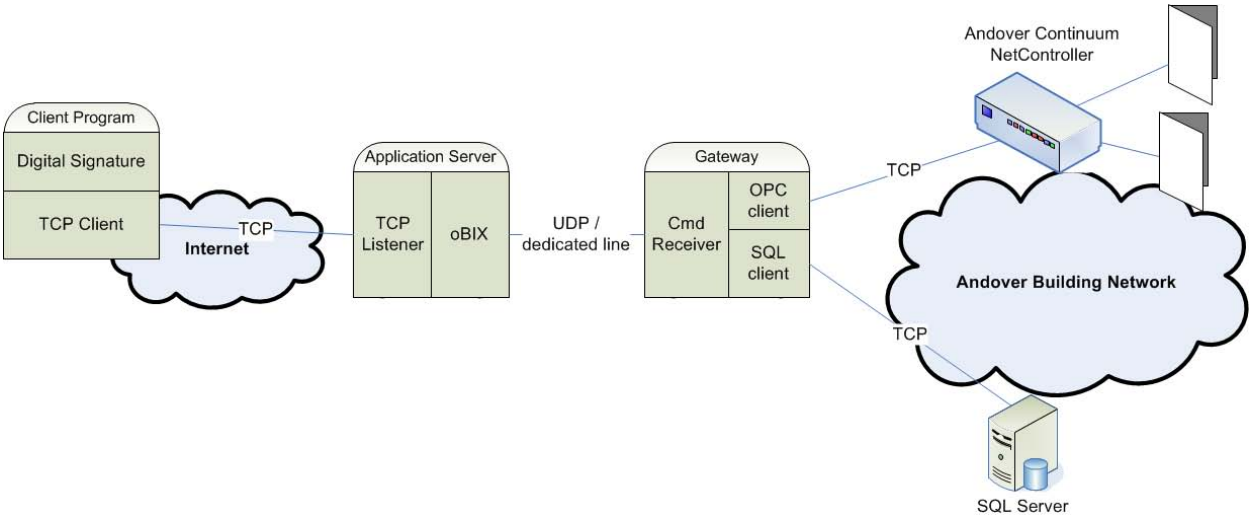

Figure 5.2: Prototype Implementation

The overview of our implementation of the application discussed in the previous chapter is illustrated in Figure 5.2. This shows the implementation choices we made to satisfy the requirements of Chapter 4. A user who wants to open a door using our application uses the client program to make a TCP connection to the application server. After the connection is made, the client program sends a request to open a door with the user id and the door name. The details of the client program and the communication process between the client program and the application server is described in Section 5.1. The application server forms

a command based on the information received from the client and sends it back to the client for a digital signature. The user must verify the command from the application server before signing it. After the command has been signed, the application server passes it to the gateway with a low-level oBIX query that the gateway understands using a dedicated communication line. The application server and how it utilizes the oBIX to communicate with the gateway is described in Section 5.2.

The gateway proxy, described in Section 5.3, is a trusted entity that enforces the principle of least privilege on every request to the building control system. It is also capable of enforcing its own access control policy independent from the building control system policy. The policy data such as a black list and access control lists are stored in a database located within the building control network, and can be queried using standard SQL. The gateway first checks the integrity of the command using the digital signature scheme and verifies whether the command satisfies its policy. After the verification, the command is translated into the OPC standard and sent to the Andover Continuum NetController which send an unlock command to the appropriate door.

5.1 Client Program

The Client program is a critical component of our application. First of all, it is not possible to use the application without the client program. This gives users a choice on whether to use the application or not. If the client program or the application server gets compromised, those who use the application might be at risk. However, those who do not have the client program are not exposed to the risk. In addition, it provides the non-repudiation proof to the gateway via digital signature. This is required to enforce access control on every request to the gateway.

This client program is currently written as Windows Forms Application in C# language. Figure 5.3 shows an initial user interface form that appears when the client program starts. On the left, it has two input boxes for a user ID and a door name. The user must fill them

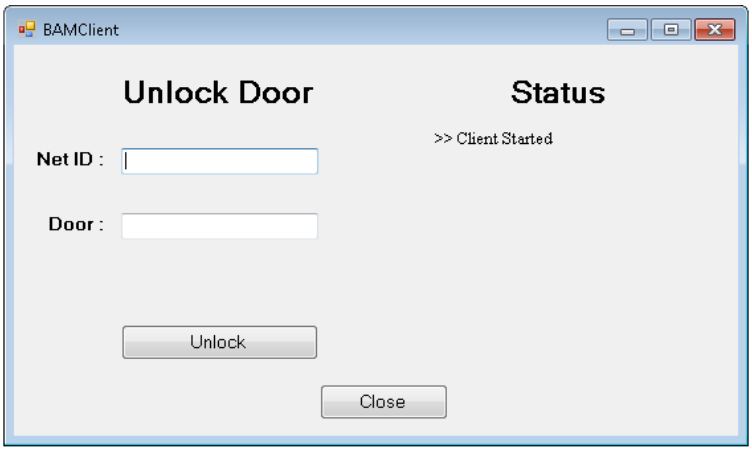

Figure 5.3: Client Program User Interface

before clicking the unlock button below to connect to the application server. Upon clicking the unlock button, the client program creates TCP connection to the application server and sends the user ID and the door name entered in the input boxes. After the connection is made, the unlock button is disabled to prevent another connection. Status window on the right side displays important messages about actions performed by the client program. For instance, "Server Connected" message is displayed when the connection to the application server is made successfully. Also, if an error occurs, an error message is displayed.

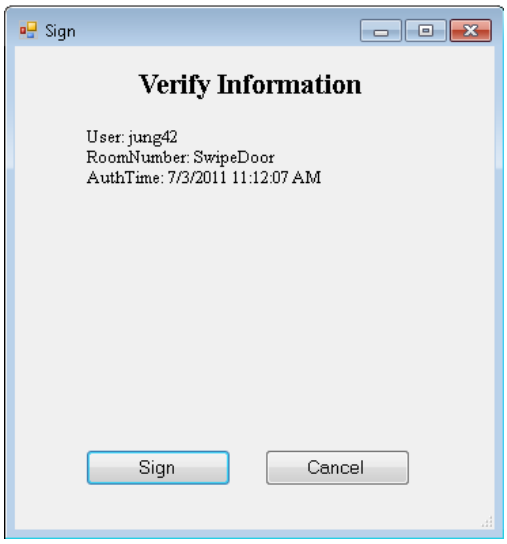

Figure 5.4: Client Program Signing Window

The application server creates a temporary file and writes the information received from

the client program. It also writes the current time to calculate the expiration time of the request. This file is then sent back to the client. The client program displays the contents of the file on a new window. This window is shown in Figure 5.4. As shown in the figure, user ID, room number (or door name), and the authentication time are displayed. If the information is correct, user clicks the sign button on the left. The client program then digitally signs the command with the private key of the user and sends the digital signature attached command to the application server. It waits for a response from the application server. If everything goes well, the door will be unlocked with a success message box and if not, a failure message box will be displayed. If the information shown is different from what the user entered in the previous window, the user can cancel the process by clicking a cancel button on the right.

The gateway must maintain trustworthy copies of the public keys of all users who use the application in order to verify the signatures. One way is to use a trusted third party certificate authority to issue digital certificates. However, this method must allow the gateway to have a channel to the third party certificate authority. This channel must be secured and dedicated otherwise it could be used by attackers to gain access to the gateway to compromise it. Alternatively, the building operator can create and issue a special key pair to each user only to be used for this application. The operator can issue the key pair physically on a digital medium like a flash drive when users first register and get their identification card. This method is relatively safer than using the certificate authority since there is no additional outside connection to the gateway proxy. Also, it is easier for the operator and the user to revoke the key pair. They can simply discard the key pair because it is independent from other uses.

5.2 Application Server

The main role of the application server is to accept connections from multiple clients and pass the requests to the gateway proxy. It is basically a multi-threaded web server implemented

in C# that accepts TCP connections on a special port for the application. It listens on this port for incoming connection requests and when the request comes from a client it creates a new TCP connection and a client state to store information about the client. It is a temporary state which is maintained until the application terminates. With the information the application server produces a command which will be sent to the gateway after signed by the client and writes the command in a temporary file. The application server then sends the file to the client for verification of the contents of the file and for the digital signature on the command. After the client returns the signed command, it passes the command to the gateway proxy using the oBIX XML language.

The oBIX provides an XML-based interface for operating mechanical and electrical control systems in buildings [12]. Recently, the purpose of oBIX has been generalized to include any type of embedded software systems. General object model and a set of operations on these objects are defined in the specification. Although any kind of objects can be represented with this model, we decided to use our own XML specifications that are more appropriate for objects of our application including users and non-repudiable credentials. In oBIX, every objects are referenced using a unique URI and accessed with the read or write operations. More complex commands than a read or write could use the invoke operation. The doors in our prototype implementation have URI for reference and each door has two sub-objects called unlock and open. The unlock sub-object tells whether the door is unlocked and the open sub-object shows whether the door is opened. When clients send a request to open a door, the application server sends write operation to the unlock sub-object of the door object. The open sub-object is read only since our application does not require the write operation on it. However, it is possible to design the system in a different way since these are simply a design decision we have made. In addition to the above objects, we also have an area object that is associated with door objects. An area can have more than one door assigned to it. In this way, we can have an access control list for each area instead of keeping it for every door. It is more convenient if an area has more than one door. Therefore, every request to unlock a door is checked against the access control list of the area associated with

the door. These area objects only allow the read operation since the application does not need to modify the access control list.

Using the oBIX XML language described above, the application server sends requests from users to the gateway. It also passes the non-repudiable credential which is a signed command of what the user has requested. It uses a dedicated line and UDP connection to communicate with the gateway. It also utilizes the Web Service Security [20] to secure the communication.

5.3 Gateway

The gateway is responsible for performing access control on requests from the application server. It is also written in C#. When a request comes from the application server, it first checks the integrity of the command by verifying the signature. We assume that the gateway already has the public key of the client for the verification. We discussed possible ways to acquire the public keys in Section 5.1. Unlike the other two components, we consider the gateway as a trusted component. Therefore, we assume that the gateway will perform the verification honestly. It is also the last line of defense against attacks to the building control system. If it cannot verify the signature, then it simply rejects the request by sending an error message to the application server which will pass that to the client program. Once the signature is verified, the gateway retrieves information from the request. It includes the user ID, the door name, and the authentication time. The authentication time is used to find out how fresh the request is. The gateway proceeds only when the request is made less than the configured expiration time. This expiration duration could be configured to allow longer or shorter window. Currently, it is set to five minutes for our prototype implementation. Next, the user ID is checked against the gateway policy. We implemented a white list and a black list for the user ID. The user ID must be in the white list and must not be in the black list. After the check, the gateway looks up the area that the requested door is assigned to and determines whether the user has the access privilege on that area. If so then the gateway

sends the request to the BAS.

The gateway communicates with the BAS in two ways. First, it accesses the BAS database to check the access privileges of users and to record a log of recent requests. Standard SQL is used by the gateway to communicate with the BAS database. Second, the gateway interfaces with the Andover Continuum NetControllers to operate the BAS. It uses an OPC [23] interface to communicate with the NetController. The gateway can control building objects and read current states of those objects with the OPC interface. In our implementation, the gateway sends the unlock command to doors and reads the current state of doors using the OPC interface. Here, the current state of a door represents whether the door is opened or closed.

In summary, the gateway verifies the signature, checks the authentication time, examine the policy constraints, and refers the BAS database for access rights. If the request proceeds without an error, the gateway finally issues the unlock command to the associated NetController to unlock the door. Then it reads the state of the door and records relevant information on the BAS database. Lastly, it sends a success message to the application server which will forward it to the client program. The user is now able to open the door.

Chapter 6

Analysis

Here, we discuss a risk analysis of our system in providing interconnectivity to control systems. In particular, we look into the different components of the architecture for possible risks and consequences. In addition, we compare our strategy with other possible alternatives that try to mitigate damage from malicious components or outside attacks.

6.1 Risk Analysis

Our system applies the multi-tier system to separate functionalities in providing the interconnectivity to the control system. Therefore, we introduce three new components to the control system in providing the interconnectivity and supporting useful applications. Each of the components could be a target for attack as well as the complex communication links between them. However, because the multi-tier system compartmentalizes each component, it could effectively contain faults or attacks to the targeted component. Additionally, the multi-tier system makes the security-critical components to be partitioned easily.

The client program is usually as vulnerable as the machine that runs the program. If the machine gets compromised, there is a high chance that the client program could also be compromised. Computers are compromised in various ways. For example, people can accidentally download viruses or malware on their machines or they could open and execute an infected file. Attackers could exploit bugs of operating systems or applications to take the control of the computers as well. Even though we expect that the users will have standard precautions for viruses and malware, we do not rely on the users to keep their computers safe. Consequently, we consider the client program not trustable. Our system limits the

effect of the compromised client program by enforcing the principle of least privilege and the gateway policy. The compromised client can only perform actions that are already allowed to the client in the control system. Moreover, the gateway policy can enforce additional policy that does not allow access to critical functions using our system although the user actually has the right. Once the gateway finds out that the client program or the machine is compromised, it should revoke the key it of the client. It can do it simply by deleting the key and including the user ID in the black list.

The application server is highly vulnerable as well since it accepts connections from various clients, presenting a large attack surface. Attacker can install the client program and try to compromise the application server by exploiting the client program. As the application server supports more applications, it will have more complex and larger code base for those applications. This might provide a source for bugs and vulnerabilities that could be used by attackers. Again, we expect that the application server will have general protection mechanisms like firewalls, but some attacks like zero-day exploits are hard to defend. Therefore, we do not trust the application server and limits the impact of the compromised application server with the non-repudiable credential mechanism. Since the compromised application server cannot acquire appropriate keys to sign fake requests, all it can do is to use the signed requests that are within the expiration duration. As long as the gateway proxy is secure, the actions of the compromised application server are very limited.

We assume that the gateway proxy is a trusted component. The reason for this is because the gateway exports a limited, static interface only to the application server. We believe that there is a rare chance of vulnerability in that narrow interface. However, it is highly dangerous to the control system if it gets compromised because the gateway is the last line of defense to the control system. If it gets compromised, accesses from outside to the control system are not properly managed and could allow full access to attackers. Therefore, it is important to thoroughly examine the interface for any bugs and vulnerabilities. Every possible protective mechanism must be in place as well.

6.2 Possible Alternative Approaches

Sandboxing is a common way to monitor and limit access of a component. It provides limited interface to an untrusted component or program such that the untrusted component or program can only access predefined safe functions or data. It is designed to enforce the principle of least privilege on the sandboxed component. Extensive use of sandboxing appears in limiting system call interface to untrusted programs [4, 15, 14, 25] and in execution environments like Java. The idea of sandboxing can also be applied to our system to enforce the principle of least privilege on the application server. This can be done by providing limited interface to the application server. This approach can limit the possible actions of the compromised application server but not as much as our system. For example, the compromised application server with sandboxing can send requests to open arbitrary doors, whereas our system cannot do that due to the non-repudiable credential.

Another possibility is privilege separation [26] where an application is separated into two components: a privileged part and an unprivileged part. Because of the separation, attacks and damages on the unprivileged part is contained to itself, leaving the privileged part intact. However, there are several disadvantages of this approach compared to our system. First, every application must be separated into two parts, either manually or automatically [9]. As the number of application increases, more work needs to be done. Our system, on the other hand, utilizes the multi-tier architecture to provide reusable interfaces and components, making it easier to add new applications. Second, the separation must be conducted wisely requiring knowledge about the application and some efforts. Simply placing entire application in the privileged part will not give much security benefit. Finally, the separation might introduce more complexity to the application, leaving more attack surface. Sometimes it is not clear how to separate an application into the privileged and the unprivileged part. If the application requires a lot of complex communications between various components, and if they are separated into different parts, it is not simple to provide a secure communication between the two parts. If the communication becomes complex, there is a higher chance for

a bug which could be used by attackers.

Intrusion detection system could also be used to protect the control system from outside connections. It looks for anomalies in requests or suspicious access patterns and sends alerts to the management system. For instance, it can consider many rejected requests from a single client as suspicious and send an alert to the operator of the control system. However, the intrusion detection system alone is not sufficient to provide enough protection to the control system. It is weak to zero-day attacks and is very hard to enforce policy constraints like our system. For the intrusion detection system to enforce the principle of least privilege, it must be able to figure out who has sent a request and decide whether that user has appropriate access privilege. But we believe that the intrusion detection system can be incorporated into our system to provide better protection. Audit records from the gateway could be used by the intrusion detection system to find patterns and anomalies.

Chapter 7

Conclusion

In this thesis, we introduced a new system which allows interconnectivity to the controls systems from outside networks with a method to effectively mitigate risks to the control system. In particular, we applied a multi-tier system to the control system and a way to enforce the principle of least privilege on a higher-tier. In addition, our system requires the non-repudiable credential with every access to the control system. These protective mechanisms effectively restrict possible actions of malicious or compromised components on the control system. Our system can also support a security policy separate from the existing access control policy of the control system. With the secure interconnectivity provided from our system, management of the control system becomes more convenient and efficient. Moreover, the control system can now support various useful applications to help users of the control system.

We have validated our system with an implementation on the building automation system test bed. We presented a door unlock application which allows users to unlock doors without physical keys or cards. Our system consists of three components: the client program, the application server, and the gateway proxy. The client program provides a user interface to send unlock requests to the control system. It utilizes digital signature scheme to provide the non-repudiable credential to the gateway proxy. The application server acts as a web server, accepting connections from clients and passing the signed requests to the gateway proxy. The gateway proxy is a trusted component which enforces the principle of least privilege on the application server as well as its own security policy. It checks various constraints including signature verification and access rights. It directly communicates with the BAS.

We believe that our system could be applied to various control systems and support many useful applications.

References

[1] Final report on the august 14, 2003 blackout in the united states and canada: Causes and recommendations. Technical report, U.S.-Canada Power System Outage Task Force, April 2004.

[2] Standard 135-2004. Bacnet, a data communication protocol for building automation and control networks, 2004.

[3] EIA/CEA 709.1-B-2002. CONTROL NETWORK PROTOCOL SPECIFICATION, 2002.

[4] Anurag Acharya and Mandar Raje. Mapbox: using parameterized behavior classes to confine untrusted applications. In *Proceedings of the 9th conference on USENIX Security Symposium - Volume 9*, pages 1–1, Berkeley, CA, USA, 2000. USENIX Association.

[5] American Gas Association. Cryptographic protection of SCADA communications. AGA Report No.12, 2006.

[6] Lujo Bauer, Scott Garriss, Jonathan M. Mccune, Michael K. Reiter, Jason Rouse, and Peter Rutenbar. Device-enabled authorization in the grey system. In *In Proceedings of the 8th Information Security Conference (ISC'05*, pages 431–445. Springer Verlag LNCS, 2005.

[7] Lujo Bauer, Scott Garriss, Jonathan M. Mccune, Michael K. Reiter, Jason Rouse, and Peter Rutenbar. Device-enabled authorization in the grey system. Technical Report CMU-CS-05-111, Carnegie Mellon University, Computer Science Department, February 2005.

[8] Jodie P. Boyer, Ragib Hasan, Lars E. Olson, Nikita Borisov, Carl A. Gunter, and David Raila. Improving multi-tier security using redundant authentication. In *ACM Computer Security Architectures Workshop (CSAW '07)*, Fairfax, VA, November 2007.

[9] David Brumley and Dawn Xiaodong Song. Privtrans: Automatically partitioning programs for privilege separation. In *USENIX Security Symposium*, pages 57–72, 2004.

[10] Echelon Corporation. LonTalk Protocol Specification, 1994.

[11] CPNI and Homeland Security. Configuring and Managing Remote Access for Industrial Control Systems, November 2010.

[12] Paul Ehrlich and Toby 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`.

[13] L. Fazal, S. Ganu, M. Kappes, A.S. Krishnakumar, and P. Krishnan. Tackling security vulnerabilities in vpn-based wireless deployments. In *Communications, 2004 IEEE International Conference on*, volume 1, pages 100 – 104 Vol.1, june 2004.

[14] Tal Garfinkel, Ben Pfaff, and Mendel Rosenblum. Ostia: A delegating architecture for secure system call interposition. In *IN NDSS*, 2003.

[15] Ian Goldberg, David Wagner, Randi Thomas, and Eric A. Brewer. A secure environment for untrusted helper applications: Confining the wily hacker. In *IN PROCEEDINGS OF THE 6TH USENIX SECURITY SYMPOSIUM*, 1996.

[16] Carl A. Gunter, Sanjeev Khanna, Kaijun Tan, and Santosh S. Venkatesh. Dos protection for reliably authenticated broadcast. In *NDSS*, 2004.

[17] Srikanth Kandula, Dina Katabi, Matthias Jacob, and Arthur W. Berger. Botz-4-Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds. In *2nd Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005.

[18] Byeong-Ho Kang and Maricel O Balitanas. Vulnerabilities of vpn using ipsec and defensive measures. *Science And Technology*, 8(9):9–18, 2009.

[19] Sherif M. Khattab, Sameh Gobriel, Rami G. Melhem, and Daniel Mosse. Live baiting for service-level dos attackers. In *IEEE INFOCOM*, pages 171–175, 2008.

[20] Kelvin Lawrence and Chris 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`.

[21] David Mankins, Rajesh Krishnan, Ceilyn Boyd, John Zao, and Michael Frentz. Mitigating distributed denial of service attacks with dynamic resource pricing. In *ACSAC*, pages 411–421, 2001.

[22] William G. Morein, Angelos Stavrou, Debra L. Cook, Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. Using graphic turing tests to counter automated ddos attacks against web servers. In *ACM Conference on Computer and Communications Security*, pages 8–19, 2003.

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

[24] Bryan Parno, Dan Wendlandt, Elaine Shi, Adrian Perrig, Bruce M. Maggs, and Yih-Chun Hu. Portcullis: protecting connection setup from denial-of-capability attacks. In *SIGCOMM*, pages 289–300, 2007.

[25] Niels Provos. Improving host security with system call policies. In *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12*, pages 18–18, Berkeley, CA, USA, 2003. USENIX Association.

[26] Niels Provos, Markus Friedl, and Peter Honeyman. Preventing privilege escalation. In *USENIX Security Symposium*, 2003.

[27] Supranamaya Ranjan, Ram Swaminathan, Mustafa Uysal, and Edward W. Knightly. Ddos-resilient scheduling to counter application layer attacks under imperfect detection. In *INFOCOM*, 2006.

[28] Mudhakar Srivatsa, Arun Iyengar, Jian Yin, and Ling Liu. A middleware system for protecting against application level denial of service attacks. In *In Middleware*, pages 260–280, 2006.

[29] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. DDoS Defense by Offense. In *ACM SIGCOMM 2006*, Pisa, Italy, September 2006.

[30] XiaoFeng Wang and Michael K. Reiter. Defending against denial-of-service attacks with puzzle auction. In *IEEE Symposium on Security and Privacy*, pages 78–92, 2003.