2011

An Analysis of Remote Biometric Authentication with Windows

Brandy Marie Eyers

University of South Florida, BrandyEyers@gmail.com

Follow this and additional works at: https://scholarcommons.usf.edu/etd

Part of the American Studies Commons, and the Computer Sciences Commons

Scholar Commons Citation


This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
An Analysis of Remote Biometric Authentication with Windows

by

Brandy Marie Eyers

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Jay Ligatti, Ph.D.
Adriana Iamnitchi, Ph.D.
Lawrence Hall, Ph.D.

Date of Approval:
March 23, 2011

Keywords: Multi-Factor, Computer, Active Directory, Kerberos, Programming, Security

Copyright © 2011, Brandy Marie Eyers
ACKNOWLEDGEMENTS

I would like to thank Charles B Bradley II for all his help on getting this project started and his continued help throughout it. I would also like to thank Glenn R Kaufman for all his help with the technical side of things, such as getting the computers and virtual machines up and running. This work was part of the USF-Raytheon project “Avatar DNA using Biometrics and User Access Controls.” It was funded in part by Raytheon Company with matching funds from the Florida High Tech Corridor program.
# TABLE OF CONTENTS

**LIST OF FIGURES** ii

**ABSTRACT** iii

**CHAPTER 1 INTRODUCTION** 1

1.1 Related Work 4

1.2 Outline 7

**CHAPTER 2 INITIAL DESIGN** 8

2.1 Introduction 8

2.2 Enrollment 10

2.2.1 The Application 11

2.2.2 Storage 12

2.3 Credential Provider 14

2.4 Kerberos Interactive Unlock Logon 15

2.5 Proxy 16

2.6 Summary 16

**CHAPTER 3 CHALLENGES WITH THE INITIAL DESIGN** 18

3.1 Introduction 18

3.2 Enrollment/Credential Provider 19

3.3 Extending the KIUL 20

3.4 The Proxy 21

3.5 Summary 21

**CHAPTER 4 WINDOWS BIOMETRIC FRAMEWORK** 23

4.1 Introduction 23

4.2 What WBF Does 25

4.3 Initial Design Vs WBF 27

4.4 How WBF Works 28

4.5 Issues 30

4.6 Summary 31

**CHAPTER 5 CONCLUSION** 33

**LIST OF REFERENCES** 36
LIST OF FIGURES

Figure 2.1  This shows you an example of storing a fingerprint template in a partition. 9

Figure 2.2  The early enrollment application opened up this Upek SDK, which was what took the fingerprint template. 12

Figure 4.1  This is an outline of how Microsoft has structured WBF. 28
ABSTRACT

One thing that everyone seems to be worried about when it comes to his or her computer is security. If your computer is not secure then private information could be stolen. Many people now use passwords to protect themselves though they are discovering that using multi-factor authentication is much more secure. It allows you to use multiple different proofs of who you are. Biometrics is one of the ways to prove identity. Using it, you could log into a system with just a fingerprint, which is something that is very difficult to steal. We present a suite of software tools that allows you to log into a network using multi-factor authentication. This thesis describes our design of a multi-factor authentication solution, the problems we encountered realizing this design, and Microsofts own biometric system.
CHAPTER 1

INTRODUCTION

There are many forms of authentication in this day and age; you can prove who you are in many ways. There are three main types of authentication: something you know, something you have, and something you are. An example of these forms is: you know a password, you can have a smart card, and you can use your own fingerprint. Most systems only need one form of verification to allow you to logon and that is usually just a username with a password. It has become increasingly popular for the other two forms to be used and some industries are looking into using multiple ways to verify yourself. This is known as multi-factor authentication. As you add on more ways to authenticate the security gets stronger. Some of these ways to gain access are more secure than others. You could figure out someone’s password or possibly steal a smart card, yet when it comes to biometrics it gets difficult to impersonate someone else. You can’t easily steal someone’s fingerprint. It is possible, but that would require fingers being cut off or capturing the data as it’s sent to be verified and at that point you know that someone is after the account. In 2005 this actually happened when some thieves cut off a mans index finger so they could start and steal his Mercedes Benz [1].

Many new computers now come with biometric readers allowing you to logon to it with just a swipe of your finger. More and more large companies are now asking if it is possible for their users to logon with more then just a password. For most big companies, when you logon to your workstation at the office it requires going in a remote repository to check who you are instead of storing the data locally
on the computer, as done on personal computers. The question arises if you can store more than just your username and password to a remote repository. We found that RFID values can already be stored in a repository. Can you store your smart card value? What about biometric information? When we first started this project all applications on the market still stored the templates for fingerprints on the local machine or the actual reader (if it plugs in via USB). If this problem can be addressed think about what you could then do with it. Greater security can be achieved within large companies and not just to a single workstation because it would now require a fingerprint to get onto the network. This means that someone could not hack just any user’s account because a fingerprint would now be required no matter where they were logging on from.

In this thesis we will take a look at our original solution to the biometric problem and how we hoped to integrate it with Active Directory, a remote repository which can control a network’s members, servers, workstations, and everyone in the company that accesses those resources. Many questions and problems arise when looking at this. In short this system can store a user’s fingerprint (and eventually other biometric data) and then use it to log them in over a network connection using various security measures.

First an enrollment would be needed to register biometric data to the system but we stumbled upon an issue. We found there were two different ways to store the user’s information in Active Directory. Out of the two which would be the most secure? One was a simple attribute which would be stored with the rest of a user’s information. The second choice was a completely separate partition of the database from which the enrollment application could transfer the biometric data. We will also discuss how Active Directory is set up as well as our implementation of the enrollment.

Next is the actual login over the network. There are two different functions that we must adjust so that the system can authenticate properly. We have to first extend the
Credential Provider, which is the login screen you see when you start up a computer, to allow the capturing of a fingerprint. We were glad to find that the fingerprint device we were using came with an application that could capture templates. An authentication blob called Kerberos Interactive Unlock Logon, KIUL, gets sent out by the Credential Provider once a user wants to submit their password. This KIUL must be extended to allow the fingerprint template to be taken with for authentication. Second, we have to intercept the KIUL by a proxy before it reaches the Local Security Authority, LSA, which it cannot handle the biometric matching. Our proxy will be able to do this 1-to-1 matching and if the templates match then the password will be sent to the LSA for verification. The results would get sent back to the Credential Provider and allow the user on to the workstation or deny them.

We had high hopes for this solution to work, but as we ran into problems we decided to discuss them with Microsoft. We found that the SDK from Upek, the maker of the fingerprint device we were using, did not make the template to enable its use as we proposed and therefore, we would have to rewrite the Enrollment [2]. These templates also were too large to be stored properly and Active Directory could not accept that file type for storage. As the discussions with Microsoft went on we found that the KIUL was unable to be extended. We also found out that the Proxy would not be able to handle the template matching as well.

There was good news from all of this though. Microsoft told us that their newest operating system, Windows 7 and Server 2008 R2, would be able to accomplish something similar to what we had hoped to do. The Windows Biometric Framework, WBF, was the answer. It would be able to make templates and store them in a database. The framework would also be able to retrieve a template and match it against a new one and do verification the way we envisioned. It was now possible to login to a computer over a network with just a fingerprint.
Perhaps one of the best ideas driving WBF was that almost all fingerprint devices would work with it. Companies would have to program a new driver to work with it but this allowed devices to be interchangeable. This also meant that separate SDKs and programs from these device companies were no longer needed. Microsoft put out their own support for applications to be used with this framework. Although they did not release any finished applications, programmers can now create applications that can do anything from enroll a template to verify a user.

It was evident that a lot of work went into WBF. As we will see it could handle many of the ideas we had about our own application but Microsoft was able to take it to another level. They did not have to use existing hardware or software because they had the resources to implement their own. There are three core pieces to the framework: storage, retrieval, and matching templates. Microsoft also developed an API that allowed for the creation of custom applications that would fulfill anything a company may want to use WBF for. This API also allowed administrators a way to maintain a system of this size.

We did find there were things that may have been left out or could have been improved when compared to the ideas for our own system. The overall idea was the same; to allow an extra layer of security when it came to authentication. We had hoped to take it a step further and allow for a more multi-factor approach. Microsoft does indicate that WBF will be able to handle other types of biometrics at a later date. Other types of authentication, like RFID cards, could be added as well. It may also be of interest to Microsoft to precode applications that many large companies may be able to use instead of hoping they will just program their own.

1.1 Related Work

There is quite a lot of information available on the idea of authentication and on the multi-factor level as well. Also there are already applications that can log
someone onto a computer yet they aren’t exactly the same as what was proposed or found here. We were also able to find a lot of resources on how all the database pieces worked.

While researching we found quite a few applications already implemented that intended to do a similar thing. One was the BioCert Intelligent Identity Manager. Their program can do a single authentication or even a multi-factor one. What differs is that even though they support many different types of verification, you can only choose one [3]. The application that we were working on would have eventually been able to plug in different authentication types and use any number of them at once. Most of the others that were found were similar to this. The other three applications, Biometric Network Logon 2007 R2 (by Griaule Biometrics), Biometric Computer Logon (by Bayometric Inc.), and IDenium (BioLink), all can only handle biometrics and/or a password and username [4, 5, 6]. What separates all of these existing applications is how they handle the biometrics. Their sites state they work with Active Directory yet only the username and password were actually stored there. The biometrics is all handled locally on the machine or the biometric device and not in the Active Directory like ours. Sadly none of these implementations have any formal papers or write-ups.

Though there is not much literature on using biometrics over a network or proxies, there are many ideas about multi-factor authentication that includes biometrics. One of these papers is “Comparing Passwords, Tokens, and Biometrics for User Authentication” by O’Gorman [7]. This paper is very helpful because it has many definitions of things that have to do with authentication. It shows that we don’t need to mainly rely only passwords. The three types of security: Knowledge-based, Object-based, and ID-based are all discussed and examples are included. The authors also go into the detail of pros and cons and then compare the types. There is also a section on attacks.
Another paper, “Privacy Preserving Multi-Factor Authentication with Biometrics,” by Bertino et al, paints a great picture of how just security works with biometrics [8]. It includes how the fingerprint template is obtained and stored and an outline of the entire system. The authors propose a different system from what is normally used, which is comprised of two steps. The first takes a fingerprint and uses an algorithm on it (which takes four steps), creating what they call a key. The second is the use of a Zero Knowledge Proof to do the actual authentication against the key made earlier.

The next paper, “Ensuring Privacy of Biometric Factors in Multi-Factor Authentication Systems” by Argles et al, is similar to the last as it primarily looks at biometrics in a multi-factor environment [9]. It differs from the last because it focuses on how a program can keep biometric information secret during the authentication process. One thing that had to be kept in mind for this paper is that it is dealing with just a fingerprint and not a template (which is more secure to use since its not the actual print). The approach first proposes to keep the fingerprint in a separate place (not local, like on a thumb drive). The paper then proposes a few functions like hashing, which could be used on the print before storing.

The last of the related literature contains many of the resources that hold the key to understanding how the system worked in the end. One of the first pieces that we had to understand was how Kerberos worked. Kerberos is a “network authentication protocol” and was created by MIT and was integrated into Windows security [10]. Technet had a great quick overview on the normal setup. Perhaps the most helpful source was the MIT website on Kerberos, which has all kinds of information and a list of more papers. A user asks for a ticket that allows them to get another ticket; they then acquire that ticket and must decrypt it. Once that one is decrypted they use it to request what is called a Service Ticket, which allows them to get authenticated by
the server. Kerberos is what is sent to the Active Directory with the users information that is to be authenticated [11].

The next item that needed to be understood was the setup of Active Directory. Both Technet and msdn have a good amount of information on all of its inner workings [12, 13]. In short, the Schema is like a forest, full of trees or classes. Classes can be comprised of many branches (or attributes). The other part of Active Directory that is important is the Lightweight Directory Services (LDS). The LDS is basically a separate partition that is used by applications, such as ours. It is set up similarly to Active Directory, like a forest. The last concept from Microsoft that we had to understand was that the LSA is how a user can log on to a single workstation or one over a network.

One final piece of information that needed to be researched was how Upek handled storing their fingerprints. It’s done with a template and normally stored straight onto their reader. This template is stored in another structure called Passport, which is binary data [14, 2].

1.2 Outline

This thesis is organized as follows: Chapter 2 discusses the application we started to implement; the enrollment to the proxy. Chapter 3 looks at all of the issues we had with our application. Chapter 4 discusses Microsoft’s Windows Biometric Framework, which is their own implementation. In Chapter 5 we conclude with our findings and what we discovered about Microsoft’s own framework.
CHAPTER 2
INITIAL DESIGN

2.1 Introduction

In this chapter, we discuss the initial solution we came up with to the biometric problem that we discussed in the previous chapter; this was based purely on ideas for Windows Vista. We originally planned to do this by integrating biometrics with Active Directory, a Lightweight Directory Access Protocol (LDAP) repository. Active Directory is part of the Windows Server Domain Controller functionalities, which is what handles the security between users and the domain. Active Directory is able to manage members, servers, workstations and other functions of a company’s network. We proposed a system that could store a user’s fingerprint and then use it to log them in over a network connection through various security measures. Figure 2.1 presents an overview of the system we will propose.

The first part to be handled is the actual enrollment of a user’s biometric data. Here is where we came across the first problem. Within Active Directory we could store this information in two ways. Though both would have worked we had to figure out which of the ways was the best; the most secure. The first way is by storing the fingerprint in an attribute within the user’s class of information. The second is in a separate partition of the database. Figure 2.1 shows an example of the second storage type where the enrollment application can send the information to the partition where it will be stored. The entire implementation of the enrollment and how Active Directory is set up will also be discussed.
Figure 2.1. This shows you an example of storing a fingerprint template in a partition. The partition sits in the Domain Controller where both the enrollment application and the credential provider can access it. The proxy is also located here and has access to the partition as well. [15]

Now that we have the fingerprint stored we have to figure out how to login over the network. Two things need to be done here in order to authenticate properly. First we have to change the login to the machine. This is known as GINA or the Credential Provider (the name was changed in Vista, Windows 7, Server 2008, and beyond). The Credential Provider needs to be extended to allow the capturing of a fingerprint. Luckily, it is easy to get a fingerprint reader’s SDK (software development kit) working with it. The Credential Provider sends out an authentication blob called a KIUL to the LSA. We must figure out how to extend it to also carry the biometric information. This KIUL would also be used to eventually send other types of biometric data.
Lastly we have to now intercept this KIUL with a custom module or proxy, since LSA can’t handle the biometric comparison. This proxy will do the 1-to-1 matching using the fingerprint reader’s template. If the information checks out it will then send off the password to the LSA for the final piece of verification. This entire process will then return the results, whether the user passed the verification or not. As shown in Figure 2.1, the proxy will be able to access the partition itself, prior to forwarding the login information to the LSA.

2.2 Enrollment

To understand how the enrollment process will work, you have to know how Active Directory is set up. Active Directory is Microsoft’s solution to managing a computer network. It is really an LDAP repository that has a specific purpose; it provides Domain Services to a network. A large company could use it to control its employee’s access to certain computers. It also contains many pieces of information about each user. The Active Directory is split into three parts: the domain, which contains objects like users and computers; the schema, which contains the classes and their attributes; and the configuration, which holds information on services and other partitions [16]. The main part we needed to focus on was the schema part of Active Directory. The schema is where the definition for an object can be found, where it is stored, what it is connected to, and what type it is. It itself is broken up into different parts and the two we want to concentrate on are classes and attributes. Classes are comprised of grouped attributes and any related information. An example of a class is the User Class which is where all of the attributes of a user are stored, like their name, password, address, etc. These attributes can have a single type or syntax as they call it [17].
2.2.1 The Application

Part of the actual enrollment is quite simple. The User class already exists in the Active Directory along with many attributes where you could store information about someone. When in the User class, it is easy to create a new user; all you need to put in at first is a username and a password. An administrator can easily add new accounts as needed. All our enrollment requires is that the user we wish to register is already in the system.

Now all we need to do is enroll this user’s fingerprint so they can later log into the system using it. The user will have to physically be present for this part. It is all done on the computer where the server’s Active Directory is located. This enrollment can also be done on a workstation that is setup to handle this process; where the Registration Authority has the application and Active Directory privileges. Our implementation was a C# graphical user interface that reads in a username and password, Figure 2.2 shows an example of the GUI. The password is needed because even though the administrator or Registration Authority has access to the Active Directory directly, you wouldn’t want them to be able to store their own fingerprint. The application works by first having the user click the ‘Grab’ button which will open up another window (which is the Upek SDK in our case) and prompt to get a fingerprint. Once the user is satisfied with how their print looks, they would enter their username and password and hit submit in the application screen. The application will grab the template from the Upek part and send it using .Net code. This first makes sure the username and password are correct and will then save the fingerprint template in the Active Directory. There would also be a third button, Verify, which would allow the user to input another finger swipe of the same finger that would then retrieve and match against the one that is now on the server. This verification would be the last step, and would insure that the stored print is a good copy and can allow for a correct match.
The early enrollment application opened up this Upek SDK, which was what took the fingerprint template. The application would verify that the username and password are correct and then store the template in the partition.

After reading further into how the Upek reader works we ran into some problems. This Grab program only takes a snapshot of the fingerprint; an image. It doesn’t make the binary template that we want to store to Active Directory. Luckily in the Upek class they have functions that can create these templates. If this design was to work, the other class from Upek’s SDK would be changed out with Grab to properly get this template and would work similarly to how we initially planned.

2.2.2 Storage

Now that we have the fingerprint we need to store the template in Active Directory. The reader we were using is from Upek and doesn’t store the image of a fingerprint, which would be very unsafe. Instead, it stores a template that is made by running the image through an algorithm. After talking with Upek directly they were able to tell us that this template information is binary [14]. We read through Active
Directory’s information and found that there are three attribute syntaxes that are of the binary type: String(Octet), String(Sid), and Object(DN-Binary). The last one can be eliminated since we do not need a distinguished name (the DN part). The other two are actually the same type, so we went with String(Octet) since the String(Sid) seemed to be named for a special case. We now have a syntax type and must find a place to store it. After some investigation we determines that we had two options.

The first type of storage is what Active Directory calls an attribute. In short all we would need to do it create an attribute of the String(Octet) syntax and then add it to the user class. This would be the simplest way to go but there is a problem with doing it this way. There isn’t any security at all for doing it this way because it is just another variable. It would be easy for any person that had access to the Active Directory to simply store anything they wanted, like their own fingerprint in that variable.

The other way is using Active Directory Lightweight Directory Services (AD LDS), or Active Directory Application Mode (ADAM) as it was formerly called in Server 2003. It is a LDAP like Active Directory but does not need domains and you can run more than one separately. These are separate partitions that are used for applications that don’t want to deal with using the Active Directory but still need to store things to the server. This would allow us to create one of these partitions for the entire application that could store the fingerprint, other biometrics, RFID, and others. The LDS can work with Active Directory as well. The user’s information can still be kept in the main user class. The Active Directory would know that the user class had additional information in a LDS because there would still be attributes in the Active Directory as well as in the LDS. When it would get a request for those attributes, it would send it to the LDS to be handled [18]. What is great about this option is that only the allowed applications can get to the data that is stored here.
Either way would work, it’s a matter of deciding which is the best one to use in the end. Before we came to a stop with this, we were testing both ways to see which would work the best. We knew that going the partition route would have been the best because only this application and the proxy can access it. The attribute would still be tested to see what kind of security it had as well.

2.3 Credential Provider

The next step is to setup the Credential Provider which was previously known as Graphical Identification and Authentication (GINA). Microsoft changed the name from GINA to Credential Providers with Windows Vista. This is the screen that you see when you turn on your computer where you login. The normal set up asks for a username, or to select one, and then for the password. Most machines are setup to do the authentication locally from the machine. Here, is where we needed to add in the other factors to authenticate a user, such as our fingerprint reader.

We had to implement our own custom Credential Provider. The Upek reader we were using came with both the drivers and the SDK which were downloaded to the machine. Microsoft has a few sample Credential Providers online that we downloaded and tested. To get the reader working, we were able to copy the code that would grab the image from the reader and then integrated the functions into the Credential Provider. In short this is similar to when we grabbed the template to store data in the Active Directory. Doing this would make the reader’s own grab application open up along with the prompt for a user’s password and username. This grab code would have later been changed out to use the SDK that created the template and not just show the image. Now that we had it all coded it needed to be loaded properly. The Credential Provider needed to be registered within the workstation and then the dll needs to be copied to windows/system32 folder. The dll and graphics for the reader also had to be moved there. Once this is done and after the machine is
turned off and then back on, this new version of the login will be the one that you can see. The Credential Provider is what sends the user’s information (username, password, biometrics) to Active Directory by KIUL which will be handled in the following section.

2.4 Kerberos Interactive Unlock Logon

Now that we have the user’s information from the Credential Provider we now must send it to be authenticated. Within the Credential Provider there is an authentication structure that gathers the data and sends it off. The one that is normally used is called the KERB_INTERACTIVE_UNLOCK_LOGON, or KIUL for short here. This can unlock a workstation by sending the username, password, and the name of the domain on which it can do the authentication. There are a few other existing structures including one for a smart card. We needed to extend the KIUL so that it could handle sending the biometric template along to the next step. None of the existing structures can directly handle the binary data so we had to extend the KIUL.

First what needs to be done is that the Credential Provider needs to call LsaLogonUser which contains many important data elements including what is called the Authentication Package. The package will be explained in the next section but it basically tells which package we need to use to authenticate later. This also requires us to specify which Authentication Information to use; this is what we currently need to figure out. This Authentication Information is a pointer to a buffer that contains the data we need to use to login. The format was specified in the Authentication Package [19]. For our biometric solution we would want to call a custom Authentication Package that we will discuss later. Our KIUL would be able to grab the binary template along with the rest of the normal authentication data and send it over to the Active Directory.
2.5 Proxy

The very last thing we now need is the proxy that will handle the authentication part. The proxy will be server side like the enrollment. This is where we intercept the KIUL and perform a few tasks. First we handle the 1-to-1 matching of the biometrics or whatever other data we decide to pass along. This proxy will go into our partition and grab the template from there to do the matching with the one we got from the KIUL and will use the reader’s algorithm to authenticate. It will then pass on the username and password to the LSA to make sure that those are also correct. Finally it will send back the answer as to whether the user trying to login can do so or not.

In our last section we mentioned something called an Authentication Package. This proxy is basically a custom Authentication Package.

Our proxy will “sit in front” of the normal LSA and intercept these messages coming in. We do this by actually having our proxy registered instead of the normal LSA and then our proxy will take care of loading and calling the LSA when needed. The LSA will appear as though it does not exist and the server and the process will have to use our custom proxy instead by default.

2.6 Summary

It is clear that multi-factor authentication is the best route to go when you want to secure information, though current systems seem to lack it or such authentication is limited to the computer and not allowed over a network. This implementation would allow all data to be stored in the Active Directory allowing it to be secure. What was great about Active Directory was that it contained all the information for a user as well as the setup of the network. Active Directory was capable of storing fingerprints as well as other biometric data, which would allow someone to login over a network.
The enrollment process would have been done on the local machine using a fingerprint reader that would allow for custom applications to be created. This application would create a template and store it on the Active Directory. We did come across a problem here. This information would either be stored as a simple variable or in a separate partition for the application. Out of the two options the second would prove to be more secure since the only thing allowed to access the information would be the enrollment.

Since this takes place over a network, the machine that a user would login from would need to be able to authenticate the user properly. For this to happen the machine’s Credential Provider would have to be changed. The same reader brand from the enrollment would have to be used again, so we would be able to make application’s using that company's SDK. Now that we have the template of the fingerprint, we need to get it back to Active Directory. Here is where the KIUL would take over. This extended structure would take the template along to the LSA, but since the LSA can’t handle this information a custom module would intercept it and take care of the template. Using the 1-to-1 matching process from the reader’s SDK, it would match this new template against the one stored in the partition. If this matches, the module sends the username and password to the LSA to be verified. Once the LSA gets this information it will tell the Credential Provider if the user can log in or not.
CHAPTER 3

CHALLENGES WITH THE INITIAL DESIGN

3.1 Introduction

We know that passwords are a good way to protect something. If you add in using biometrics like we have been discussing then access becomes more secure. It is clear that multi-factor authentication is the best option when you want to securely store or retrieve information. Unfortunately current systems seem to lack it or such authentication is limited to the computer and not over a network. Originally, we thought that the initial design was going to be the way to go. It would have added in that extra piece of authentication. That extra bit of information would have made it more secure because of how difficult it is to steal a fingerprint.

Near the end of the project we ran into a few problems. For example Microsoft discussed the idea of a custom proxy on their website, but had no information on how to actually write it. So we went ahead and contacted Microsoft. We had some challenges and had to perform some work arounds up until this point, but it wasn’t until we contacted Microsoft that the project came to a halt. That and the accumulation of other issues made us take a step back to re-evaluate our design. First we had issues with the enrollment and the credential provider. We had started to use Upek’s SDK, which could capture a fingerprint. It was not until later when we looked into the code and realized that Upek’s SDK could not make the template that we thought it would make. There was also a problem with sending the template and storing it. One of the largest problems was with the extension of the KIUL. This is what we had discussed
with Microsoft. In short they told us it was impossible to actually extend it. Lastly, we had problems with the proxy. Here Microsoft again told us this was extremely difficult to do, and since we couldn’t change the KIUL it was useless. There were also several other problems we ran into before we knew the KIUL could not be changed.

### 3.2 Enrollment/Credential Provider

Many of the problems that we ran into while working on the enrollment application also occurred in the Credential Provider. Overall both performed similar functions. The enrollment takes the fingerprint template along with the username and password, and stores it within Active Directory. It would also take a second snapshot to verify that you stored a good template. The Credential Provider would also take the same information, but it would send it to Active Directory to be verified in the proxy. At the end of the project there were still things that we wanted to do with both; from finishing the project to adding in other biometrics to test, like RFID cards. There were also problems we came across with the actual capturing of the template and how to send or store the data. These were among the other problems that we had.

The original idea for the enrollment would possibly still work. At our stopping point, it could verify a user’s login information and open up the application that could take a fingerprint snapshot. Sadly after digging through Upeks information we discovered the function we were using originally would only make an image of the fingerprint, not create the template that we needed. After speaking with Upek we found which function was able to actually create the template. Unfortunately there was a problem with it. This template was not just a single fingerprint image; it was actually made up of several images. We feared that because the template was made up of more than a single image that our storage type would not be able to hold the template due to its size. This uncertainty also made us wonder if we could still send the template to Active Directory. We needed to be able to do this so we could store
the template and also verify that we stored a good template. Our last problem with Upek’s functions was that within their code it was unclear as to what the template was. Since we were unable to figure out what they had named the template we couldn’t send it. At that point our enrollment was able to open an already made Upek application but we were not able to work with any of the variables.

Let’s say that we were able to get the template sent to Active Directory. We would then have another problem. Though we knew the template was of binary type, it will most likely be too large to store. All three of the binary storage types we found in Active Directory could only hold 8-bits of data. We do know at least that the battle between the Active Directory LDS and ADAM could be easily solved. The separate server obviously would be the most secure way to go. There was one last thing that we planned to add to the enrollment. We wanted to further protect it from the administrator so we thought to encrypt the fingerprint template while it was getting sent to Active Directory. Performing this encryption would allow for another layer of security within Active Directory.

The Credential Provider luckily had fewer problems for us. Our Credential Provider was able to authenticate a username and password already. We also had it using one of the Upek functions that took a fingerprint image. Again, we would need to change this because we would want a template. Here, we would have had to write a completely new function. This function would either be worked into the actual GUI of the Credential Provider or have been a separate window that would pop up once you started to login. Either way it would need to grab the user’s fingerprint to make the template and then send it along to Active Directory to be verified.

3.3 Extending the KIUL

As we explained earlier the KIUL is the authentication structure within the Credential Provider that collects the username and password. It takes them to Active
Directory to be verified. We were hoping to go into the Kerberos.dll and just add in the binary structure to hold the binary template. To our dismay there were some problems with this. Like before, we’d have to find something large enough to hold the template. We also didn’t know how the KIUL would know when or how to grab this information. This was one of the main questions that we had when we contacted Microsoft. They told us that the KIUL wasn’t meant to be extended; it had to be used as is. We asked them if it was possible to extend the KIUL and be able to log a user in using Active Directory like this. They said that it wasn’t possible with Windows Vista.

### 3.4 The Proxy

Lastly we hoped to then make a custom proxy to intercept our KIUL. This would have sent the username and password to the LSA to be verified. The proxy would then retrieve the original template from Active Directory and perform a 1-to-1 matching. It would also decrypt the template before using it. Microsoft calls this an authentication package and again we hoped to extend one, like the KIUL, to use this additional information. We had already found that Upek had a function that allowed comparison of two already taken templates. The question about this custom authentication package was our other big question when we spoke with Microsoft. We got a similar answer here. Again they told us that it wouldn’t work and we’d have to try something else if we wanted to use biometrics with logging in.

### 3.5 Summary

We had originally thought that this design would work. We ran into many problems. All of these problems are summarized in the list following this paragraph. There were items we could work around at that moment and others that had to be pushed to the side to be dealt with later. We even had a few that made us stop and really
think about what to do. Each of the main functions had difficulties within them, which led us to examine them closer.

- Within the enrollment and Credential Provider, the function that read in the fingerprint didn’t make a template and would need a complete rewrite.

- The template size was larger than what we originally thought.

- The binary structures of Active Directory, both sending and storing, would not be able to handle the size of the data.

- There was no information on how to extend the KIUL. We found out from Microsoft that it was impossible to extend it to take along another piece of information.

- The proxy, which was going to be extended to use the data from the KIUL and do the verification of templates, was also not able to be extended.

Even after these difficulties we still hoped to solve the problem of how to securely login over a network. Microsoft did have good news after our discussion with them about the project. The news was that Windows 7 and Server 2008 R2 were able to do what we wanted (it wasn’t like previous applications we found that said they could store to Active Directory but then really stored the fingerprints locally). They actually implemented a similar idea to ours in their newest operating system and server which will be discussed in Chapter 4.
4.1 Introduction

In Chapter Three we discussed the problems that our developing application ran into. We then mentioned that Microsoft had actually implemented something similar to our idea that would work with Windows 7 and Sever 2008 R2. Microsoft calls it the Windows Biometric Framework or WBF, as we will refer to it. According to Microsoft, the WBF is used “to create client applications that securely capture, save, and compare end-user biometric information” [20]. Microsoft recognized the problem that people and companies wanted to use biometric information to login over a network using something similar to Active Directory. Like Microsoft stated, it now sounds possible to make an application to capture a fingerprint template and store it in a database. This also means that they found a suitable attribute to store the template, which was one of our problems.

We understand the overall idea of what WBF does. It can do what we wanted to implement, though to what extent? What other functions will WBF be able to do? We had our own ideas for a system like this so we will look at what exactly theirs does. They claim that it handles all biometrics, but for now we see that it only supports fingerprints. We already know that this is a good choice. There is already technology out there that can capture such information. WBF can handle some functions we wanted to implement like enrolling a new user and retrieving already stored templates [21]. We will see that Microsoft created an entire new framework to handle this idea.
There were also many new functions that they made allowing for a large degree of customization for companies using this framework.

After reading an overview of the system we found that the Microsoft solution still had not solved all of the issues we had thought relevant. There are additional functions that are included that we had not thought of like the support of different fingerprint readers. There are functions we wanted that were not included. One item is multi-factor security; at the moment they do not support any other information to login with (besides username and password). We had also thought to include RFID at some point. There were a few things that we had hoped to incorporate into our solution one day that WBF may soon handle, like extra biometrics.

The next obvious question was how did Microsoft accomplish a system like this? The information they had on WBF was vast. Part of the basic framework is set up like it was in past systems. New parts were then included, either giant chunks or new features in between old pieces. The applications all access the templates separately. We had thought to do this through existing databases though it makes sense to have a dedicated one. One requirement for WBF is that only certain fingerprint readers will work with it. Another shortcoming is that it does not contain a complete solution. It only contains a database and code samples. Microsoft does, however, provide detailed webpages to help in coding certain functions.

Not all systems are perfect and WBF is no exception. While we find it exciting that Microsoft had developed WBF there are still problems, things that could be changed, or functions that could be added. This new feature has the capacity to add in more biometrics besides just fingerprints, so why not add the support? With all the code they have written as help, why not have a basic enrolling system already in place? We also hope that they decide to add in other types of verifications, which would make systems even more secure.
4.2 What WBF Does

As we have discussed many times in this thesis we know the importance of using biometric information within the login system. It allows for a more secure login since it is close to impossible to replicate another’s fingerprint. Microsoft finally saw the need to include this in their newest operating system, Windows 7. Before WBF each biometric company had to develop it’s own drivers and SDKs. This is how we tried to go about designing our own application. The problem with this is these applications only work with a single biometric reader type, Company A’s won’t work with something made with Company B’s SDK. In Windows 7 Microsoft decided that this would be a problem no longer and that they would now support all fingerprint readers. This meant that all fingerprint devices would have to code new drivers to work with this system. SDKs would no longer be needed now because everything can now be done through WBF [22].

A single SDK allows for many of the good elements of this framework. Now all supported fingerprint devices will work the same. Users will not have to remember how to use each different device. Perhaps the best thing is that there is no more remembering how to use all of the different SDKs because there is one central framework that is used by all. The fingerprint devices can now be managed either locally or over a network [22]. The support Microsoft lends to those developing in WBF is ample. When we originally sought more information on Upek’s biometric device we found they barely had any documentation on it and it took a while to get a response from their support. We found there to be a good amount of information online about using and developing in WBF.

Earlier in the chapter we discussed the three main goals WBF wants to solve. These are: capturing a fingerprint, saving a template, and later comparing two templates against one another [20]. These three ideas work similarly to how we envisioned them working in our own system. The capturing process is quite easy. A fingerprint
is scanned multiple times and from the scans a template is created. The template is securely sent directly from the device to a database meant to hold just the templates. WBF will also handle all comparisons for matching two templates.

Besides the three main functions of WBF there are many possibilities for other functions it can perform. This is all thanks to the API of WBF. A programmer can create applications to do whatever they like with the biometric information. A common example is an enrollment application for new users to a system or maybe retrieving a template for another purpose. Microsoft does not provide any pre-written software to use with WBF although they have a lot of references on their msdn website on how to code your own applications. This approach allows for companies to customize their systems to work exactly the way they would like them to. The WBF solution allows for two different types of data flow; synchronous and asynchronous. Currently there are five code examples that Microsoft has posted: capturing, enrolling, identifying biometric information of a user, locating a biometric unit, and verifying user identity [23]. Each code page contains the sample code, a few main points on how it works, and header files that must be declared. There are two additional examples, which are for administrators only. These functions are how to retrieve a user’s identity and their credentials [23].

As we had discussed earlier in the chapter about our own application, a user’s credentials are very important. In the WBF framework there is a separate API, which just handles all of a user’s other information, like username and password, for example. All of this information is just as important as a user’s fingerprint template. The last important thing to understand about WBF is the sensor pool. There are three classifications of sensor pools: system, private, and unassigned. This piece of software does all of the work and is called a biometric unit. Each unit is assigned to one application and also one sensor pool. The sensor pool type allows the system to know what the application is allowed to perform.
4.3 Initial Design Vs WBF

There were plenty of ideas that Microsoft had for WBF that we had thought to implement in our own application. While researching what they had done we also started to see that they did things different, did things we did not think of, or did not do things we thought important. The basic idea was the same; to have a framework that would handle biometric information that could be used for logging into a computer. Due to limited resources our only option was to build this into the already existing Active Directory. Once we learned this would not work we wondered what the answer could be. Microsoft answered this; a separate framework would take care of just biometrics. This means they solved most of the problems we had. WBF handled everything with a few functions that could be customized to meet your needs.

Microsoft thought to do different things with their approach and most of the new ideas made good sense. While we were developing for a single company’s needs Microsoft decided that supporting all biometric devices would allow for consistency. If a company found they wanted to change brands on their biometric device the authentication would still work. This also allowed for a more unified system of writing an application.

As good as the WBF solution is, we feel as though Microsoft left some key ideas out. One of the main hopes that we had for our initial design was the opportunity for the support of multi-factor identification. At the moment WBF only supports fingerprint data while it claims to accept any type of biometric data. There is the hope that it will support more forms of biometric data but it does not as of yet. We had also hoped the system would take into account other forms of security to use to login besides just biometrics. Being able to add something in like an RFID card would allow for greater security.
4.4 How WBF Works

As we just discussed there are many similarities between what we wanted to implement and what Microsoft did with WBF. The next question we had was how did they accomplish this? Microsoft designed a framework that was completely separate from any previously existing structure. Our only option was to add our implementation to an existing framework. Microsoft was able to create a new framework because they had the resources to take on this large task.

We know the goal of the framework, but how does it actually accomplish handling all of the biometric data? Microsoft mentions there are three “core platform components” to WBF [24]. To use a biometric device it needs to be compatible with the Windows Biometric Driver Interface, which is the single GUI that handles the capturing of a fingerprint. The second is the WBF API, which we discussed earlier. It
allows the creation of custom applications to use the framework. The last piece is the Windows Biometric Service which links the other two parts together. It is responsible for securely sending all data between the interface and API as well as storing and retrieving needed data [24].

The most impressive part of what Microsoft accomplished was the actual structure of this new framework. As we have already pointed out they were able to implement a completely separate framework to just handle biometrics. Figure 4.1 shows the overall scheme of how Microsoft set up WBF. The main part is the Windows Biometric Service we discussed earlier. It controls all of the inner functions. The next layer is the piece that manages all of the different devices that can be used like fingerprint readers. The handling of all biometric data is up to the Biometric Unit. The Biometric Unit contains three different adapters, which talk to the biometric device. They are: the sensor adapter, the engine adapter, and the storage adapter. The sensor adapter is in charge of displaying the GUI for the device and capturing fingerprints, for example. Next the engine adapter creates the template and would also handle matching if it is needed. The final adapter, the storage adapter, maintains the template database [25].

As a programmer for a company using WBF you would have to create your own applications to use the authentication. To do so one would need background in C and C++ [20]. Microsoft also points out that the three adapters that handle the biometric templates within the Biometric Unit are not pre-coded but provide guidelines to coding your own. They point out that they are actually DLL files [26]. As we have mentioned before Microsoft has plenty of helpful documentation and examples on creating custom applications. Like the application we tried to implement, the credential provider would have to be rewritten to allow the collection of a fingerprint and creation of a template to be stored for verification. Microsoft provides code examples for managing credentials [28].
The last important function to any system is the administration and managing of a framework such as WBF. An expansive one like this has more ways to maintain the system compared to one we would have made. Any changes to biometric devices or updates needed for the drivers can easily be obtained from Microsoft. There would be no worrying about compatibly if there were updates. Administrators would have access to the Biometric Device Control Panel, which allows them to work on an array of options. There they can manage anything from deleting a user’s template to disabling a biometric driver [27].

4.5 Issues

We found most of what Microsoft did with WBF to be a great solution. Many functions were big pieces of our project to include biometrics as a security measure. There were however, some functions that we had problems with or issues that Microsoft did not touch on at all. There were two main issues we had: no multi-factor authentication, and no useable applications. Another issue is that WBF could only handle fingerprint data at this time.

As we pointed out, Microsoft calls WBF a biometric system even though it only handles fingerprints. Microsoft has indicated that other types of biometrics will be incorporated in WBF in the future. We hope that it will allow multiple types of biometrics for authentication at the same time. The authentication we wanted to add to our own application was an RFID card. Allowing more and multiple types of authentication would make the systems even more secure as we have already discussed.

There are plenty of code examples on Microsoft’s website for customizing WBF, but since you have to create a custom application to use the framework many companies may not choose to do so. It may be easier for companies to use the added features if Microsoft implemented a few simple applications to go along with the framework such as an enrollment, setup of the framework, and an updated credential provider.
4.6 Summary

From just a short overview the Windows Biometric Framework API sounds very impressive. We saw that it was able to do many of the things we hoped to do ourselves. Clearly Microsoft had seen this need earlier and perhaps their users told the company how they felt about the biometric login idea. Either way, being able to login to a computer over a network is now possible and a secure measure to take.

Microsoft’s WBF is able to handle a lot of functions. One main idea Microsoft developed was that of supporting all fingerprint reading devices. This allowed any of such devices to be used without worrying about compatibility. That also meant that SDKs were no longer needed because Microsoft supported writing applications to work with their WBF. Although they did not release applications to use with WBF, the platform allowed companies to write their own applications. These custom code modules could enroll users, verify them to allow onto a computer, and handle other information pertaining to the biometric data.

We did have some things to say about Microsoft’s system when compared to our own idea. While both had similar thoughts on how the authentication was set up and would work there were also some differences. WBF made some great improvements with features we had not thought of. Most of them were items we would have not been able to do because they were related to device drivers. There were a few ideas left out of WBF that we had thought to include that Microsoft had barely discussed. As we talked about earlier, multi-factor authentication is the best way to go though WBF, but it only supports one type of biometrics. Microsoft did say that it will one day handle more types although we hope they consider handling other types of authentication as well.

The amount of work that Microsoft had put into making the WBF was impressive. Unlike our idea to just make it work with existing software and hardware, they implemented a complete new framework to do so. Three core pieces were developed
to handle the capturing of a template: the storage/retrieval, and matching of templates. The solutions are the Windows Biometric Driver Interface, the WBF API, and the Windows Biometric Service. These pieces are what makes up the actual framework. Using the API, companies would be able to create custom applications to fulfill whatever need they had. There are also ways to maintain the system from an administrative point of view.

Finally, we had a few minor suggestions for WBF, which could be seen as improvements Microsoft could make in a future release. As we have pointed out they only handle fingerprint data but will expand to other biometric types. We thought the addition of other security measures, like RFID cards, would allow for a multiple authentication framework. The addition of already coded applications could allow more companies to use this biometric framework instead of making them code their own. Companies that would want to make larger and more complicated applications still could.
CHAPTER 5

CONCLUSION

It is clear that authentication on any system is useful for helping to secure information. Most current systems seem to lack either multi-factor authentication or only have a local authentication system and not one that is over a network. We have discussed three main topics as well as learned many lessons throughout this project. The implementation we worked on would have allowed all data to be stored in Active Directory allowing it to be secure. It would have eventually allowed for a user to login over a network with multiple ways to verify who they were. We were disappointed to find that in the long run this idea would not work out. Originally we were working on a solution that would be used with Windows Vista. Microsoft had then told us that it would not work with Vista at all. However, Microsoft also told us that they had already been working on a similar implementation in Windows 7, which we then began investigating.

We had first tried to solve this biometric problem ourselves since all research had showed at the time that biometrics could only be stored locally on a machine. The idea was to use Active Directory to control all of the templates as well as the rest of a user’s information. The enrollment was able to take in a fingerprint and store it in a separate partition just for the templates. Users would then be able to login over a network and the system could verify who they were based off of the fingerprint which would have been stored in Active Directory and not the local machine. The Credential Provider was rewritten to add in the function to capture a template of the fingerprint. The KIUL now takes over. Originally it is what takes a username and
password to Active Directory for verification. We had hoped to extend it to allow the template to be taken as well. Once sent to the LSA a proxy would first intercept the KIUL. The Proxy would first send the username and password to the LSA for verification. It would then retrieve the user’s stored fingerprint template and perform a 1-to-1 matching on the template that was sent with the KIUL. If the template matches and the password is correct the LSA would tell the Credential Provider that the user may be allowed to logon.

Near the end of this project we started to run into problems. We realized that we should have contacted Microsoft earlier then we did instead of letting the issues pile up. Just about each main function had one issue or another, which really made us stop and evaluate what we had done. We found out that the function we were using from Upek did not actually create a template like we thought it would, which meant the enrollment would have to be rewritten again. The templates that were created by the right function were too large to be stored in the binary structures of Active Directory. After much researching we found that no information on how to extend the KIUL was available. After speaking with Microsoft we found that both the KIUL and the proxy could not be extended to deal with templates as we wanted.

There was good news because Microsoft told us that Windows 7 would be able to do something similar to what we wanted to do. Their solution was the Windows Biometric Framework, which supports any fingerprint device as long as it has a compatible driver. Supporting devices this way means that separate SDKs are no longer needed; Microsoft supports development of applications that use the framework. WBF has three main functions of capturing a template: storing, retrieving, and matching are all supported. This means that it is now possible to authenticate using biometrics like how we had hoped. They also developed the WBF API, which as we said, would allow a company to write their own applications that can use fingerprint data to log a user in without having to store the templates locally. This is
all achieved because the framework stands by itself unlike our idea, which was going to be built into an existing system.  

While Microsoft’s WBF did improve upon ideas we had, there were ideas that we thought they left out or barely helped with. At the moment WBF only supports fingerprint devices but we hope that one day they will be able to support other biometric types. This would allow them to make a step towards using a more multi-factor approach to a user logging into a system. One suggestion we did have was to add in other security types like RFID cards. Perhaps having already finished applications would increase the interest for some companies that do not have the resources to make their own. What other ideas could they add to WBF? If Microsoft was to add more layers of authentication as well as other biometric types you could secure a system much more flexibly. There are many possible future applications for security, especially multi-factor authentication.
LIST OF REFERENCES


