Achieving Cross-Domain/Cross-Platform Single Sign On:

*UNIX, Windows, Kerberos, and Active Directory*

Dane Warren Henry

*August 07, 2007*
Cross-Domain/Cross-Platform Single Sign On, Month DD, YYYY

Table of Contents

1. INTRODUCTION ..................................................................................................................1
  1.1 Background and History ................................................................................................. 1
  1.2 Document Purpose ......................................................................................................... 2

2. HOW THE KERBEROS PROTOCOL WORKS ..................................................................3
  2.1 Windows Active Directory ............................................................................................ 4
  2.2 UNIX and Apache’s Role ............................................................................................... 4

3. IMPLEMENTATION .........................................................................................................5
  3.1 Setting Up The Active Directory Environment ............................................................ 7
    3.1.1 Setting Up the Client Workstation ........................................................................... 7
    3.1.2 Setting Up the UNIX Server (Including Apache and Kerberos) ......................... 7
    3.1.3 Configuring the UNIX User Account in Active Directory ....................................... 8
  3.2 Configuring the UNIX Environment ............................................................................. 9
    3.2.1 Initializing the Kerberos Environment ....................................................................... 10
    3.2.2 Installing and Configuring Mod_Auth_Kerb ............................................................ 10
  3.3 Testing for Successful Configuration of Cross-Realm Authentication .......................... 12

4. CONCLUSIONS ..............................................................................................................13

List of Figures

Figure 3-1 Environment and Authentication Process Steps .....................................................6

List of Appendices

APPENDIX A, CITED REFERENCES
1. Introduction

Providing users with a fast, easy, seamless, and most of all, secure method of accessing a data resource is one of the most important aspects of system design. Single-sign-on applications are the dream of every user and the dread of every system administrator. Single-sign-on is the ability to provide access to multiple protected resources through the single authorization of a single set of credentials. There are numerous ways to implement single-sign-on, and use of the Kerberos protocol is one of them.

Because many system environments employ the use of different operating systems for different aspects of the system, obtaining true single-sign-on is difficult. However, thanks in part to recent changes and innovations on the part of the Windows and Open Source communities, single-sign-on is possible even in a mixed operating system environment. Using the Kerberos protocol, and with proper configuration of both hardware and software, it is possible to securely authenticate clients from differing domains and across differing operating systems.

1.1 Background and History

MIT researchers initially created Kerberos to provide a secure sign-on process for their Project Athena. Kerberos spun off from that, becoming an entity in and of itself, and was first released to the public in 1989. Over the next several years, researchers at MIT worked on and ultimately developed what is now the current implementation of Kerberos version 5. Kerberos version 5 is an upgrade from the aging version 4 of the Kerberos protocol, which had numerous security issues. Version 5 set out to correct those issues while maintaining itself as a premier secure authentication protocol for enterprise level applications.

During the same period, Windows Active Directory, offered by Microsoft, the owner of the Windows Operating System, was growing to become one of the most used Lightweight Directory Access Protocol (LDAP) directory services.

First implemented with Windows Server 2000, Microsoft began using a revised version of the Kerberos version 5 protocol to provide authentication of clients for Active Directory. However Microsoft chose to heavily modify the protocol, in some cases adding extensions that are only compatible with other Microsoft products.

Finally, UNIX enters the picture, mainly because of its open source licensing agreements and because of the raw power that the operating system offers. UNIX was the first operating system to be developed by big-name corporations and universities, and was the operating system of choice for those institutions. Kerberos, being an open-source protocol itself, became heavily used as the authentication protocol of choice for most UNIX operating systems.

Ultimately, the mixing of the UNIX and Windows environments, coupled with Microsoft’s modifications to the Kerberos protocol, virtually guaranteed that proper cross-platform authentication was going to be difficult at best.
1.2 Document Purpose

Because of the mixing of the operating systems and the widely varied implementations of the Kerberos protocol, any sort of secure authentication between servers running different operating systems is extremely challenging. Authentication can never happen if the differing Kerberos protocols cannot agree on how to communicate the data. Kerberos is a secure, tested protocol that can and has successfully provided authentication for enterprise level applications. But any hope of mixing true MIT Kerberos with Microsoft’s implementation of Kerberos is gone.

There is, however, a solution that allows for UNIX servers to securely authenticate Windows clients against a Windows Active Directory structure. To understand this solution, we must first understand the particular structure and mechanisms of the protocols and components involved.

The remainder of this document is devoted to providing an understanding of these structures and mechanisms, and to providing the framework for developing a single-sign-on solution for any application that must authenticate users across operating system platforms.
2. How The Kerberos Protocol Works

The Kerberos protocol operates on a ticket system that is designed to authenticate clients from any operating system as long as the username is in Kerberos’ database. Kerberos’ database is formally known as the KDC, or Key Distribution Center. The sole purpose of the KDC is to authenticate Kerberos clients. Every user in the KDC is identified as belonging in a realm. Users within a realm are authenticated by the specific KDC for that realm; however, a single KDC can authenticate users from multiple realms.

The KDC and the Kerberos protocol itself work on the premise of tickets being passed back and forth between the client, the protected resource, and the KDC. These tickets are what authenticate the user and are the mechanism that provides security. The ticket system works much in the same way as a series of handshakes between various people. Before discussing the ticket process, it is important to understand the tickets themselves.

There are two types of Kerberos tickets: Ticket-Granting Tickets (TGT) and Service Tickets (ST).

A TGT is generated from the KDC and sent to the client when the Kerberos user is first authenticated during initial logon (through their username and password). The TGT (now contained on the client) is handed to the appropriate server anytime the client needs to be re-authenticated (for example, to access a protected resource). The TGT essentially provides basic authorization and validation of credentials, eliminating the need for the user to reenter their credentials.

An ST is created whenever the user must be re-authenticated in order to access a protected resource, which is done by passing the TGT from the client to the server to show that the user has permission to access the resource. STs are the tickets that provide direct authorization for access to a protected resource.

For example, a user logs onto a client workstation that belongs to a particular Kerberos realm. When the user logs on, after being authenticated by the KDC, a TGT is returned to the client. The user then attempts to access a restricted Web site. The Web server sends a response back to the client saying that the resource is protected and that the user needs to be authenticated in order to access the site. In addition, the response from the Web server also contains a list of protocols that can be used to authenticate the user. The client, seeing Kerberos on the list of accepted protocols, responds with the user’s TGT. The web server then forwards that TGT to the KDC to confirm that it is a valid TGT, that the user is who they say they are, and that they have permission to access that resource. If the TGT is valid, the KDC responds to the Web server with an ST, which the web server then passes back to the client. At this point the client again attempts to access the protected resource, and when challenged, provides the ST, which allows the user to view the requested Web site.

This process is provided in more detail, including a diagram of the environment that shows the order of communication steps, in Section 3, Implementation beginning on page 5.
2.1 Windows Active Directory

Windows Active Directory was formally released in Windows Server 2000 (though it had been previewed as early as 1996). By the time of its live release, Microsoft decided to base their authentication mechanism on a variation of MIT’s Kerberos security protocol.

Active Directory (AD) has a structure that is based on forests, trees and domains. Forests are the top level of the AD structure, and contain every object within the AD structure. Forests contain one or more Trees that in turn hold one or more Domains. Each domain can be broken down or grouped in Organizational Units (OUs). OUs include items like individual users and user groups. Additionally the general administration of AD usually occurs at the OU level, especially for such things as Group Policy and individual user administration.

Domain Controllers (DC), are the servers within AD that allow for user authentication and authorization. In addition, they allow for cross-domain authentication as well as enforce policies throughout the domain. In Kerberos terms, the DC acts as a KDC by authenticating users and returning the tickets that allow for access to a protected resource. This is because Microsoft chose to use Kerberos as an authentication protocol in its Windows environment.

To state it another way; the KDC for a Windows Active Directory environment is the Domain Controller. So that for the remainder of this document, when a KDC is referred to within the Active Directory environment, the reference is actually to the Domain Controller (because it functions as the KDC for Active Directory).

2.2 UNIX and Apache’s Role

UNIX is an open source operating system, considered to be a heavyweight when it comes to enterprise-level applications. Apache Web server is the primary Web server being used on the Internet today. As a result, UNIX and Apache are frequently paired together for the operation of many types of Web sites.

Because of the open-source nature of both UNIX and Apache, there are numerous extensions or modules available for either of them. One such module for Apache is called mod_auth_kerb. mod_auth_kerb is an open source project that allows Apache to use the Kerberos protocol as an authentication mechanism. This opens up huge opportunities for cross-platform authorization, because Apache can handle requests from any operating system or Web browser, as long as it operates along the HTTP protocol.

This cross-platform capability, in conjunction with the mod_auth_kerb module, is what allows the combination of Apache and Kerberos to handle authentication across UNIX and Windows environments.

NOTE: Refer to Appendix A, Cited References for the sources of the factual details cited in this section.
3. Implementation

Now that we have described the basics of the individual pieces involved, we can begin to set up a cross-realm, cross-platform authentication application.

While there are a wide variety of possible permutations to this setup, the configuration used as the example for this document contains the following components:

- A Windows Active Directory environment containing three separate servers, each running Windows Server 2003 (the Domain Controllers)
- A UNIX server with Apache Web server installed
- One or more client workstations running Windows XP and installed with Internet Explorer 6 - Service Pack 2.

The UNIX server must also contain a resource or directory that is flagged as protected within Apache, meaning that access to it requires authentication. For our example this directory will contain a Web page that only valid users can access. The user-access to the Web site will be through Internet Explorer (IE) on the client workstation.

While most of the necessary configuration for the environment will occur on the Web server, there will be some “back end” configuration on the Domain Controllers as well.

The following steps provide a basic overview of the process that will occur and thereby indicate successful configuration of our environment to handle cross-platform user authentication:

1. User logs onto the client workstation using Active Directory credentials (username and password).
2. The Active Directory environment returns a Kerberos Ticket Granting Ticket (TGT) to the client workstation.
3. Once logged in, the user opens IE and attempts to access the protected Web site.
4. Web server sees that the web resource being requested requires authentication, responds to the IE client with a 401 (Unauthorized), but includes a list of protocols the client may use to authenticate the user for access. Kerberos is on this list.
5. IE client sends the Kerberos TGT to the Web server as authentication for the user.
6. Web server sends the TGT to the Active Directory server to confirm it is a valid TGT.
7. AD server authenticates the user’s credentials identified in the TGT and sends a Service Ticket (ST) back to the Web server.
8. Web server passes the ST back to the IE client.
9. IE client then re-attempts access of the protected resource, this time with the ST.
10. The web server returns a 200 (OK) response. User now has access to the protected website.

While this seems like a very long process, it typically takes only nanoseconds to authenticate the user and provide access to the protected resource, depending on how far away the servers and clients are from each other.
Figure 3-1 below provides a diagram of the environment described and configured in this paper, along with the communication paths required to complete the process steps listed above.

Figure 3-1  Environment and Authentication Process Steps
3.1 Setting Up The Active Directory Environment

The first step is to set up the Active Directory environment by building and installing three Windows Server 2003 servers with the optional Support Tools. These servers will act as the Domain Controllers that provide the functionality for the Active Directory environment. The optional Support Tools are necessary on the Domain Controllers in order to set up the required accounts for Kerberos.

Using the three servers, create the directory structure such that there is one parent forest and two child domains (each of which will then act as trees in and of themselves). While technically this step is not necessary for basic Active Directory, this configuration (and hence our example) is designed to demonstrate authentication within a cross-domain environment.

The parent forest should not contain any users other than the top most administrators. In our example the parent forest is named `foo.bar.edu`. The two child domains are named `primary.foo.bar.edu` and `secondary.foo.bar.edu`, and should then be set up to have a 2-way transitive shortcut trust. This will ensure that cross-realm authentication can occur successfully.

**NOTE:** While the parent forest should only contain the top most administrative users, the two child domains can have any number of users or other OUs within them; those particulars are not important.

3.1.1 Setting Up the Client Workstation

After the domain structures are set up, build a workstation for the user to log into and use to access the Web site. Install Windows XP on the workstation and set it up to use the DC to authenticate the client’s username and password. The user account will be set up at the end of Section 3.1.3, Configuring the UNIX User Account in Active Directory beginning on page 8.

The last step on the client workstation is to ensure that within Internet Explorer (IE) the Advanced Option for “Enable Integrated Windows Authentication” is checked. To do this:

1. Open Internet Explorer.
2. From the Tools menu, select Internet Options.
3. Click the Advanced tab to activate it (it is the right-most tab).
4. Scroll down the list of Options to find Enable Integrated Windows Authentication.
5. This option must be checked. If the box is not checked, check the option and then restart the browser by closing IE and opening it again.

3.1.2 Setting Up the UNIX Server (Including Apache and Kerberos)

Finally, install a UNIX server with Apache 2.x Web server on it. Once the initial Web server has been set up, install MIT Kerberos, either pre-compiled for the specific UNIX installation or from source. If you are installing from source, use the following commands as a basic guide:

```
gunzip krb5-1.x.x.tar.gz
tar -xf krb5-1.x.x.tar
cd krb5-1.x.x/src
./configure
make
```
**make install DESTDIR=*/path/to/destdir/**

**Mod_auth_kerb** still needs to be installed, but before that happens, some configuration needs to occur on Apache, Kerberos, and Active Directory. There are two configuration files that will require changes; the first is for Kerberos (**krb5.conf**) and the second is for Apache (**httpd.conf**). Those instructions appear later in this document.

### 3.1.3 Configuring the UNIX User Account in Active Directory

In order to allow the UNIX Web server to use Active Directory to authenticate clients, you need to create a service account within Active Directory for the Web server. The user account for the UNIX Web server will be created on the **primary.foo.bar.edu** domain.

The user account created for the Web server in Active Directory should have a name similar to the name of the Web server so that it can be easily recognized as the Web server account. However, this is only recommended for usability and is not a functional restriction (the Web server user account can be called anything you want). This point about the lack of a functional limitation is important because the SAM account name (User logon name [pre-Windows 2000]) can only have 20 characters. If you are configuring an environment where the Web server has a name longer than 20 characters and you attempt to create a user account that matches the Web server name, you may encounter issues. The Service Principle Name or SPN (User logon name), on the other hand, has no such character limitations.

In our example configuration, the SAM and SPN will be the same name. However, since the SPN and the SAM account names can be different (if the SPN name is too long to be a valid SAM account name), the **ktpass** command identified below ensures that the SPN, which in the case of Kerberos is the only name we are concerned with, is mapped correctly to the SAM.

For our example environment, the name of the UNIX Web server is **access.bar.edu**. So to set up the account in Active Directory, the “User logon name” (SPN) is configured as **access.bar.edu**, and the “User logon name [pre-Windows 2000]” (SAM) is also configured as **access.bar.edu**.

After the Web server account has been created, certain flags on the account must be set. Specifically, the account must have the all of the following options enabled:

- Password never expires
- Use DES encryption types for this account
- Do not require Kerberos pre-authentication
- User cannot change password.

Using the Support Tools that were installed when the Domain Controller was built, use the **ktpass** command to create the mapping between the SAM and the SPN, and to set the SPN for the particular server. The generic command for this step is:

```
ktpass -princ HTTP/access.bar.edu@PRIMARY.FOO.BAR.EDU -mapuser PRIMARY\ACCESS.BAR.EDU -crypto DES-CBC-MD5 -ptype KRB5_NT_PRINCIPAL -mapop set +desonly -pass Password1! -out c:\access.bar.edu.HTTP.keytab
```
Notice that the **ktpass** command also generates a **keytab** file from the Domain Controller. This is an essential file and needs to be placed on the UNIX server. The **keytab** file can be placed anywhere on the UNIX server. This location will be entered later when another **keytab** file is generated from within UNIX using the **ktutil** command (in the next section).

Finally, you must create at least 2 user accounts, one in **primary.foo.bar.edu** and one in **secondary.foo.bar.edu**, so that the cross-realm authentication can be proven. It might be useful to create multiple user accounts in each domain to test with, but strictly speaking only one user account per domain is necessary to verify success.

### 3.2 Configuring the UNIX Environment

After the configuration for Active Directory is complete, it is time to configure Apache and Kerberos on the UNIX Web server. Be sure before starting this part of the configuration that the **keytab** file (that was generated using the **ktpass** command above) has been copied to the UNIX server.

The first action necessary is to regenerate the **keytab** file from within UNIX. This is done using the **ktutil** command that came with the Kerberos package. It is a standard straightforward process that requires only the following commands:

```
ktutil
  rkt {enter path for saved keytab file}/keytab
  wkt /etc/krb5.keytab
quit
chmod 644 /etc/krb5.keytab
```

In the **ktutil** command, **rkt** is for “read keytab” meaning that it reads from the **keytab** file that was generated in Active Directory and copied to the UNIX server. The **wkt** stands for “write keytab” and generates a UNIX version of the **keytab** file using the information from the Active Directory **keytab** file, and then placing it into the directory identified. The default location for the Kerberos **keytab** file is **/etc/krb5.keytab**, however the file can be stored anywhere if you choose to alter the **wkt** segment of the **ktutil** command from what is shown above.

The **chmod** command is executed in order to allow other users in UNIX to be able to read the **keytab** file, which is necessary for authentication.

Before **mod_auth_kerb** can be installed, a working Kerberos environment needs to exist. If there isn’t already an **/etc/krb5.conf** file, you must create one. For our example environment, the information in this file would appear as follows:

**NOTE:** For specific implementations, the domain and server names appearing in this file would match the names of the applicable servers resident in your environment.

```bash
[libdefaults]
default_realm = PRIMARY.FOO.BAR.EDU
default_tkt_enctypes = des-cbc-md5 des-cbc-crc
default_tgs_enctypes = des-cbc-md5 des-cbc-crc
```

```bash
[domain_realm]
```
access.bar.edu/{protected_directory} = PRIMARY.FOO.BAR.EDU
access.bar.edu = PRIMARY.FOO.BAR.EDU
.bar.edu = PRIMARY.FOO.BAR.EDU
bar.edu = PRIMARY.FOO.BAR.EDU

[realms]
PRIMARY.FOO.BAR.EDU = {
    kdc = {ip.address.of.PRIMARY.DC}
    default_domain = PRIMARY.FOO.BAR.EDU
}

The [libdefaults] section of the file details the encryption types to be used as well as the default realm. Remember that realm and domain are for the most part interchangeable; one deals with Kerberos and the other with Active Directory.

The [realms] section is where the individual realms are declared and the KDC is listed. Each realm can have multiple KDC’s, and in fact it is a good practice to set up slave KDC’s since having only one KDC allows for a single point of failure. If our primary.foo.bar.edu DC goes down, then no one can log in and be authenticated against that domain. The same is true with a KDC; if the KDC goes down, no one is able to be authenticated. Since in this case our KDC in Kerberos is the DC for primary.foo.bar.edu, the two failures are one in the same. To list a slave KDC, simply add the following below the line for the master KDC:

    kdc = {ip.address.of.PRIMARY.DC.backup}

The default_domain line does exactly what the name implies, identifying the default domain or realm that is to be used.

### 3.2.1 Initializing the Kerberos Environment

To initialize the Kerberos environment, run the `kinit` command. Using our example Web server username, the `kinit` command would read as follows:

    kinit HTTP/access.bar.edu@PRIMARY.FOO.BAR.EDU

The command execution will ask for the password of the Web server account (the one created on the DC). Once finished, the successful result can be tested using the `klist` command, which provides a list of the tickets that the UNIX server has authenticated for itself.

### 3.2.2 Installing and Configuring Mod_Auth_Kerb

Now that a working Kerberos environment exists, we can install `mod_auth_kerb`. There are precompiled packages for specific UNIX installations, and some Linux distributions come with `mod_auth_kerb` pre-installed. However, if you are installing from source, use the following as a basic guide (where you supply the path to the destination direction and the path to Apache):

    gunzip mod_auth_kerb-5.x.tar.gz
    tar -xf mod_auth_kerb-5.x.tar
    cd mod_auth_kerb-5.x/src
The next step is the configuration for Apache, which, among other things, will point to the location of the keytab file generated via the instructions in Section 3.2 Configuring the UNIX Environment on page 9. If you put the keytab file in a location other than the default location specified in those instructions, you will need to change the identified location in the Apache httpd.conf configuration file.

In addition to editing the location of the keytab file (if necessary), there are two other additions that need to be made to httpd.conf.

The first change is to add the following line to the list of LoadModules (if any others exist):

```
LoadModule auth_kerb_module modules/mod_auth_kerb.so
```

This change tells Apache to load the mod_auth_kerb module so that Apache can use the Kerberos protocol to authenticate clients.

The second change can go anywhere in the file, but for ease of finding it later it is best to put it at the end. Add the following tag to the httpd.conf file (substituting your environment’s details for our example environment’s details where appropriate):

```
<Location {/protected_directory}>
  AuthType Kerberos
  KrbMethodNegotiate on
  KrbMethodK5Passwd off
  KrbAuthoritative on
  KrbAuthRealms PRIMARY.FOO.BAR.EDU
  KrbVerifyKDC off
  KrbServiceName HTTP/access.bar.edu@PRIMARY.FOO.BAR.EDU
  Krb5Keytab /etc/krb5.keytab
  require valid-user
</Location>
```

The <Location> tag specifies which directory in Apache’s home directory requires authentication to access. The lines inside the tag specify the protocol to be used (Kerberos, thanks to mod_auth_kerb), the Kerberos realm the clients should belong to, and the service name required to get into the KDC (which is the Web server account name created on the DC of primary.foo.bar.edu).

Additionally the <Location> tag points to the location of the keytab file that was generated using the ktutil command. This is where you would specify the location of the keytab file if you placed it in a different location than is the convention.

One final optional change to httpd.conf is to increase the log level. Look for the line that specifies the log level, and set it to debug by changing it to read as follows:

```
LogLevel debug
```

Save the httpd.conf.
3.3 Testing for Successful Configuration of Cross-Realm Authentication

In order to test the validity your authentication setup, place a dummy webpage into the `/protected_directory` in Apache. Then start up the Apache server. The most common way is by using the following command (although creating a startup script is also useful):

```
{path/to}/apache/bin/apachectl start
```

Before actually testing, it is often helpful to tail the Apache `error_log` file to make sure that `mod_auth_kerb` is performing its job successfully. To tail the log, enter the following command on the UNIX Web server using the IP address of the client workstation:

```
Tail -f /path/to/Apache/logs/error_log | grep {client.ip.address}
```

With the tail on the log configured and running, sign onto the client workstation as one of the users configured on the `primary.foo.bar.edu` domain, and then attempt to access the protected resource (the dummy web page placed into the protected directory). If you have properly set up Kerberos and Active Directory for the `primary.foo.bar.edu` domain, both of the following will occur:

- You see the webpage without any prompt for a password
- The apache log shows the following (or something similar):

  ```
  kerb_authenticate_user entered with user (NULL) and auth_type Kerberos
  kerb_authenticate_user entered with user (NULL) and auth_type Kerberos
  Acquiring creds for HTTP/access.bar.edu@PRIMARY.FOO.BAR.EDU
  Verifying client data using SPNEGO GSS-API
  Verification returned code 0
  GSS-API token of length 161 bytes will be sent back
  ```

Log off of the client workstation.

Continue to tail the Apache log and log onto the workstation again, this time as one of your `secondary.foo.bar.edu` domain users. Attempt to access the protected resource again (now as the second user). If you receive the same positive result, your cross-realm authentication is verified as successful.
4. Conclusions

Successful set up of a Kerberos environment gives system designers the freedom to use a myriad of hardware and software and still have them all be compatible with one another. This is an essential feature in today’s world where the vast majority of workstations are running some version of Windows and the majority of the servers on the web are running Apache, usually paired with UNIX or Linux. Since its adoption by Microsoft, Kerberos as a secure authentication protocol has become one of the most heavily used protocols in existence, spreading even to Apple Macintosh’s OS X operating system. This means that the 3 major operating systems being used today can all use the Kerberos protocol to authenticate users against a KDC, or as our sample configuration demonstrates, Active Directory.

The configuration demonstrated in this document is just one of hundreds of setups that are possible and can occur in a live environment. Once the initial structure is created within Active Directory and UNIX, any variation on this setup can be implemented, including the (recommended) addition of backup Domain Controllers and support for hundreds or thousands of workstations and usernames.

The need to have a secure authentication mechanism for identifying clients is an ever-present issue and is creating an ever-increasing demand on system designers and system administrators. Kerberos offers a seamless solution for near-single-sign-on operations that is cost-effective and highly cross-platform compatible. Because of the mixed environments that exist now and which are being implemented every day, the demand for a product that can handle everything from every operating system only increases the appeal of the functionality Kerberos provides.

Kerberos does have some low points though, such as single point of failure, overall complexity in cross-platform applications, and despite its popularity, a lack of detailed documentation to aid system administrators in anything other than basic setups. The single point of failure is (as mentioned earlier) the KDC. However (as also mentioned earlier) this risk can be significantly mitigated by setting up slave KDC’s to take over if or when the master fails.

In the case of Active Directory, having multiple DC’s for the same domain should be enough to eliminate the threat of being locked out of authentication because of a DC failure, simply because of how Active Directory works and the way it performs DNS lookups to find the closest DCs for that domain.

Despite the complexity of setting up Kerberos, compared to other methods of cross-platform and cross-realm authentication it requires no more than the time and patience needed to learn something new. The lack of true documentation only becomes an issue when attempting to use Kerberos to do something that hasn’t been done before or is not a typical setup, such as the environment and uses described in this document.

While Kerberos does have some shortcomings, the years of testing in both controlled and live environments lend to it being one of the most secure protocols in use today. Because of the extensive trials that Kerberos has passed, it can be trusted as a solid and (as our example shows) versatile authentication mechanism.
APPENDIX A
CITED REFERENCES
Cited References for this document include the following publications:

