Kerberizing Our Network

Simon Wilkinson
School of Informatics
University of Edinburgh
simon@sxw.org.uk

April 5, 2006

1 Introduction

Around 5 years ago, the University of Edinburgh merged together a number of its computing related departments to form the new Division (later renamed School) of Informatics. This merger involved the unification of three distinct computing networks and presented the opportunity to develop a new, state of the art, computing infrastructure which we named DICE.

As part of this infrastructure we have deployed a single-sign-on authentication and authorisation system based around Kerberos and LDAP. This paper will discuss the authentication portions of this system, examining our successes and failures, the experiences of our systems staff and users, and hopefully provide pointers for those considering deploying a similar infrastructure.

The School of Informatics hosts around 4500 user accounts, and 2000 machines. Our infrastructure comprises of Red Hat / Fedora Linux, Solaris and Mac OS systems. Whilst there is currently no emphasis on MacOS or Windows infrastructure support, we do supply services to a number of these clients.

2 Background to Single Sign-on and Kerberos

Single sign-on is an often misunderstood and misused term. In our deployment, and in this paper, single sign-on has two distinct aims.

- The user should only be required to authenticate once per session.
- Application services should never get access to the user's password, and so, the compromise of one application server should not compromise the network.

The use of the same password shared or synchronised between multiple services is not single sign-on, and was considered unacceptable to meet our security needs.

Before the merger, our infrastructure varied significantly between the individual departments. However, it could be characterised by the use of NIS-distributed password maps for some services, along with locally managed password files for others, with increasingly baroque solutions for moving accounts (and passwords!) between these two sources.

We wanted to unify these authentication solutions with the eventual goal being that every user would only authenticate once at the start of each working day. Following this no further passwords would be required to access services, and this would be achieved without caching passwords on the local disk. The only services which would require additional authentication would be those involved in administering the system, for which a further password would be requested.

One of the key goals of the new infrastructure was to remove any dependence upon machine-based trust, where access is given based purely on the network address of the client or, worse still, upon who the client says is using the machine (rsh, for example). We had a requirement to be able to allow users root access on arbitrary clients on the network, without compromising the security of the services that those machines accessed.

In addition to this, we had another technical goal - we wanted to minimise our key management overhead. In the years leading up to the merger a number of services had proliferated within the departments which required the generation, authentication, storage and distribution of key material. Services such as SSH and HTTPS web servers all required a high level of administrator involvement to either generate, or vouch for, the certificates or key material they used. We wanted to provide one master key for each host we managed, and then use that key to verify any additional keys required by services provided from that host.

3 Protocols and implementations

At the time of our initial investigation, Kerberos[1] was the only serious player in the single sign-on field. There were, and are, other solutions. Generally these are based around the use of an X509 public key infrastructure, with key material either stored on smart cards, or in a central certificate store. None of these solve the initial authentication problem as simply as Kerberos does, and to date none of them have achieved the same level of ubiquity as Kerberos.

Kerberos was originally developed by MIT as part of Project Athena, and most Kerberos implementations in current use descend in some way from MIT's original implementation. During the 80s and 90s export of Kerberos code (and many other cryptographic products) was limited by the US government. This led to the implementation of a new Kerberos library, Heimdal[2], by the Royal Institute of Technology (KTH) in Sweden. The open source market is now split between these two libraries.

We were fortunate enough to start our design efforts shortly after the lifting of these export control restrictions, and it was clear that both of our operating system vendors of choice (Sun and RedHat) were standardising around the MIT implementation. Rather than recompiling all of our system packages to use an alternate Kerberos implementation, we decided to use MIT's libraries and KDC. It's important to note that this doesn't prevent us from deploying clients (such as those running a BSD operating system) which use Heimdal, or for that matter Microsoft or CyberSafe's commercial Kerberos implementations.

Since we made the decision to base our architecture on Kerberos, there have been a number of changes in the computing world. Despite the arrival of other public-key based technologies such as Globus's GSI[3], none of these changes have invalidated our decision to go down the Kerberos route. Many major system vendors now ship with Kerberos support included - it is trivial to add machines running most Linux's, all the BSDs, Mac OS, AIX, HP/UX and many more to our Kerberos infrastructure. The increasing plug-n-play qualities of this architecture have helped greatly with its acceptance, and with the argument for deprecating the use of other means of authentication to our services.

4 Architectural Considerations

We have a number of particular architectural peculiarities within the DICE environment. Whilst the details of these are specific to our local circumstances, it's likely that many sites may have similar requirements, so we'll discuss some of these here.

4.1 Installation

Installation, and ongoing system configuration, is performed by means of a locally developed, automated system named LCFG[5]. This system handles the initial installation of a machine and the configuration of all of the services it provides, along with maintaining and upgrading the software and configuration as required. One of the key advantages of the LCFG system is that these configuration changes can be performed without human intervention, so it was vital that a mechanism be provided to obtain new Kerberos service principals with a similar lack of intervention.

During machine installation we require the person installing the machine to enter an administrative password, which is used to obtain a Kerberos identity for the machine. The key material for this machine identity is stored on the local disk so it can be used by unattended processes running with root privilege on the machine to authenticate themselves to other services. Amongst these services is kadmin, running on the KDC, which allows the machine to obtain key material and service principals for all services configured to run on that machine.

4.2 Foreign key management

We also use the machine's local Kerberos key to establish its identity when requesting other key material. We run an automated X509 certificate signing service named sixkts[6], which provides machines with an unattended mechanism for obtaining X509 server certificates for web, or other TLS/SSL protected, servers that they may run. The machine generates its certificate and private key locally, in the same way as if applying to a normal certification authority. It then uses a Kerberos authenticated connection to a central service to request the signature of this certificate. Providing the certificate is for a service that the machine is configured to run, the service will sign the certificate and return it to the machine.

In the past, we also used a similar mechanism to verify a machine's SSH keys. In this case, SSH keys were held in an LDAP database which the machine would use its Kerberos identity to authenticate to. It was then permitted to upload its SSH public key to the database from where clients could fetch it as required. This mechanism is no longer widely used as all of our clients now support Kerberized ssh key exchange, which is discussed in more detail below.

4.3 Disconnected Authentication

One of the initial requirements of DICE was that it be easy to support disconnected machines, such as laptops, without having to create a large number of special case options for them. To this end, we developed a mechanism where a machine could store a user's Kerberos passwords locally in order to authenticate them when the network, and the Kerberos servers, were not available. The set of passwords stored on a machine was restricted those of the machine's owners, as defined in our configuration database. Whilst the security of this mechanism was acceptable within our overall world-view, it was far from ideal.

We're now moving away from supporting Unix laptops, so its likely that this mechanism will die off. PADL have produced a set of PAM modules[7] which support password caching in a far more robust fashion from the ones we're currently using, and these would be strongly recommended to people wishing to implement disconnected authentication in the future.

One final consideration here is whether you need to have the same passwords for local and network access on mobile devices. During our design stage we decided that this was desirable, especially given our desire not to special case mobile machines. Current usage would, however, suggest otherwise. By far and away the most common usage of Kerberos on mobile devices utilises a user selected authentication scheme to gain access to the machine itself, followed by a distinct Kerberos login when network services are required.

4.4 Kerberos Servers

One of the main considerations in designing a Kerberos infrastructure is selecting the location of the Key Distribution Centres. It is imperative that these servers are both highly secure and highly available. In the event of a security incident on a machine hosting a KDC every password on your system should be considered compromised. Also, if all of the KDCs on your network become unavailable for any reason, then no one will be able to use any of your services.

Selecting the correct number and location of KDCs is therefore highly important. If you have sections of your network that are likely to become partitioned from the rest of the system (for instance, geographically distinct sites connected by an unreliable backbone) then it is a good idea to locate a KDC at each of these sites. However, the need to ensure both physical and system security for all of your KDCs may be a deterrent to deploying KDCs in locations where suitable secure facilities are not available.

Within Informatics we have a KDC at each of our four geographical locations. This has already proved invaluable by ensuring the continuation of our service in the face of failure of the central University backbone which links our sites.

5 Service Population

We were fortunate in that we deployed Kerberos as part of a larger roll out of a new computing infrastructure. This meant that we simply a new account for every user on the new system, rather than having to handle migrating passwords over from the old databases. Migrating passwords into Kerberos is particularly difficult, because Kerberos needs the plaintext form of the user's password - the hashed form that's in traditional Unix password files is not suitable.

We initially populated our KDC using the same technology as we continue to use to create new accounts on it today. A locally developed set of perl modules tie together account information from a number of sources, including the school's HR database and various systems maintained by central Computing Services, which hold information about allocated username and UID. These modules handle account creation, manipulation and deletion in both our Kerberos database, and our LDAP directory.

6 Kerberizing Services

An authentication system isn't any use without services which can use it. At the heart of Kerberos is a requirement that applications support its tickets for authentication purposes. Having servers accept passwords which they then verify against your Kerberos servers, and thus simply using Kerberos as a central password database, removes any security benefits and completely misses one of the key points of single sign-on.

Unfortunately, getting applications to support Kerberos can be easier said than done. It requires standardised mechanisms in the protocol definition, as well as requiring that work be done by all implementers to realise those mechanisms in their code. Due to the IETF's requirement for their protocols to provide strong security through a small number of standardised means, these mechanisms are defined for many protocols, with the notable exception of HTTP. We take a look below at the options we've chosen for Kerberizing many of our services, however a few general points need to be made first.

In an environment where many people are deploying new services it is vital that everyone is aware of the importance of having these services fit into the overall authentication architecture. To truly realise the benefits of a single sign-on system as many services as possible must interface to that system. As, in many cases, deploying services that support Kerberos may require slightly more work, or some custom configuration, it is important that all systems staff appreciate the need to carry out that work. In more extreme cases, Kerberizing some services may restrict the choice of software, require local code to be written, or even involve standardisation work. In all of our software decisions since developing DICE we've tried to weigh up the benefits of maintaining our single sign-on environment, versus the development costs in doing so. In general, this has resulted in a cohesive system where the vast majority of our services can be accessed using Kerberos authentication.

We've found that the services we've deployed have fallen into two camps. There are services that can natively use Kerberos, and those which require some form of 'shim' to connect them to the Kerberos infrastructure. It is vital that these shims are used with extreme caution - poor infrastructure design when it comes to their choice and implementation can cripple the entire system. We use two shim systems - KX509 and cosign, which we will discuss in more detail below.

It is also important to be pragmatic. No matter how desirable it may be to remove direct password authentication for all services in favour of Kerberos authentication, our experience suggests that it is highly unlikely that this is achievable. There are a number of environments in which it is unlikely that initial Kerberos authentication will be available, such as on web-cafe style machines, or on palmtop computers. In these cases it is necessary to continue to provide password based access to services. However, it is important to consider the vulnerability of these passwords with respect to the access that they can provide, and secure services appropriately. For instance, it is highly desirable that all passwords are transmitted across the network via an encrypted channel such as a TLS secured connection. The implications of continued password use form part of broader concerns about passwords and realms of trust that we'll discuss in the Future section, below.

6.1 Shell access

Historically, remote access to our systems was provided through a combination of telnet, rsh and ssh. Due to security concerns, rsh use was already being phased out before the transition to DICE. Whilst mechanisms exist to perform Kerberos authentication over all of these protocols (and, in fact, the MIT Kerberos distribution ships with clients and servers for both rsh and telnet), the decision was made to standardise on the SSH protocol. We had a number of reasons for this, all of which have stood the test of time. The support for inbuilt port and X11 session forwarding makes ssh more desirable to users than either of the other two mechanisms and, in addition, we were concerned about the ongoing support situation for Kerberos versions of both the r-command suite and telnet.

A number of mechanisms exist for supporting Kerberos in the SSH protocol. Two of these are in the process of standardisation as an RFC[8], and are supported by many vendors. Other mechanisms do exist, but are considered to be less secure. Their use is not recommended, and they will generally only be found in legacy code.

Two mechanisms are being standardised due to the layered nature of the SSH protocol, which provides two locations at which Kerberos authentication may be performed - either as part of the user authentication step, or during the initial key exchange. Each mechanisms provides support in one of these locations. Both of these have advantages and disadvantages. Using key exchange allows the use of Kerberos not only to authenticate the user to the server, but also to authenticate the server to the user. This has the benefit that administrators need not distribute ssh host keys, representing a huge saving in the time and effort spent performing key management tasks.

Unfortunately, the most commonly used SSH server for Unix, OpenSSH, does not support key exchange as part of the standard distribution. However, many vendors including Apple and Debian have patched their distributed OpenSSH to include key exchange support. Patches to add key exchange support are also available from the author's web site[9]. Solaris's bundled SSH client and server, SunSSH, supports both user authentication and key exchange via Kerberos.

The patches to OpenSSH to support both key exchange and user authentication were written by the author as part of our Kerberos deployment process. We use this patched version of OpenSSH as our exclusive means of providing remote login to our systems. Whilst the user authentication patches were eventually accepted as part of the OpenSSH core, there has been no movement on getting the key exchange code similarly included.

One key advantage of providing a Kerberized SSH service is that other services may then be layered on top of it. We run CVS, SVN, rsync and a number of locally developed protocols on top of a ssh connection layer which handles all of the authentication requirements.

6.2 Email

Until recently, one of our significant problems with Kerberizing the email service was the lack of support in one of our recommended clients - Mozilla. For a long time, Mozilla has been our recommended GUI Email client, across all of our platforms. The lack of Kerberos support in the Suite, and later in Thunderbird, was a significant impediment to pushing for the widespread use of Kerberos authentication to our mail servers. Recent work by the author[10], in conjunction with Mozilla staff, has resulted in Kerberos support for POP, IMAP and SMTP in the latest (1.5) release, which will hopefully reduce the number of password authentications to our mail server.

It's worth noting that most Unix mail clients do now have Kerberos support, as does the standard mail client on Mac OS X - our problem was due to our choice of supported client, rather than any lack of products on the market!

Whilst we've provided a Kerberized IMAP and POP3 service using the University of Washington server since the early days of the DICE deployment, we've only just started work on providing a Kerberized SMTP service. This work is using a standard RedHat sendmail configuration.

6.3 Printing

Initially, we controlled all printing from Unix hosts using the Kerberos support in LPRng. However, problems with this code, together with a requirement for non-authenticated printing for Windows clients through Samba, have resulted recently in a relaxation of this requirement and all of our printing from local hosts is currently unauthenticated.

It's hard to see which direction our printing system will take in the future. LPRng's Kerberos support appears to have problems that we have been unable to trace (although it works well for others). CUPS would appear to be its obvious successor. However, a convincing Kerberos based security solution for it is not yet available, especially given the Internet Printing Protocol's layering on top of HTTP, and the issues with that protocol which we discuss below.

There are discussions within the CUPS community about implementing Kerberos support in a future release[11] although these seem to be at a relatively early stage.

6.4 Directory Services

With the move to the new authentication infrastructure, we also deployed a new authorisation and directory services system using LDAP. We've now been running OpenLDAP for a number of years. Initially, getting OpenLDAP to work reliably with Kerberos authentication was problematic at best. However, work done during the intervening years has improved both the reliability and robustness.

One significant problem was the lack of thread safety in MIT Kerberos. Whilst support for locking is now available in recent versions of the MIT distribution, we added locking support to Cyrus SASL which we currently use in production. Other sites report that Heimdal is significantly faster than MIT Kerberos for busy services, as well as not suffering from the thread safety issues of older MIT versions.

Discussion of the authorisation system we implemented on top of LDAP is outside the scope of this paper, but it should be noted that a distributed authorisation system will be a key part of any successful Kerberos deployment.

6.5 Instant Messaging

We're currently investigating a Jabber based Instant Messaging Service, both for interactive communication, and for transmitting monitoring information. We're using Jabberd2 on the server side, and a mixture of Gaim, Psi and AdiumX on the client side. Jabberd2 and Gaim both now both include locally developed Kerberos support in their distributions, Psi and AdiumX are currently maintained through local patches[12].

Jabber is interesting because it is an arena in which Kerberos support should have been trivial to achieve. The underlying protocol, XMPP, was standardised through the IETF to support SASL authentication - for which a number of toolkits exist providing GSSAPI capabilities as standard. Unfortunately, all of the Open Source projects chose to roll their own SASL implementations which only supported those authentication mechanisms that were mandatory in the standard.

6.6 Web services

Web delivered services seem to be the primary growth area for new applications. Unfortunately, they're some of the hardest services to add Kerberos based authentication to, due to the one-shot nature of the original HTTP protocol, and the proprietary nature of many web browsers. Broadly speaking there are 4 major techniques available, all of which have their drawbacks.

6.6.1 Kerberos-SSL

RFC2712[13] is an internet standard proposing a method of using Kerberos as part of the SSL connection establishment handshake, as a replacement for X509 certificates. There is an implementation of this in OpenSSL, but other than that it is not widely deployed. We implemented code for Mozilla's NSS library to also support this[14], which the OpenSSL code was originally implemented and tested against. Unfortunately the NSS code was never integrated and has now significantly bitrotted.

RFC2712 suffers from a number of major issues. As a standard, it does not contain sufficient implementation information to enable developers to produce interoperable code without additional, out of band, agreements. It only allows the use of single DES based encryption, and requires implementations to have detailed internal knowledge of Kerberos's tickets, rather than simply being able to pass those to a library through a well-defined API.

It also does not support credential delegation, which is of significant value when developing web services which front other Kerberized systems. A perfect example of this is webmail, where a web front end passes user requests on to an IMAP or POP server. This would ideally use the Kerberos credentials the user presented to the web server to authenticate the IMAP request.

6.6.2 KX509

KX509[15] was an initiative from the University of Michigan to allow commercial-off-the-shelf web browsers to perform Kerberos authentication. It works by using the user's Kerberos credentials to obtain an X509 client certificate which can then be used as part of the HTTPS handshake to authenticate the user to the web server. It requires a client application (to obtain the certificate), and, in some cases, a browser plugin (to make the certificate available to the web browser).

We've been using KX509 for web authentication for a number of years. Whilst it works well when being used from local, managed, systems, getting it to work for the remote, web-cafe, case is difficult. Local attempts to develop a portal solution which provides client certificates over the web, rather than requiring installed applications, have hit upon browser usability and credential destruction issues which make the portal system unusable for the inexperienced user.

KX509 has the advantage that it can be generalised to any protocol which uses TLS as its security layer. We also use it to secure our OpenVPN based VPN solution, and have considered it as a possible solution to the CUPS authentication issue.

KX509 does not support credential delegation, unless an additional daemon (kct) is run on the KDC. The version of this daemon supplied by the University of Michigan has a number of issues - whilst we have made local changes to make the code suitable for production use, we have yet to deploy it, and at this point we are unlikely to do so.

6.6.3 NegotiateAuth

NegotiateAuth is the colloquial name given to the protocol originally deployed by Microsoft to provide single sign-on between Internet Explorer and IIS. The protocol was extended by them to provide Kerberos support, and is documented in a (now expired) Internet Draft[4].

Implementations of this protocol have recently appeared in a number of browsers, and modules to implement it are also available for Apache[16]. The protocol just performs an initial Kerberos handshake and provides no connection security, and so must be run over HTTPS to prevent man-in-the-middle attacks.

NegotiateAuth requires some browser configuration in order to work properly, as most browsers contain a 'white-list' of sites which support it. In addition, it requires that the machine it is running from have Kerberos support installed and configured. For these reasons, it isn't suitable for use in the web cafe case. However, as a native mechanism for providing Kerberos authentication without requiring any clumsy interface code on client, or server, NegotiateAuth has much to commended it.

6.6.4 Cookies and Cosign

There are a number of technologies available which use browser cookies to provide web-based single sign-on capabilities. These vary greatly in their security and in their ability to interface with existing Kerberos infrastructures. After examining a number of these technologies, cosign appeared the closest fit to our requirements, and is also in use by other areas of the University.

Cosign[17] is another technology from the University of Michigan. It integrates with Kerberos on the server side, and the complete system can be used to delegate Kerberos credentials to web applications by means of a cosign daemon, running on a secure machine. Cosign has the disadvantage that it requires users to authenticate to it by typing in their username and password on a secure web page. This is perfect for the web cafe case, but defeats the single sign-on objective when the user has already authenticated to their workstation.

It is our intention to replace our current KX509 based infrastructure with one that uses Cosign for service authentication. However, we intend to supplement the username and password initial authentication scheme with one that supports NegotiateAuth from suitably configured browsers. We believe that this will give us a suitable trade off preserving single sign-on, whilst also allowing use by users who are in web-cafe style situations.

6.6.5 Deployed services

We are currently running Bugzilla, RT, TWiki, Mailman, and a number of locally developed web services, all of which use KX509 for authentication. These all use the enterprise mode of web authentication, where the authentication is performed by the server rather than the application, and the application is informed of the user connected to it by means of environment variables. We've implemented the authentication stage within the sever by means of a locally developed Apache module[18], which checks the certificate, sets a number of environment variables based on its contents, and provides a success or failure indication to the server in the same way as a standard Apache authentication module.

Most of the applications we are running support this form of authentication through configuration changes. However, local patches to Mailman were required.

7 User experience

One of the key issues when moving from a system with lax authentication to one which tightens this up is user perception. In many cases, a higher level of security can increase user frustration as they start having to repeatedly authenticate, rather than just being allowed access. Implemented correctly, single sign-on can actually avoid many of these pitfalls. Instead of having to remember multiple passwords for multiple services, the user has one password, which they only have to enter once at the start of the working day. After that, all service authentication is handled transparently.

The main problem is one of credential expiry. In our configuration, Kerberos credentials have a lifetime of 18 hours. Once that lifetime is exceeded the user has to re-authenticate to the machine before they can access any Kerberized services. This limited lifetime is often seen as an inconvenience by users who wish to be able to leave long running processes and sessions operating, without having to re-authenticate repeatedly. Currently, this only affects a small group of users who are mainly doing interesting things with fetchmail and our mail service. However, we anticipate this becoming a far larger problem with our forthcoming move to a secure filesystem.

We have put effort into ensuring that the authentication step is as seamless possible from the user's perspective. All of our authentication is performed through PAM, which is used to chain authentication modules together, so that a single PAM authentication will get all of the tokens that a user requires to access our services in one operation. We have replaced the standard kinit command used for renewing tickets with a local version called renc. renc uses a PAM stack to determine which authentication modules to use. Immediately after obtaining a user's Kerberos ticket renc uses this to obtain tokens for any other authentication services in use. In our case, it currently obtains KX509 certificates and AFS tokens.

We also use this approach with our screensavers. If a user is running a locking screensaver which uses PAM to check the password when it is unlocked, the password validation step will also renew all of the user's credentials. In this way, users who lock their screens when leaving their computer overnight may never actually see their credentials expire.

8 Future

As mentioned above, we intend to replace our current KX509 based web authentication service with a hybrid solution using NegotiateAuth and Cosign. The KX509 solution is probably the most significant issue with our currently deployed service. Whilst it works perfectly for managed, local machines, we've never managed to adapt it work well with unmanaged machines whose proliferation was not anticipated in the original system design. That said, KX509 will continue to be provided as a service, as we have a requirement on it to authentication those protocols which use TLS/SSL but not HTTP, such as our OpenVPN system.

The other major avenue of current development is secure filesystems. One of the original aims of DICE was to get rid of a machine-based trust model. However, the continued proliferation of NFS has made this impossible. We began an effort last year to examine alternatives to NFS, and looked in detail at NFS v4 and AFS, both of which support secure authentication. We've decided to deploy AFS, and are in the process of a gradual pilot and rollout of this service. It's far to early to say anything concrete about our experiences here, but early indications suggest that the usability and user education issues of a kerberized file system will be far more difficult to deal with than any of the usability points noted above.

As noted, a secure filesystem will cause significant issues with credential expiry. Our proposed solution is based around Kerberos's concept of instances. We intend to allow users to have multiple Kerberos identities, to which they can assign a subset of the authorisation privileges of their main account. This would allow, for instance, a user to set up a Kerberos identity of user/mail to do long term mail processing actions, or user/myfileaccess to allow them long term access to particular files. Key material for these identities would then be stored on the local disk of the machine running the long-term proccess. We have yet to implement this solution, although it is seen as an essential part of our secure filesystem rollout.

This work is part of longer term thinking on ways of dividing up the access that is given to an identity into more manageable chunks. One of the problems with any single sign-on scheme is that there is one secret (the user's password) that becomes all powerful. As technology advances, these passwords are becoming increasingly vulnerable despite all efforts taken to protect them. Users using untrusted clients are always at the risk of trojan horses, and even trusted machines are now at risk from the hardware key logging technologies that are readily available.

We have already discussed the two-tier system of splitting access rights between normal and administrator usage that we are currently using. However, this is not sufficiently flexible to deal with the increasing diversification of access methods. It is possible that a user may want to have a password that they can purely use for fetching mail from an insecure location, without allowing someone who captures that password to access everything in their account. We have tentative plans to develop both our authentication and authorisation environments to support multiple tiers of access, which can be controlled by the user. We're also very interested in exploring the options of hardware token based authentication for more critical activities.

9 Source Code

We try to be as proactive as possible in returning the enhancements we make back into the upstream code. In general, all of the locally written modifications detailed in this paper should now be in either shipping versions, or the code repositories, of the relevant packages. The exceptions to this, and those for which code has yet to be released, are as noted below. Please feel free to contact the author if you have trouble finding any of the relevant code.

AdiumX	Kerberos support is only available through patches[12].
Gaim	Kerberos/GSSAPI SASL support will be in the forthcoming 2.0.0 release
Jabberd2	Kerberos support is not in the current 2.0 series, but is in CVS and will be in a future 2.1 release.
Mailman	Enterprise web authentication support is only available through patches
OpenSSH	User authentication support is in all releases, key exchange support may be included, depending on your vendor. Patches available[9].
Psi	Kerberos support is only available through patches[12].
Thunderbird	Improvements to Thunderbirds GSSAPI support to enable it to work on platforms without development libraries installed are currently in their CVS and should be in a future release

10 Looking back

Kerberos has become far more widespread over the intervening years than we ever expected when we started the project. This, and the introduction of service location to the core Kerberos libraries, mean that the amount of configuration required on a client before it can use our service is minimal. Most Unix distributions now ship with Kerberos support as standard, and build their applications with Kerberos enabled. This has enabled us to cope with a shift towards non-managed clients accessing our services. It is now the case that most Unix and Mac OS clients can use our network services out of the box, with little or no configuration required on the part of the user.

By and large, the roll out was surprisingly painless. Whilst some effort has been required to ensure that new services fit into the existing authentication framework, the growing ubiquity of Kerberos has meant that this work has become increasingly easy. Some users continue to have problems with credential expiry, and finding solutions to this will become increasingly pressing as we move towards a secure filesystem.

Deploying Kerberos whilst unifying computing infrastructures gave us a number of benefits. We didn't have to deal with password migration, as we gave users new passwords for the new DICE infrastructure. The gradual rollout of the new system allowed us to migrate both clients and services in an orderly fashion, without requiring any big-bang changes.

References

[1] B. Clifford Neuman and Theodore Ts'o *Kerberos: An Authentication Service for Computer Networks* IEEE Communications, 32(9):33-38 September 2004

[2] *Heimdal* http://www.pdc.kth.se/heimdal/

[3] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, V. Welch. *A National-Scale Authentication Infrastructure* IEEE Computer, 33(12):60-66, 2000. http://www.globus.org/alliance/publications/papers/butler.pdf

[4] J. Brezak *HTTP Authentication: SPNEGO Access Authentication* Expired Internet Draft http://www.kerberosprotocols.org/index.php/Draft-brezak-spnego-http-03.txt

[5] Paul Anderson and Alastair Scobie *LCFG: The Next Generation* UKUUG Winter Conference 2002 http://www.lcfg.org/doc/ukuug2002.pdf

[6] Simon Wilkinson *Simple X509 Key Transfer Service* Source code repository http://cvs.inf.ed.ac.uk/cgi-bin/cvsweb.cgi/sixkts/

[7] PADL Software *pam_ccreds* Web page http://www.padl.com/OSS/pam_ccreds.html

[8] J. Hutzelman, J.Salowey, J.Galbraith, V.Welch *GSSAPI Authentication and Key Exchange for the Secure Shell Protocol* Internet Draft (work in progress) http://www.ietf.org/internet-drafts/draft-ietf-secsh-gsskeyex-10.txt

[9] Simon Wilkinson *GSSAPI/Kerberos support in OpenSSH* Software http://www.sxw.org.uk/computing/patches/openssh.html

[10] *GSSAPI SASL support in Thunderbird* https://bugzilla.mozilla.org/show_bug.cgi?id=303160

[11] *Kerberos support in CUPS* http://www.cups.org/str.php?L646

[12] Simon Wilkinson *GSSAPI/Kerberos support for Jabber* Software http://www.sxw.org.uk/computing/patches/jabber.html

[13] A. Medvinsky, M. Hur *Addition of Kerberos Cipher Suites to Transport Layer Security* RFC2712 http://www.ietf.org/rfc/rfc2712.txt

[14] *RFC2712 support in Mozilla's NSS library* https://bugzilla.mozilla.org/show_bug.cgi?id=61932

[15] Olga Kornievskaia, Peter Honeyman, Bill Doster and Kevin Coffman *Kerberized Credential Translation: A Solution to Web Access Control* Usenix Security Symposium, Washington D.C. http://www.citi.umich.edu/techreports/reports/citi-tr-01-5.pdf

[16] *Kerberos Module for Apache* Software http://modauthkerb.sourceforge.net/

[17] *Cosign: Web Single Sign-on* http://www.umich.edu/ umweb/software/cosign/

[18] *mod_authssl* Apache Module http://cvs.inf.ed.ac.uk/cgi-bin/cvsweb.cgi/mod_authssl/