Pp. 65-74 of the *Proceedings of LISA ’02: Sixteenth Systems Administration Conference* (Berkeley, CA: USENIX Association, 2002).

Embracing and Extending Windows 2000

Jon Finke – Rensselaer Polytechnic Institute

ABSTRACT

We were recently presented with the challenge of deploying a large scale Windows 2000 environment, initially for the Administration Division, but eventually including academic and other users. Rather than try to eventually re-integrate independently administered domains, we took this as an opportunity to develop the tools and resources to provide a campus-wide Windows 2000 environment that is well integrated with the existing enterprise information and computing systems, much like we integrated our Unix systems. This would automate many of the mundane administrative functions, yet provide appropriate delegation of control to departmental administrators as needed. This paper describes the systems we developed to make this happen.

Introduction

Rensselaer recently embarked on a major building initiative: a BioTech research center, an electronic media and performing arts center, a new central boiler and chiller plant, a parking garage, and a new campus entrance, and with this new construction come some new requirements for information sharing and archiving. The Administration division decided to handle this with a Windows 2000 Exchange email system. The good news is that they came to the Chief Information Officer to ask for help. The bad news is that the CIO agreed to help.

With this new initiative, we thought that it was very important to get this new system deployment right “the first time,” as attempting to go back later and fix things would be very difficult. We also wanted to look beyond the requirements of this specific project, and deploy a campus-wide solution to integrate support of academic programs as well as other administrative units. Our existing Windows administrators were spread over a number of administrative units, as well as a few academic departments. It was important to come up with a system that they would be comfortable with, and one where they would be willing to “turn over control” to the central computing center. This was quite a change in approach for our Windows administrators.[1]

At Rensselaer, we have a long history of automating our Unix systems administration tasks (via an Oracle database) and striving to get information from an authoritative source. Figure 1 gives a high level view of the flow of people-related information at Rensselaer. Human Resources and student record data flows from the administrative system running SCT/Banner into the Simon system’s [4, 5] Oracle database. One of the things this is used for, is to automatically create and expire Unix/email user accounts. From Simon, the relevant information is sent to each of the various client information and authentication systems, including the central telephone directory (LDAP, ph and web based), the ID card system, AFS and Kerberos, and the new Windows 2000 domain and Exchange email system.

In addition, we have developed many tools and techniques that allow us to delegate responsibility with a great deal of fine-grained control. For example, our telephone directory system [6] allows a person from each department to maintain directory information for their staff. Our eventual proposal was actually built on top of several existing systems, with some enhancements and extensions.

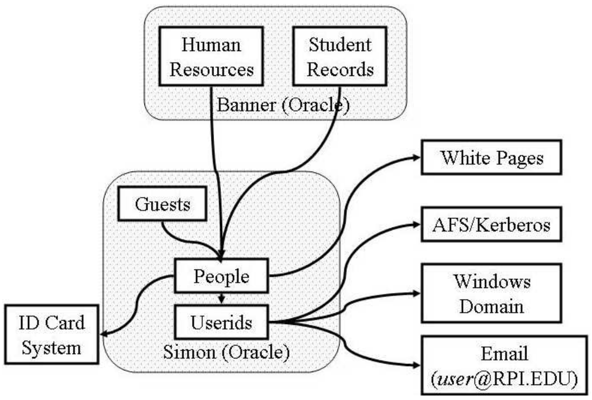

Figure 1: Information flow at Rensselaer.

One of our existing projects was to provide a comprehensive LDAP [9] directory service, and make this the directory service of choice for information consumers. To this end, the data needs to be both accurate as well as timely. Since it appeared that our LDAP service would be driving our Active Directory service, this project gained a key role in our Windows 2000 project.

Another existing project was single signon, or at least some form of password synchronization between systems. While we may have a bunch of different systems behind the scenes, our users think that they have a single account and password and we provide the magic to make it all work. To this end, we wrote a

[1]During some training on MS Exchange, we learned that Microsoft invented Kerberos.

web-based password changing tool. Unlike the one developed at Auburn University [10], we use public key encryption to protect and store the passwords; and we use a relational database to manage and store both the public keys and the encrypted passwords, as well as to drive the actual password changing process on Windows, Oracle and Kerberos.

In our original Windows 2000 proposal, the organization hierarchy would be based on the University structure, as defined in the telephone directory. However, after meeting with some of the Windows administrators, we saw that we needed a way to allow these folks to create their own structure, yet keep them from interfering with other administrators. Fortunately, our existing telephone directory model provided a good fit in both respects, and we decided to extend it to handle the Windows domain.

The Institutional Layer

While there are many technical challenges to implementing a system like this, institutional politics add a number of other "opportunities to excel." In our case, we were fortunate that we had addressed many of these in the past, having long established the practice of feeding data from Human Resources and the Registrar into the Simon system to automatically create Unix and email accounts for everyone on campus. This same system was grown to manage the campus telephone directory and feed the campus ID card system. With these projects in place, we had established our credibility in delivering campus-wide information services.

One thing that helped us a great deal was always insisting on going to the authoritative source for all data elements. All student information comes from the Registrar, employee information from Human Resources, and so on. If we have changes to that data, we pass it back to those systems. Because we are not trying to maintain our own "student database" or "employee database," we can insist that the "owners" of the data share it with us. We have been able to sweeten the pot in many cases by providing some services to them based on their data.

As information systems evolve, we sometimes move away from this ideal because user demands may outpace the ability of support organizations to react to changes. As a result, we have discovered that several important data elements that should have been maintained centrally had moved into the Simon system.[2] We now have an ongoing task force monitoring the relationship between Simon and the rest of the administrative information systems.

Care and Feeding of LDAP

We begin by describing the development of our existing LDAP-related information flow, which provides the base upon which the Windows implementation is built. We needed a way to detect changes to data in our enterprise information systems and propagate just those changed records to LDAP. Since we wanted these changes to be as close to "real time" as possible, the process needed to be very lightweight. By making the load on the enterprise database[3] very small, we can make frequent checks for changes without annoying the DBAs or impacting performance on that system.

To make things more interesting, we had to do this in such a way to make a minimal impact on the keepers of the enterprise database. These MIS folks have a great many demands on their time, and are often not available to take a large role in the deployment of new services. This necessitated that our approach require very little involvement of the DBAs and developers in the MIS department.[4]

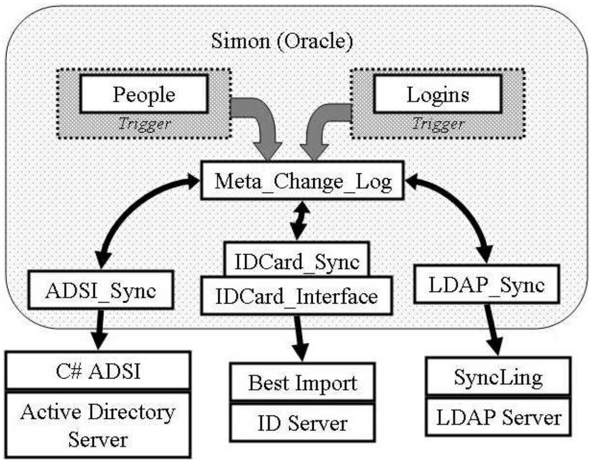

Figure 2: Information change flow architecture.

In our initial LDAP deployment, we were populating a general "people" directory following the "eduPerson"[5] [3] to replace our PH white pages directory server. A second LDAP project was to provide a "POSIX" account space to provide /etc/passwd to our Unix hosts. We did this by generating LDIF files using a PL/SQL package [13][6] and some interface code to generate these into files [7]. While generating LDIF files to test the LDAP servers was okay, this was not going to handle the ongoing updates that we needed.

Triggers and Change Queues

A powerful feature of Oracle and other high end databases is the ability to define triggers [1] that will

[2] If you are publishing the enterprise phone directory, you WILL get good data feeds, and the rest of your projects can benefit.

[3] We run SCT/Banner for our student records, finance, and payroll.

[4] While this is a local condition, I suspect over-booked MIS shops are a fact of life at many other sites.

[5] This is an effort by EDUCAUSE to come up with a set of standard attributes for a directory entries at educational sites.

[6] PL/SQL Packages are a collection of functions, procedures, cursors, and variables. These can be both public and private. Access to the public procedures can be granted to Oracle users or roles. Once accessed in a session, a package maintains state between calls.

automatically execute when a particular table is changed in some way. This gives us a way to insert our own business rules in the general operation of the database. For example, our code will get executed when a person is added to the people table. What we did is record key information about the change into a queue table that would be processed later. Our intention here was to place some of these triggers into our main administrative database, SCT/Banner. In this way, we would not need to change the vendor code, which should make handling new releases much easier. For applications we write ourselves, we have the option of using triggers or putting these calls directly in the applications themselves.

In Figure 2, we have triggers "watching" for changes to the Logins and the People tables. When a change in a table is detected, we write a record into the Meta_Change_Log table (see Table 1). We include a number of identifying fields. The first three – Tname, Subtype and Change_Type – identify which table, and in some cases, which part of the table was changed and how it was changed. We also have a number of fields to identify which entry was changed. Since many of the changes we are interested in involve people, we include the primary keys used by each of our two main information systems. However, we may be looking for changes to something other than people (department names, group membership, etc.) so we have a generic character and numeric key available for tables dealing with non "people" information. The exact set of identifiers used depends on the table.

The second half of the Meta_Change_Log (see Table 2) deals with how we process these records and manage the queue. Each entry has an entry date and a sequence number to assist with ordering. We are feeding several systems with changes: our LDAP directory server, our Windows 2000 Active Directory Server and our Photo ID Card system. We need to be able to process each of these queues independently, since they may be operating with different schedules for backup, maintenance, etc. All of these queues has a "Process Needed" flag and a process date. When a record is inserted into the queue, the appropriate "Process Needed" flags are set to "Y".

Each of the _Proc flags has an index on it. When a record is processed, this flag is set to Null. Oracle do not index Null values, so that only the active (change pending) records are included in the index. This means that, in normal operation, these indexes are very small and can be accessed and updated very quickly. This keeps the load on the database to a minimum and allows us to make frequent checks to look for changed records.

Name	Type	Size	Description
Tname	Varchar2	32	Identifies the table that was changed
Subtype	Varchar2	32	Optional subtype to allow specialized processing based on which fields in a table were changed.
Rrowid	Rowid		The Rowid (Oracle record identifier) of the row that was changed. This allows for direct access to the desired row with no searching needed.
Change_Type	varchar2	1	A flag indicating Insert, Modify or Delete of the record.
PIDM	Number		Person identifier for our administrative system (SCT/Banner)
Person_Id	Number		Person identifier for the Simon system.
Pkey_String	Varchar2	32	A table specific character string to identify the record in question (such as a Unix account name).
Pkey_Number	Number		A table specific numeric value to identify the record in question (such as a Unix UID).

Table 1: Meta_Change_Log Table – Identification.

Name	Type	Size	Description
Entry_Date	Date		The time and date when the record was entered.
Entry_Number	Number		An ever increasing sequence number.
LDAP_Proc_Date	Date		The date when the LDAP SyncLing processed this record.
LDAP_Proc	Varchar2	1	A flag set to "Y" when this record is awaiting processing by LDAP.
ADSI_Proc_Date	Date		The date when the ADSI synch program processed this record.
ADSI_Proc	Varchar2	1	A flag set to "Y" when this record is awaiting processing by ADSI.
IDCARD_Proc_Date	Date		The date when the IDCARD synch program processed this record.
IDCARD_Proc	Varchar2	1	A flag set to "Y" when this record is awaiting processing by IDCARD.

Table 2: Meta_Change_Log Table – Queue processing.

SyncLings

Now that we had a list of changes that needed processing, we also needed a way to get them into LDAP. To this end, we wrote an Oracle package called LDAP_Sync. This package references the Meta_Change_Log table and provides two procedures, Get_Changes and Ack_Change. The LDAP_SYNC package is called by a Java program called a "SyncLing." It calls the Get_Changes routine to get the next change, processes it, and then marks that change as done with the Ack_Change routine.

The Get_Changes routine (see Table 3) is called, and the next LDIF record [8][7] is returned. This is then compared to the existing record in the LDAP server by the SyncLing, and the appropriate action is taken. As each record is processed, the Ack_Change routine is called with the Rec_Id. This will mark the record as processed (clearing the LDAP_Proc flag in the table). This process is repeated until Get_Changes returns a null record. At this point, the SyncLing pauses for a few seconds, and then resume asking.

As part of the process of the initial data load of the LDAP server, we had already written a package to generate LDIF records for each person at Rensselaer from the directory database, so it was a trivial matter to call this routine from the Get_Changes routine.

We are actually populating several "trees" in our LDAP database: a general people tree, and a POSIX account tree (in part, to replace our /etc/passwd file and some group files). There are corresponding routines to generate flat LDIF files that we can call for each of these cases. The type field identifies the type of LDIF record being returned, so that the processing program knows what to do. It can also ask for just records of a given type, or it can ask for all types of records.

This system works very well for our Unix systems for which it is deployed. In the next section, we discuss how we extended it to service our Windows 2000 systems.

Feeding of Active Directory

Our initial plan was to use our LDAP server to feed the Active Directory[8] server. However, this ran into a few problems. Although we were feeding both a POSIX user id base and the general person/directory information into LDAP in essentially real time, the data did not match well with the requirements of Active Directory and our Exchange server. Although we were able to load people into Active Directory via LDAP, we ran into limitations due to our focus on the "eduPerson"; fields we needed in Active Directory were not available.

The second problem is that we needed to propagate password changes from our general system into the Windows 2000 world. Our password changing scheme (see section below) relies on public key encryption to secure the passwords while in transit; implementing this via LDAP was not practical.

We had recognized early in the project that LDAP was not going to be able to handle the password changing, so we had started a second project to manage the passwords. At that time, Microsoft was moving to the use of JAVA in some aspects of their systems, so we started development of a JAVA program that would talk to the database; get the password changes, and then apply them to Active Directory. When it became clear that our LDAP approach was not going to handle our needs, we expanded the role of the password propagation to a more general Active Directory update system.

The database side of the LDAP solution described above was exactly what we needed, so we simply cloned it. In fact used the same table, and simply added an ADSI (Active Directory Service Interface) [11] propagation flag to duplicate the LDAP propagation flag, and duplicated the PL/SQL package to provide ADSI with its own interface to the database. This allows for ADSI and LDAP to run in parallel.

The ADSI_Sync package starts out much like the LDAP_Sync package with a Get_Changes and Ack_Change routine, and they operate in the same way as described above. However, rather than returning an LDIF record, we just return the username. We have also added two more routines, Get_Dirinfo (see Table 4) and Get_Dirinfo2 (see Table 5). The rollout of this service was under some pretty tight time pressure, which is why these are two separate routines, rather than just one.

As the Windows 2000 service was being rolled out, and more administrative users were being added to our exchange server, some other issues were discovered. We were already using the Windows 2000 user base and password for students to access our

[7]One method of loading data into an LDAP server is using a format known as the "LDAP Data Interchange Format" or LDIF.

[8]Active Directory is the database server used by Windows 2000 to hold user information and passwords.

Name	Type		Description
RType	Varchar2	In/Out	On call, specifies the type of records desired, and on return, indicates the type of record being returned.
Rec_Id	Varchar2	Out	Returns a record id used for the Ack_Changes routine.
LDIF_Record	Varchar2	Out	An LDIF formatted record with the information for the next person to be processed.

Table 3: Get_Changes procedure parameters.

public workstation labs, so the password changing system was being well exercised, but as administrative users were moving their email from our Unix-based pop mail service to the Exchange server, we needed to feed more information into Active Directory.

One of the other objectives of this project was to provide Exchange based mailing lists for all the people in a given department or division. Our first pass at this was to provide a pair of routines, Get_Divisions and Username_By_Division; the first would return a list of all of the divisions, and the second would return a list of everyone in the specified division. In the same way, we also provided Get_Departments and Username_By_Department routines. These would be used to maintain the desired membership lists. However, we quickly ran into a problem with the names being used; different aspects of the Active Directory service took exception to some of the special characters we were using such as ampersand, dash, and a few others. To handle this, we added a CLEAN flag to the Get_D... procedures and a Get_Username_By_Clean_D... call.

Once our users had tasted the wonders of automatically maintained mailing lists, they were hungry for more. As part of a different project, we had put together some special mailing lists for our ListProc machine such as "Deans, Directors and Department Heads" and "Building Coordinators." These lists were being dumped as flat files using our Generate_File program. A little bit of creative coding, and these lists became available to ADSI via the Get_Specials and Get_Username_By_Special.

To be Sharp, You Must C Sharp

When we first started working on a JAVA program to change passwords, it appeared that Microsoft would provide (JAVA) Class libraries which exposed their ADSI routines. As it turns out, they ended up abandoning their JAVA efforts, and we were forced to provide our own JAVA callable ADSI routines by using JNI (Java Native Interface) to wrap a Windows DLL written in C. Although it worked, it was very difficult to change, and our development staff dreaded new requirements from the Windows team. Every new function would take 30 minutes to write, and then four days to debug and tweak so it would actually work.

Microsoft's new direction was a programming language called C# [12], part of their Visual Studio package. With C#, Microsoft has provided the ADSI class libraries we need, allowing us to eliminate the need for a custom DLL. This has proven much easier to work with, and we have resumed adding new fields and streams into Active Directory from our central database. This shows up as the C# ADSI box in Figure 2.

Making Changes to Passwords; Here, There and Everywhere

When we first rolled out our campus-wide Unix service, built on top of an AFS filestore, we wrote a replacement for the general Unix passwd program that would update the Kerberos password and save a copy of the Unix PASSWORD crypt in our central database. This enabled us to build conventional /etc/passwd files for a few legacy systems that did not use Kerberos (this was 10 years ago).

As our systems evolved, many of our users moved away from the Unix workstations for their computing, but continued to use their Kerberos passwords for email, printing, dial-up and other services. Instructing people to connect to a Unix machine (using ssh, not telnet!) sign on, and invoke the passwd program to change their password resulted in a lot of frustration for both our users and our help desk staff.

Name	Type		Description
Uname	Varchar2	In	Target username – provided by Get_Changes or other routines.
Pref_Email	Varchar2	Out	Preferred email address.
Camp_Phone	Varchar2	Out	Campus telephone number.
Camp_Fax	Varchar2	Out	Campus fax number.
Camp_Address	Varchar2	Out	Campus address.
Department	Varchar2	Out	Department Name.
Division	Varchar2	Out	Division Name.
Web_Page	Varchar2	Out	URL of person's web page, if available.
Title	Varchar2	Out	Title of person (employees only – not students.)

Table 4: Get_Dirinfo procedure parameters.

Name	Type		Description
Uname	Varchar2	In	Target username – provided by Get_Changes or other routines.
Last_Name	Varchar2	Out	Person's last name.
First_Names	Varchar2	Out	Person's first and middle names.
Preferred_First_Name	Varchar2	Out	Alternate first name preferred by the person. Used in the directory.

Table 5: Get_Dirinfo2 procedure parameters.

A web based password changing system was an obvious solution.

Although we had long ago stopped collecting a Unix crypt, we still liked the idea of collecting passwords for use on other systems, such as a trouble ticketing system (Oracle based), our academic Oracle servers, and our Kerberos 5 server. We were not comfortable with the idea of storing, or even transmitting, clear text passwords. While we could protect the web session with SSL, we still had the traffic from the web server to the database server. So we decided to use public key encryption, encrypt the plain text password with a public key on the web server, and transmit the encrypted text to the database for processing and storage. Since we were already connecting to the database for other reasons, this was a logical spot to store the public keys. As part of this process, the password change web page also signals (via Oracle signals) [2] the back end password processing.

Using Oracle to broker the password changes makes it much easier to add new authentication services. The web page used by the users does not need to change, nor does it need to understand every new type of authentication system. To add a new authentication system, we just need to write a new back end that understands that world, along with some public key encryption and database access. It also allows for people to request password changes when some authentication services are not available; the change is held until the service is restored.

In Figure 3, we have data flow for a password change. We start with a web page, connecting to our secure web server via an SSL connection (step 1). The secure web server does some password strength checks, and, if things are okay, makes an immediate password change to our AFS Kerberos server (step K). There is also an option to simply test a new password to ensure that it will pass the strength tests.

Once the password change is checked out, the web server calls an packaged routine in the database, Get_Public_Key (step 2). This public key is used to encrypt the clear text of the user's new password, and then that encrypted text is stored back in the database (step 3) via the Store_Pw procedure. This stores the encrypted text, the user name, and the key number in the Encrypted_Passwd_Cache table (Table 6).

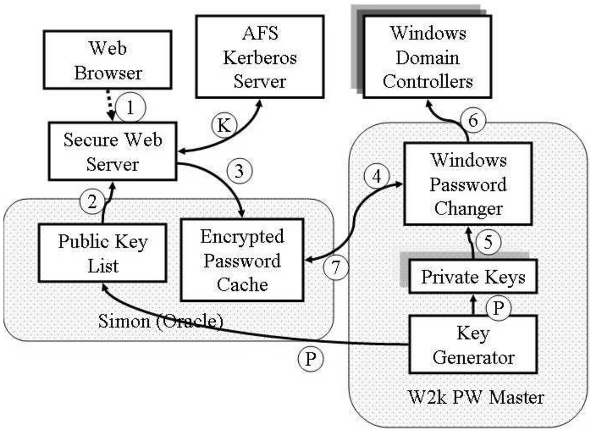

Figure 3: Password change flow.

Using the same sort of propagation polling that we used for the ADSI_Sync process, we have Windows Password Changer program running on our Windows 2000 Password Master machine. It looks for entries in the Encrypted_Passwd_Cache table (step 4) that have Windows_Prop_Pending set to "Y." Along with the principal name, and the encrypted password, it also gets a key number. It then consults it's own list of private keys (step 5) and uses that to decrypt the

Name	Type	Size	Description
Principal_Name	Varchar2	32	The user name
Encrypted_Key	Varchar2	1024	The encrypted user password.
Key_Num	Number		The key number used to encrypt the password.
Entry_Date	Date		The time and date of the change request.
Simon_Prop_Pending	Varchar2	1	A flag indicating that this password needs to go to Simon.
Simon_Prop_Date	Date		The time and date when this change was given to Simon.
Simon_Err_Code	Number		A numeric error code from this operation.
Applix_Prop_Pending	Varchar2	1	A flag indicating that this password needs to go to Applix.
Applix_Prop_Date	Date		The time and date when this change was given to Applix.
Applix_Err_Code	Number		A numeric error code from this operation.
Kerb5_Prop_Pending	Varchar2	1	A flag indicating that this password needs to go to Kerberos version 5 server.
Kerb5_Prop_Date	Date		The time and date when this change was given to the Kerberos version 5 server
Kerb5_Err_Code	Number		A numeric error code from this operation.
Windows_Prop_Pending	Varchar2	1	A flag indicating that this password needs to go to Windows
Windows_Prop_Date	Date		The time and date when this change was given to Windows.
Windows_Err_Code	Number		A numeric error code from this operation.

Table 6: Encrypted_Passwd_Cache table.

password and make the change on the Windows 2000 domain controller (step 6). Once that has been done, it then updates the Windows_Prop_Pending flag (step 7), as well as the Windows_Prop_Date field. It continues getting more rows to process until there are no more to be done. At that point, it will "sleep" for 30 seconds, and then check again. While this seems pretty quick, there is some delay in propagating the password changes between the Windows Domain Controllers, so it can take up to 15 minutes for the changes to get passed among the Windows servers. We have not found any ways in which to improve this.

Key Generation

There are a number of options and approaches to managing the public/private key pairs. For our Windows 2000 world, we have a key generator program that we run on the Windows 2000 Password Master machine which generates a key pair. The private key is stored in a secure location on the Password Master machine (and yes, we really need to keep this machine secure), and the public key is stored on the Oracle database server. This makes it available to the Secure Web Server when needed. The public keys are stored in the Passwd_Key_List table (Table 7).

We have not spent a lot of time working on our key management procedures, but we periodically generate new key pairs and remove the old private keys from service. The use of key numbers and types help keep things in order for us. However, since all of the applications (the secure web server and the back end processors) all rely on the Oracle procedures for key management, we can change those routines and the external code will do the right thing without any changes.

Multiple Streams

In the case discussed above, we are keeping two passwords in sync: our AFS Kerberos password and the one for our Windows 2000 server. In actuality, we are also keeping the Oracle password on the Simon server, as well as the Oracle password on our trouble ticketing system (Applix). We are maintaining our Kerberos version 5 server passwords via this approach.

When the secure web server encrypts a password, it actually does it twice, once with a key used for the Windows Password Changer, and a second time using a different key pair used by the Simon system. So, each password change results in two entries being made in the Encrypted_Passwd_Cache table. When the Windows key is used, the Windows_Prop_Pending flag is set, and when the Simon key is used, the Simon_Prop_Pending, Applix_Prop_Pending and Kerb5_ Prop_Pending flags are set. Since these three queues are all processed on the Simon server machine, they are able to share the same public and private keys.

At present, our Kerberos 5 server is not yet in production. By using its own propagation flag, changes to other servers can still go through when the Kerberos 5 server is not available. When service is restored, it can catch up on the changes. We also have the ability to re-encrypt the clear text to feed to other systems. This allows us to keep most of the processing on the back end server, rather than making additional encryption runs on the secure web server.

Timing Issues

Although we would like all of this to take place instantly, the back end program is doing decryption and re-encryption, as well as accessing the database. In addition, the program is actually run by another job scheduling system that is sometimes busy with other functions (creating accounts, changing quotas, etc.). We also have the problem that the system we are trying to update is sometimes down or unreachable, so the password change is stuck waiting in the queue. This was starting to result in some operational problems where a person would change their password, and then immediately try to access a service and get an authentication failure.

To assist our help desk (and advanced users), we wrote another web page which can display the exact date and time of a person's last password change, and when it was applied to each authentication base. It will also display if there is a change pending. This also lets us generate some statistics on the time it takes for password changes to propagate. The page will first report on any pending changes, and then report on the most recent set of completed changes. It produces a report like Table 8.

Name	Type	Size	Description
Key_Num	Number		The key number.
PK_N	Varchar2	2048	Part of the public key.
PK_E	Varchar2	32	The other part of the public key.
Active	Varchar2	4	A flag indicating that this key is the active key for this type.
Start_Date	Date		When a key goes into service.
End_Date	Date		When a key is removed from service.
Create_Date	Date		When a key was created.
Key_Type	Varchar2	4	What "TYPE" of key this – used to identify who has the private key that matches this key.

Table 7: Passwd_Key_List table.

There are a couple of oddball entries. For example, the Kerb 4 change always has an elapsed time of 0 seconds: if it fails, none of the Oracle processing will take place, and no logs will be written for display. Of course, the user will get a very clear error message on the web page when this happens. The other odd case is the AdminReq. This is a case where the user lost their password and went to the help desk and asked that their password be reset. This system uses a similar mechanism to the ones discussed here, and sometimes it is subject to delays.

Other password issues

For long term storage of ''clear-text'' passwords, in order to let us populate services yet to be deployed, we have the option of storing the appropriate private keys on secure storage (such as a floppy disk locked up in safe), and when we need to load the initial population of a new service, we get the floppy, restore the private keys, and load the new system.

Systems Administration as a White Pages problem

At LISA X, I presented a paper [6] showing how managing the University white pages (phone directory) was really a Systems Administration problem. Things have now come full circle, and we are applying our telephone directory tools and techniques to managing our Windows 2000 domains.

For our telephone directory, we take a data feed from Human Resources, decide (based on status and department) if they are included in the directory, or if they need to be moved to another part of the directory tree.[9] We also have facilities available to manually move someone to a different department, have someone appear in a second or any number of departments. What is more, our tools allow us to delegate control of any department or subtree to folks outside of the computer center.

This is the same problem we wanted to solve with Windows 2000 user accounts. We wanted them to be created and expired automatically, we wanted new accounts to be put in the appropriate domains by default, and we wanted to allow our Windows 2000 administrators to have some ability to shuffle folks around, but not give every admin the ability to move every account. The tools used for our telephone directory could be used for this with almost no changes.

[9]Our baseline organization structure is based on our financial accounting system. We modify it slightly to reflect the actual organizational structure.

Futures

Having the common Windows/Unix/Kerberos account and password base is very nice, in that it allows us to deploy new services requiring authentication and have some options with how we do it. Since all of the IDs and passwords are the same, the users do not know or even care how it is done. At present, we require the users to change their password in order to access Windows 2000 and some other services. We like this from a security perspective, as the initial password is stored in clear text and may have been printed. On the other hand, this creates an extra step for new users. However, it appears that user convenience has won out over security, and we will soon be removing the requirement to change the initial password.

We do have a web based account pickup system in place that does put new users right on the password change web page once they get their initial password. Many of the new users take that opportunity to change their passwords. Of the 1596 new users who used the web pickup tool, 1467 (92%) changed their passwords.

We will continue to add new fields into Active Directory from our enterprise information systems. Now that the tools are in place and understood, it is much easier add new things.

Problems and Lessons

One problem is our general job queuing mechanism, so we will be investigating adding a special password change thread or simply multiple threads to keep password changes from getting stuck behind other jobs. A priority setting for jobs might be enough, since, in general, any given job is pretty quick. The problem comes when a password change gets stuck behind 1200 user account creation requests the problems arise.

Working with public key encryption can be very tricky and we have run into problems with how keys are generated and stored. We have two different ways of generating key pairs, and although both store the public keys in Oracle, and we can successfully encrypt with either key, the private keys were not interchangeable. For example, if I generated a private key under AIX, it would not work in Windows. We have subsequently learned why this was, and have fixed this. We

Name	Requested At	Done At	Elapsed Time
AdminReq	09:47:07 Jun-10-2002	09:42:13 Jun-10-2002	00:00:06
Applix	10:43:34 Apr-23-2002	10:44:42 Apr-23-2002	00:01:08
Kerb 4	10:43:34 Apr-23-2002	10:43:34 Apr-23-2002	00:00:00
Kerb 5	10:43:34 Apr-23-2002	10:45:18 Apr-23-2002	00:01:44
Simon	10:43:34 Apr-23-2002	10:43:59 Apr-23-2002	00:00:25
Win2000	10:43:34 Apr-23-2002	10:43:53 Apr-23-2002	00:00:19

Table 8: Password status sample.

have also hit problems with packages working under one version of AIX on a particular type of processor, and, when recompiled on a different machine, not working at all. Some of the packages we are using appear to have some buffer overruns and other memory layout issues.

A Big Hammer

The requirement for a high end database server and some of the other infrastructure may make this solution "too big" for some small site to contemplate. However, I feel that part of this is a matter of perception. When we first embarked on the Simon project, using Oracle added a big expense for software and hardware, and a lot of development effort. On the other hand, the challenge of trying to deliver an enterprise wide computing environment was simply too big to accomplish without using commercial support. This project will not help a small operation; it is simply too big a hammer. But if you are working with tens of thousands of user accounts, and a large turnover in population, you have no other choice but to go for the big tools.

While the Simon system is involved in many facets of our information systems, one of the most important roles it plays is as an interface layer between different vendor applications. It gives a place to apply our business rules and needs to the areas where the vendor products fall short. Part of the cost of doing this is having a development staff who can make these systems work together.

While it is certainly nice to think that we can do everything with free software, I do not feel that that is realistic approach to a large scale system. At our site, we spend close to one million dollars a year in software licensing and support contracts. Software license costs are simply a cost of doing business. We don't seem to have a problem obtaining four servers for our LDAP directories, so a database server shouldn't be any different.

References and Availability

All source code for the Simon system is available on the web. See http://www.rpi.edu/campus/rpi/simon/README.simon for details. In addition, all of the Oracle table definitions as well as PL/SQL package source are available at http://www.rpi.edu/campus/rpi/simon/misc/Tables/simon.Index.html . We can make the ADSI (C#) routines available as well. It seems unlikely that we can distribute the public key encryption routines used in the password management, but presumably you can find them elsewhere.

Acknowledgments

I would like to thank AEleen Frisch for her shepherding of this paper, as well as Deb Wentorf of Communications and Collaboration Technologies at Rensselaer for her proofreading and editing. I also want to thank Rob Kolstad for his excellent (as usual) job of typesetting this paper. Thanks also to Mike Douglass <douglm@rpi.edu> who wrote the LDAP part of the project, and Alan Powell <powela@rpi.edu> who did the ADSI part of the project.

Author Biography

Jon Finke graduated from Rensselaer in 1983, where he had provided microcomputer support and communications programming, with a BS-ECSE. He continued as a full time staff member in the computer center. From PC communications, he moved into mainframe communications and networking, and then on to Unix support, including a stint in the Nysernet Network Information Center. A charter member of the Workstation Support Group he took over printing development and support and later inherited the Simon project, which has been his primary focus for the past 11 years. He is currently a Senior Systems Programmer in the Networking and Telecommunications department at Rensselaer, where he continues integrating Simon with the rest of the Institute information systems. When not playing with computers, you can often find him building or renovating houses for Habitat for Humanity, as well as his own home. Reach him via USMail at RPI; VCC 319; 110 8th St; Troy, NY 12180-3590. Reach him electronically at <finkej@rpi.edu>. Find out more via http://www.rpi.edu/~finkej .

References

[1] Armstrong, Eric, Steve Bobrowski, John Frazzini, Brian Linden, and Maria Pratt, *Oracle 7 Server Application Developer's Guide*, Chapter 8, pp. 1-29, Oracle Corporation, Dec 1992.

[2] Armstrong, Eric, Steve Bobrowski, John Frazzini, Brian Linden, and Maria Pratt, *Oracle 7 Server Application Developer's Guide*, Appendix A, pp. A15-A20, Oracle Corporation, Dec 1992.

[3] eduPerson Working Group, eduperson object class, Technical report, EDUCAUSE, http://www.educause.edu/eduperson/, 2001.

[4] Finke, Jon, "Automated Userid Management," In *Proceedings of Community Workshop 1992*, Rensselear Polytechnic Institute, pp. 3-5, Troy, NY, June 1992.

[5] Finke, Jon, "Relational database + automated sysadmin = simon," Invited Talk, Sun Users Group, East Conference, Boston, MA, July 1993.

[6] Finke, Jon, "Institute White Pages as a System Administration Problem," *The Tenth Systems Administration Conference (LISA 96) Proceedings*, pp. 233-240, USENIX, October 1996.

[7] Finke, Jon, "An Improved Approach to Generating Configuration Files from a Database," *The Fourteenth Systems Administration Conference (LISA 2000)*, pp. 29-38, USENIX, December 2000.

[8] Good, Gordon, *RFC 2849: Ldap Data Interchange Format*, June 2000.

[9] Howe, Timothy A., *The Lightweight Directory Access Protocol: X.500 Lite*, Technical Report CITI TR 95-8, University of Michigan, July 1995.

[10] Hughes, Doug, "User-Centric Account Management with Heterogeneous Password Changing," *The Fourteenth Systems Administration Conference (LISA 2000)*, pp. 67-76, USENIX, December 2000.

[11] Microsoft, "Active directory service interfaces overview," http://www.microsoft.com/windows2000/techinfo/howitworks/activedirectory/adsilinks.asp.

[12] Microsoft, "Visual C#.net," http://msdn.microsoft.com/vcsharp, 2001.

[13] Portfolio, Tom, *PL/SQL Release 8 User's Guide and Reference*, Oracle Corporation, Part No. A58236-01, December 1997.