Using LDAP to solve one companys problem of uncontrolled user data and passwords

Andres Andreu

Case Study in Information Security
GSEC – Assignment 1: Option 2
Version 1.4b

Using LDAP to solve one company's problem of uncontrolled user data and passwords

Author: Andres Andreu
October 30, 2003

1. Abstract

This case study will analyze a massive undertaking of centrally consolidating user data, and in particular passwords, from numerous sources. The effort goes way beyond just the securely capturing and storing of the data but it builds a framework whereupon the data can be successfully used by numerous disparate applications for authentication, authorization, access control, and overall data management. This project faced some hurdles which seem all too common in the IT industry today. The number of applications for the company had grown out of control due to legitimate business needs and with each application came the responsibility and burden of user data security and management in very different fashions. Each target application existed in an island, or in a stand-alone model.

The analysis will show how using open source tools we faced the challenge, put a solid solution in place, and did so without losing control of the technology and while keeping the budget at hand manageable. Commercial software alternatives were analyzed but we were after the maximum level of flexibility, control, robustness, and resilience. As such, very expensive and closed third party solutions did not appeal to us.

2. Before

2.1 The Challenges

The challenges our company (we will refer to it by the bogus name “Foo”) faced were daunting and 2 fold. Foo is a very large media company with offices/employees in over 110 countries. Moreover, Foo's client base ranges from very large Fortune 50 companies to smaller, yet still global clients. There was an IT driven requirement for change as well as a user community, or business, driven one.

On the technological/IT front was challenge 1 - the problem of inconsistent authentication mechanisms (mostly third-party driven), with different standards, across numerous unrelated global applications. This was coupled with the end-user facing issue, or challenge 2, of too many un-synchronized passwords to remember for different, disparate, applications/data sources. These challenges were set forth to me and I was tasked with designing, architecting, and deploying a technology solution that would centralize end-user address-book type data/password management, standardize authentication mechanisms, standardize & centrally enforce corporate application password policy, and subsequently introduce the end-user community into a single-password environment across all targeted applications/systems. This was supposed to be accomplished while keeping overall cost to a minimum. The cost factor was important but not as important as the time frame. Foo had operated in chaos for way too long and it was affecting client confidence in Foo's technological capabilities. The engineering effort would end up spanning across multiple IT disciplines including software engineering, networking, and security.

We had to contend with budget constraints, the number of user objects in question (initially 10,000+), and the fact that the design had to factor in user data external to Foo. This rendered advanced authentication equipment/methods such as biometrics equipment, smart cards, dual-channel mechanisms, etc, too expensive and just downright unrealistic. The bottom line is that we at Foo had to design and implement a solution that would work for Foo and Foo's clients accessing our applications and data. Moreover, all the target applications for the authentication piece were web based so browser usage was a hard, unavoidable, requirement and the end-user authentication experience had to be simply browser based. Besides that, we could not factor in the introduction of any devices, or extraneous software, to a clients environment. We had to create a solution to be effective when facing 2 lowest common denominators, the human being (end-user) and the web browser.

2.2 The Team

The team was put together and led by me. As is typical in I.T. these days the team members and I wore many hats during the life cycle of the project. I managed the work of the internal team, as well as all outside contractors but I played a major hands-on role as well. I directly led the architecture and engineering efforts in a totally hands-on fashion.

3. During

3.1 The Design Decisions

A solution like the one we were tasked to build at its heart has a central data store, or repository of some sort. The traditional norm of a database came to mind immediately. At Foo, the Oracle[1] and MySQL[2] databases are used extensively on other projects, and are actually a documented company standard. But from our years of experience neither one of these database server software is truly optimized for high performance in read (SELECT in the SQL world) operations as opposed to heavy writes. MySQL[2] is generally known for its positive performance in SELECT SQL operations. But we were facing a data set that would not change often but would be read heavily. The LDAP protocol seemed like the right fit because it is optimized for few writes but many reads. Our experience based decisions are echoed all over the web if research takes place. In particular, according to Jon Roberts[3], "The LDAP specifications require support for the indexing of any attribute. Because data in a directory is pre-normalized, appropriate queries are answered in optimal time (i.e. with performance on at least the same order of magnitude as with any other database system)..." On the replication front LDAP also had the edge due to its effective transactional replication capabilities.

3.2 The Technology

In order to maximize the flexibility, internal control of technology, and robustness of the proposed solution, all of the technologies chosen to meet all of the requirements were entirely open in nature. Due to flexibility limitations and lack of Foo control, proprietary technology was decided against from the onset of the effort. The exact technologies chosen and implemented for the final solution are:

- SuSE Linux
- OpenLDAP version 2.0.X
- XML
- Apache
- Tomcat
- SOAP based web services
- JBoss 3.X
- J2EE 1.4
- MySQL 3.X
- Perl 5.8.X
- Ruby 1.6.

At the heart of the solution is an OpenLDAP based directory. OpenLDAP is described as "... an open source implementation of the Lightweight Directory Access Protocol"[4]. We did extensive research into the various commercial options for directory based solutions and basically decided to start our own from scratch with a possible migration path to something commercial in the future. The contender at design time was Sun Microsystems' I-Planet[5] LDAP server. So an internal requirement for the engineering team was to build something solid based on documented standards so that if we ever decided migration off OpenLDAP was necessary, it would be as smooth as possible. But, OpenLDAP has stood the test of time thus far for us. We are now into the second year with this solution in production on a global scale and no longer are seeking to replace it.

3.3 The Schema

With technologies collectively decided upon we were set to start designing the foundation of the solution, or the LDAP/directory schema. Often times data level schemas are designed with functionality alone as a focus. But we were designing with security, authentication, passwords, and functionality as focal points.

The overall directory tree schema of the Foo directory went through an analysis phase where we built mock directories and tested each setup. In analyzing high load query performance we decided that a flat structure for user objects would serve Foo best to keep all queries contained to one level. Multi-level tree searches were just not responding as quickly as the flat hierarchy when running queries side by side with large data loads as the search targets. Hence, the user object organizationalUnit looks like:

- ou=people,dc=foo,dc=com

as opposed to something like:

- ou=internalPeople,dc=foo,dc=com
- ou=externalPeople,dc=foo,dc=com

where a query for any user object could possibly have to hit 2 sub-levels. After all, when a query request for user data is submitted it is impossible to know up front whether that object is internal to Foo or external. Under high query load the flat structure proved to be more robust.

For user object data, the foo.schema was designed with Smith's RFC 2798[6] as a focal point. From there we utilized SUP, or upwards inheritance, from inetorgperson to inherit from top. Downwards, we referenced RFC 2252[7] by Wahl, Coulbeck, Howes, & Kille. We did this to define Foo's customized schema based on industry accepted standards. There are 2 levels of customization for the schema post-inheritance, authorizedPerson and fooPerson (which has upward SUP to authorizedPerson). The objectClass for authorizedPerson looks like:

```
objectclass (
        1.3.5.1.4.1.10903.101.5
        NAME 'authorizedPerson'
        DESC 'Extension of inetOrgPerson that contains authorization roles'
        SUP inetOrgPerson
        STRUCTURAL
        MAY ( hasAccessRole )
        )
```

In foo.schema then there is:

```
objectclass (
        1.3.5.1.4.1.10903.200.1001.1000
        NAME 'fooPerson'
        DESC 'authorizedPerson subclass with specific attributes for Foo'
        SUP authorizedPerson
        STRUCTURAL
        MUST ( internalOrExternal )
        MAY ( customAttrib1 $ customAttrib2 $  customAttrib3
                $ customAttrib4 $ customAttrib5 $  customAttrib6 )
        )
```

with the required attribute of:

```
attributetype (
        1.3.5.1.4.1.10903.200.1001.1
        NAME 'internalOrExternal'
        DESC 'Flag denoting whether a person is an employee or not'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{20}
        SINGLE-VALUE
        )
```

The required (MUST) attribute of internalOrExternal is key from a security perspective because we needed a clear way of differentiating between internal users (Foo employees) and external ones (non Foo employees that have been granted access to Foo data sources). Keeping in line with the flat hierarchy of data for better performance, we totally adhere to the flat hierarchy while still being able to differentiate all user objects (as internal to Foo or not [i.e. External]) by building this attribute in to the user object schema. There are further custom schema extensions for foo.schema but they are beyond the scope of this security discussion. The final objectClass level hierarchy for objects in ou=people,dc=foo,dc=com looks like:

- objectClass: top
- objectClass: inetOrgPerson
- objectClass: authorizedPerson
- objectClass: fooPerson

The technical requirement at hand was that we had to store user objects both internal to Foo and external as well. Numerous clients that do not directly work for Foo are granted access to specific applications hosted within Foo's infrastructure. So this aspect of the directory was designed to provide granular level control of centrally managed and stored access control data.

One of the baseline foundations that we decided to build upon was Zeilenga's RFC 3112[8]. This document states that

> "... storage schemes often use cryptographic strength one-way hashing. Though the use of one-way hashing reduces the potential that exposed values will allow unauthorized access to the Directory (unless the hash algorithm/implementation is flawed), the hashing of passwords is intended to be as an additional layer of protection."

Further details of Foo's implementation can be seen in section 3.5 below.

3.4 Application Access Control

In analyzing the overall goal of tight application access security for Foo and for our numerous clients, we determined that username and password should simply not be enough to gain access in to any Foo resource. Luc Russell states that "[a]ccording to the principles of Role Based Access Control (RBAC), users are granted membership into roles based on their competencies and responsibilities in the organization. The operations that a user is permitted to perform are based on the user's role."[9] Thus, user objects would have to be granted a very specific access privilege (one per specific application), using the hasAccessRole attribute, to gain legitimate access to any Foo application resource. The foo.schema was designed to improve on the traditional 2-factor, crackable and brute-forceable, authentication of username and password. Foo's user object level schema design revolves around the forcing of a tri-factor data-based authentication model.

3.5 LDAP Access Control

Foo's ldap design forces a tri-factor data-based authentication model. So basically the destination applications need to understand this. They are taught how to query for this third factor in order to actually use the data presented by the framework. Rather then support the typical uid:userPassword LDAP bind model, we at Foo found it far more secure to implement the tri-factor model which comes with security requirements at different levels. There is more up-front work to do designing the overall scheme but the extra work provides a more secure and resilient infrastructure.

3.5.1 Access to LDAP data

We limit LDAP data read access to strategic accounts with the ability to perform LDAP bind actions. This is done via OpenLDAP's ACL mechanism in the file "slapd.conf". A snippet from Foo's environment looks like:

```
defaultaccess none

access      to attr=userPassword
            by dn.children="ou=admin,dc=foo,dc=com" write
            by anonymous auth
            by self write
            by * auth

access  to dn="cn=FooAdmin,dc=foo,dc=com"
            by * none

access  to dn.subtree="dc=foo,dc=com"
            by dn.children="ou=admin,dc=foo,dc=com" write
```

For each object granted access via the ACL, an LDIF file is created (and maintained in a CVS repository) as such:

```
dn: cn=xAdmin,ou=admin,dc=foo,dc=com
objectclass: top
objectclass: person
cn: xAdmin
sn: Admin
userpassword:: {SSHA}2K48+ANSpPaWQsp4g+vL2OlOt3h8AhH9
```

That LDIF data is then ingested to the respective LDAP server (isolated to a replicated slave) where read access is being granted. The LDIF data is added with the ldapadd utility that comes standard with OpenLDAP distributions. Foo's master LDAP server has the equivalent methodology with only a few strategic accounts having write privileges and they are monitored via verbose logging.

This modular approach provides LDAP (protocol) level security in a loosely coupled fashion. It gave us the ability to tightly design access control to the directory itself. In this way we can also allow granular levels of access control.

For example, object A is granted read access to server A, by default, object A is not allowed to see any data on server B. On server B there would have to be an explicit addition of object A's data and then a corresponding ACL entry in slapd.conf to make this access possible.

With this mechanism Foo's model allows only one very specific account per application, or any other access granted entity, to bind to an LDAP server. The reason for not allowing anonymous bind, and to only allow strategic access with read permissions, is that according to RFC 3112[8] "... [i]t is RECOMMENDED that hashed values be protected as if they were clear text passwords..." We at Foo took this recommendation very seriously.

Thus, very specific and controlled accounts are allowed to bind to LDAP and then only those accounts are allowed to perform lookups, or to perform a query, against the stored data. The data that is made available to these legitimately binding accounts are the basic attributes used for the Foo tri-factor authentication model are:

1. mail
2. userPassword
3. hasAccessRole

3.5.2 The Authentication Attributes

1. mail – this is the only true unique element in Foo's data set.

2. userPassword – this attribute has the very flexible ability to store a password in hash form. Foo's deployment, as an organizational standard, stores nothing password related in clear text. The deployment supports the following one-way hashing algorithms:

    - MD5
    - SHA
    - CRYPT (supported but not used)

For each one of these hashing schemes both hexadecimal and Base 64 encodings are supported. The solution had be this flexible because we would be importing hashed password data from numerous disparate and legacy databases.

We had no control over the legacy data we would face at Foo. As an example we will hash the string p@s$w0rD. The stored data in Foo's hashed userPassword attribute would then look like one of the following:

MD5 hex	{MD5}6b92a5db2a5cbc46e854767bbf0cf8eb

MD5 hex	{MD5}6b92a5db2a5cbc46e854767bbf0cf8eb
MD5 base64	{MD5}a5Kl2ypcvEboVHZ7vwz46w==
SHA hex	{SHA}25eac497991ea306b643be3896913409fdfc53a3
SHA base64	{SHA}JerEl5keowa2Q744lpE0Cf38U6M

3. hasAccessRole - this is a multi-value custom attribute that has the following structure in the schema:

```
attributetype (
        1.3.5.1.4.1.10903.101.2
        NAME 'hasAccessRole'
        DESC 'Property of an authorizedPerson denoting an accessRole'
        EQUALITY distinguishedNameMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
        )
```

As values this attribute accepts basic strings in the following format: uniqueIdentifier=XXX,ou=roles where “XXX” is typically a unique resource identifier. Thus the authentication module has the secure ability to accept a username and password but that will get a user no-where unless the appropriate accessRole is also present for that respective user object upon query time.

3.6 Authentication

Now that the framework for this tri-factor model was in place we created some mechanisms that would utilize it. Standardized authentication, leveraging the now single password hash stored in LDAP, would be challenging because we were facing numerous, very different systems. So we set out to create at least a flexible choice of authentication mechanisms that all communicated appropriately with the Foo LDAP space. The following modules/options were designed and put in place:

1. SOAP based web service performing the authentication function. This was written in Java and compiled using Ant. Leveraging the built in JBoss abstraction of Apache::Axis this was compiled as a sessionBean based web service and its details exposed via WSDL (for pseudo-code and a snippet of the WSDL see appendix A). One of the serious challenges at hand was the fact that the web service had to accept a password in the clear. The reason for this is that we would never know what the stored hash was. So we needed to, as securely as possible, read a password in the clear and then hash this submitted password for comparison based on the respective stored hashing scheme. Based on this, TLS was a must for all SOAP clients utilizing this service. The SSL/TLS certificate was applied server side and all respective clients had to factor in an https call for authentication query data. Each respective client is also given a static unique identifier string which identifies the

accessRole privilege that needs to be checked on each authentication query. There is a MySQL database that stores that client id data.
2. Built an LDAP TCP reverse proxy to allow direct access for those applications that were natively X.500 clients and have been presented with Foo bind credentials. The proxy protects the server on an infrastructure level by never allowing TCP sockets directly to the server. In this fashion the LDAP clients get access to the data while not allowing sockets on ports other than those used for directory services.
3. Implemented the Radius protocol, as per Rigney, Willens, Rubens, & Simpson's RFC 2865[10], as a custom server written in Ruby that effectively communicates with Foo's LDAP. It was written to understand the tri-factor model and overcome all of the inflexibilities inherent in existing Radius solutions. We found all readily available Radius servers operated in a pure directory bind model and that was just not sufficiently secure for Foo's goals. For a snippet of the server code, see Appendix B.

3.7 Replication

Replication model - the solution was being implemented for a globally dispersed organization with offices in over 100 countries and even more globally dispersed clients who needed access to LDAP controlled applications. The strategic design of the LDAP infrastructure consists of a slurpd[4] based model with one master in the NY data center and 4 physical slaves:

- one in the main NY data center
- one in a separate co-location in NY
- one in the company's data center in London
- and one in the company's data center in Hong Kong

The lessons learned from 9/11 and the recent Northeast power outage guide the strategic locations of servers holding key data with redundancy and high availability heavily in mind. Moreover, a warm spare master server is being created as this case study is being written.

3.7.1 The Master

The master LDAP server is a very hardened SuSE Linux Enterprise Server (SLES) 8 with a strict IP-Tables configuration running internally. It has also been put through an extensive Bastille[11] hardening process. Furthermore, this server is strategically positioned on a highly isolated segment of the Foo network.

3.7.2 The Slaves

Each one of the LDAP slaves is a very hardened SuSE Linux Enterprise Server (SLES) 8 with IP-Tables running internally. They also go through an extensive Bastille[11] hardening process. Hence the slave servers are protected all the way down to an OS level as part of the holistic approach followed during this project.

3.8 The Infrastructure

From a network infrastructure perspective, Foo has a meshed IPSec based

Nortel VPN to rely on. But we needed the LDAP framework to be highly available. So we decided to build in full redundancy and be operational over the public internet if there were ever problems with the VPN connections. For all remote replications we utilize ssh tunnels riding on top of the ip-sec VPN tunnels with secondary route paths across the public internet. Since IPSec and SSH, from an encryption perspective, are mutually orthogonal this works beautifully and we, in essence, have data moving to 3 physically distant destinations with nested encryption. SSH works at the session layer (5) while IPSec operates at the network layer (3) of the OSI model and so the 2 layers of encryption literally just ride one above the other.

From a network downstream perspective, we sliced the globe by region. 4 regions were addressed as per the way Foo has its regional boundaries established: EAME, NA, AP, LATAM. Each respective regional device look upstream to their regional LDAP slaves as the primary service provider. They also have secondary and tertiary backup configurations to look at other region's LDAP slaves via SSH tunnels over the public internet. To provide an example: an application server in Paris looks to London as the primary upstream regional LDAP slave. It will attempt to establish the initial connection over port 389 via the IPSec VPN. If that fails it will attempt to establish the connection to the same server in London via an SSH tunnel over the public internet. If that also fails the server will attempt an SSH tunnel connection to an LDAP slave at a highly available co-located site in NY. If that fails as well it will attempt the same type of connection to an LDAP slave residing in Hong Kong.

3.9 The Tools

For the ingestion and management of data we internally built numerous web based tools. For convenience all of them are accessible via the https protocol so a simple browser is all that is necessary. We utilize TLS certificates on all of these ingestion points so as to subvert any attempts at sniffing on Foo's transmissions. These tools range in functionality from the ability to create a single user object to the uploading/ingestion of a comma-separated value (CSV) file with mass amounts of data for processing/ingestion. One of the most beneficial aspects of the tool set is that a user can have access revoked to one, or a set of, application(s) by some very simple GUI based clicking mechanisms. The changes are reflected throughout the global LDAP framework in near real-time. Slurpd is fully transactional and operates very efficiently thus enhancing the overall security at hand by ensuring that changes of data, especially the removal of access roles, are propagated out to the LDAP slaves quickly and reliably.

3.9.1 Domino XML2LDAP Bridge

We also coded a Java based servlet to be hosted on a Lotus Domino server that would stream out XML of all the addressbook data. We then built a bridge program to pull these XML streams from Domino via https, parse through the stream, and then fully synchronize the target LDAP master server every hour. The whole process is controlled by cron statements and is almost militaristically timed/synchronized. The bulk of the Foo internal user community is managed by

these sets of functional code. No human intervention, other than the globally dispersed Lotus administrators, is necessary for the data to get into LDAP in a fresh manner every hour.

3.9.2 Password Change Tool

In order to minimize the need for human administrators we built a small but effective tool to allow external user's to manage their own passwords. Presenting appropriate credentials an external user is given a 2 hour window to hit a very specific https entity to effect a password change in LDAP. The link, which includes a very large random string which has a corresponding directory on the web server, is emailed to the account seeking to effect the change. A cron script runs every hour that does an rm (remove) on any directory that is older (by timestamp of creation) than 2 hours. A successful password change also kills the directory that just allowed the password change. This is done entirely in Perl utilizing the CGI module.

3.10 Data Purposes

From a data perspective we identified the following uses for the data that would be stored in the Sleepycat database and exposed via LDAP:

1. Authentication
2. Application/DB
3. Data Management
4. Single Sign On (SSO) Environment
5. Access Control

4. After

The current state is that Foo has gained great benefit from the labors of the design, architecture, and deployment presented in this document. In specific, in the following areas:

1. Authentication – numerous software systems now enjoy the benefit of a single password environment when before each operated as an island of data/functionality. Particular software systems at Foo that are all now using LDAP exclusively for authentication:
    - Foo's corporate web-based e-mail system
    - Foo's client facing extranet (4 globally dispersed deployments)
    - Foo's corporate intranet suite (5 distinct web applications)
    - Foo's client facing high speed file transfer solution
    - Foo's wireless network via Radius
    - Foo's VPN solution
    - Foo's PDA (Palm OS & Blackberry OS) Address Book
2. Application/DB – Foo has enabled LDAP to be the master source of user data for numerous applications and put in place synchronization mechanisms to regularly update the destination systems' databases.
3. Data Management – Foo uses LDAP for central data management.

4. Distributed Single Password Model – LDAP has enabled Foo to achieve a single password model irrespective of an environment full of disparate systems.
5. Single Sign On (SSO) Environment – In specific cases, such as the intranet suite, LDAP has enabled a true SSO environment.
6. Access Control – Integration between LDAP and strategic applications has enabled LDAP to be a strict source of access control enforcement.

The bottom line for Foo is that prior to this solution being in place password strength enforcement was impossible. There was a legitimate real world reason for this in the fact that end-user pressure was intense. Business had to run and the user's simply had too many passwords to keep track of. So as not to forget passwords many violations of sound security practices were taking place, not as overt violations but for business continuity. Password strength was softened for ease of remembrance and this in turn eroded the security strengths of the respective applications and data sources. Moreover, breach level of risk was at an all time high because the end user community was managing all of their passwords in very exposing fashions.

Couple the rogue password issue with the fact that Foo had very little control of application user access control and the situation was such that client confidence in Foo's security was far from optimal. There were just too many applications, with too many users, with too many realms to manage, and not enough qualified administrators. This is no longer an issue since our implementation has centralized the access control data. We also built a very easy to use web based administrative tool that empowers non-administrator types with very simple but effective capabilities in the access control space. These folks are not IT staff but control user access within their own mini-space, and now do so effectively with very little training because the tool we built was designed with them as the focal point.

A lot of work also went into the integration efforts to get applications to accept credentials from an LDAP source. While this was very painstaking work, and there was very little re-usable code, Foo's enterprise level of application architecture now has a coherence and unity that, at Foo, was once looked upon with nirvana-like qualities. The fact that the same unified source is authenticating a user is facilitating single-sign on capabilities at Foo. The reason for this is that application B now knows that a user establishing a session to it from application A was authenticated using the same source of data. Thus, the source is trusted just as if the user object went directly to application B.

While we now operate at risk of one password meaning a potential breach to many systems, it means we face a different level of potential offender. Social engineering aspects aside, there at least has to be some strong technical knowledge present to steal a users password. This level of risk is very different then simple knowing that 10 passwords are scribbled on a piece of paper and sitting in an unlocked drawer in an exposed cubicle. Foo user's now have an easier time simply remembering one password that will in turn (coupled with the

proper access role in LDAP) grant them access to multiple specific resources. It has allowed Foo as a whole to operate in a more secure space in the context of end-user password strength and management.

References:

1. Oracle 9i database URL: http://otn.oracle.com/products/oracle9i/index.html

2. MySQL Database Server
URL: http://www.mysql.com/products/mysql/index.html

3. Roberts, Jon: “Why use a Lightweight Directory Access Protocol (LDAP) server?”.
URL: http://www.mentata.com/ldaphttp/why/ldap.htm

4. “OpenLDAP 2.0 Administrators Guide”:
URL: http://www.openldap.org/doc/admin20/intro.html

5. Sun ONE Directory Server
URL: http://wwws.sun.com/software/products/directory_srvr/home_directory.html

6. Smith, M. “ Definition of the inetOrgPerson LDAP Object Class.” April 2000.
URL: http://www.faqs.org/rfcs/rfc2798.html

7. Wahl, Mark; Coulbeck, Andy; Howes, Tim; Kille, Steve, “Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions.” December 1997
URL: http://www.faqs.org/rfcs/rfc2252.html

8. Zeilenga, K. “LDAP Authentication Password Schema.” May 2001.
URL: http://www.faqs.org/rfcs/rfc3112.html

9. Russell, Luc, “Introduction to Securing Web Applications with JBoss and LDAP” URL: http://www.developer.com/java/ejb/article.php/3077421

10. Rigney, Carl; Rubens, Allan; Simpson, William; Willens, Steve, “Remote Authentication Dial In User Service (RADIUS)” June 2000
URL: http://www.ietf.org/rfc/rfc2865.txt

11. Bastille Linux
URL: http://www.bastille-linux.org

Appendix A
Snippet from WSDL:

```
...
<wsdl:message name="authenticateUserRequest">
   <wsdl:part name="in0" type="xsd:string" />
   <wsdl:part name="in1" type="xsd:string" />
   <wsdl:part name="in2" type="xsd:string" />
</wsdl:message>

<wsdl:message name="authenticateUserResponse">
   <wsdl:part name="authenticateUserReturn" type="xsd:boolean" />
</wsdl:message>

<wsdl:portType name="LDAPLogin">
<wsdl:operation name="authenticateUser" parameterOrder="in0 in1 in2">
<wsdl:input message="impl:authenticateUserRequest"
name="authenticateUserRequest" />
<wsdl:output message="impl:authenticateUserResponse"
name="authenticateUserResponse" />
</wsdl:operation>
</wsdl:portType>
...
```

Pseudo-code for Authentication Web Service:

```
public boolean authenticateUser(mail, password, AccessString)
    Open up connections to the target OpenLDAP directory
    Setup SearchControls for subtree scope
    establish array of String objects { "hasAccessRole", "userPassword" }
    search targetContext (mail)

    if user object exists in LDAP
      get user password and accessrole attribute (based on AccessString)
        if the attributes have legit values
          get password, cast it as byte array and convert to string
          get accessrole attrib as multi-value

          if password is in hex
            loop thru enum of multi value attrib
              if accessrole value exists return true
          if password is base64 encoded
            loop thru enum of multi value attrib
              if accessrole value exists return true

    close connection to the target OpenLDAP directory
    else return false
```

Appendix B
Snippet from Radius Server:

```
def run_server

server = UDPsocket.open
server.bind ($radius_address, $radius_port)

start_time = Time::now
loop {
  build_ldap_cache
  start_time = Time::now
  while Time::now - start_time < $max_time_per_run

    if select ([server], nil, nil, 10)
      m = server.recvfrom (4096)
      server.send (_radius_response (m[0]), 0, m[1][2], m[1][1])
    end
  end
}

end
```