LDAP Proxy AuthZ avoids *Superpowered* Middle-Tiers*

Technical report DISI-TR-08-01

Marco Ferrante
DISI, Università di Genova, Italy
ferrante@disi.unige.it

Abstract

LDAP directories offer a fine-grain authorization framework, but these capabilities are often ignored by poorly written applications which require accounts with very high privileges to manage LDAP data.

Proxied Authorization is a LDAP security mechanism which helps to develop less critical client applications. Unfortunately, developers of client applications seem to ignore this opportunity.

The article will discuss general aspects of LDAP Proxied Authorization comparing available implementations, will show, using a fictional scenario, how to use it with common tools and how to write custom applications. Finally, it will present benefits, some potential problems and possible solutions.

INTRODUCTION

LDAP directories [12] are often used as back-end authentication services in multi-tier environments like the one shown in Figure 1, e.g. where a web application can ask to the LDAP directory to validate identifier and password provided by the users.

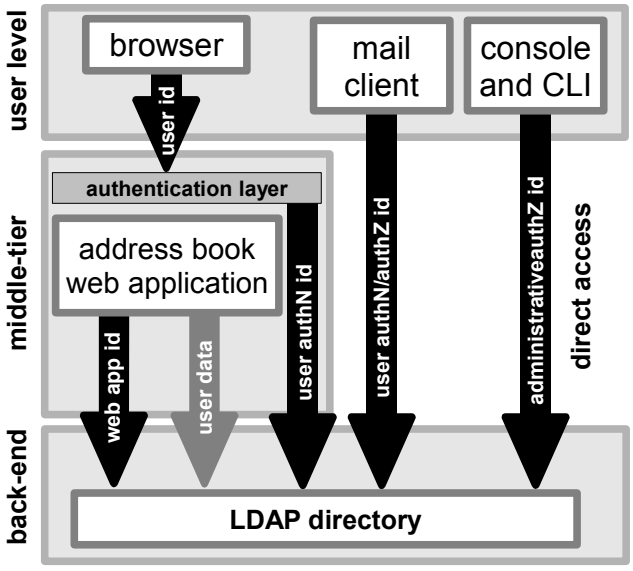

Figure 1: In multi-tier environments, the same user can access to LDAP directory using several identity.

A particular issue is related to those applications which manage LDAP directory itself. In a typical scenario, people browse corporate LDAP directory using their favourite LDAP-enabled mail agent, such as Microsoft Outlook or Mozilla Thunderbird. At the same time, a web application allows users to update their own data or data of other persons which they are responsible for. Usually, there are also other applications used by system administrators to manage LDAP data involved in e-mail routing, authentication gateway and so on. Obviously, all these applications have a protected access, using passwords stored in LDAP directory, and apply complex business rules to data, e.g. to permit registration of valid e-mail addresses only.

Such a situation requires a fine-grained authorization (*authZ*) system. If authorization is defined in the usual way of checking if "a subject is allowed to perform a certain operation on a target resource", the LDAP distinctive is that both subject and resource are LDAP entities (*entries* in LDAP idiom). As a consequence, with LDAP it is easy to implement a sharp authorization mechanism which permits to specify complex policies. In the following, we will refer to *identity* as an LDAP entry that can be used to authenticate or connect to LDAP system, and to *user* as a human, potentially associated with an *identity*. Every identity is identified by its DN (*Distinguished Name*).

Unfortunately, basic LDAP capabilities can be exploited only when a client is directly connected to the server. In multi-tier applications the authentication layer is often completely decoupled from the business tier, and therefore they impose different challenges.

The majority of LDAP servers already implemented a solution to these problems, called "Proxied Authorization", which allows an identity (*proxy*) to impersonate another identity (*proxied*). Notice that this feature is different from *proxy services*, as known in HTTP (or also in LDAP), which are programs that process the client requests by forwarding them to other services.

Proxy authorization is a security mechanism that unbinds authentication identity from authorization identity. In our fictional example, an address book application authenticates itself using a service identity but operates with the identity of the current user, leaving the LDAP server to manage user's capabilities.

BUILDING UP THE REFERENCE SCENARIO

In this article, we will refer to a scenario where an LDAP directory service used within a university has its DIT (*Directory Information Tree*), shown in Figure 2, organised in branches one for each department plus a special organisational unit for the students. Departments can be organised in sections.

* This is an author produced version of a conference paper including minor corrections. The paper has been peer-reviewed and it has been presented at the Net&System Security '07 (NSS '07) meeting, Palazzo dei Congressi, Pisa, Italy, 27 Nov, 2007. http://www.atsystemgroup.org/it/nss07

Example 1: Policy implemented using Sun Directory Server ACIs in LDIF format.

```
dn: dc=uni.test
aci: (targetattr = "telephoneNumber")(version 3.0; acl "Read contact info";
 allow(read,compare,search)(userdn = "ldap:///anyone");)
aci: (targetattr = "telephoneNumber")(targetfilter = "(objectClass=organizationalPerson)")
 (version 3.0; acl "User self update";
 allow (write)(userdn = "ldap:///self");)
aci: (targetattr = "telephoneNumber")(version 3.0; acl "Secretary access";
 allow (write)(userattr = "secretary#USERDN");)
aci: (target="ldap:///($dn),dc=uni.test")(targetattr = "telephoneNumber")
 (version 3.0; acl "Delegate administration";
 allow (write)(groupdn = "ldap:///cn=local admins,[$dn],dc=uni.test");)
```

Example 2: Policy implemented using OpenLDAP ACL as in *slapd.conf* file.

```
access to filter="(objectClass=organizationalPerson)" attrs=telephoneNumber
  by self write
  by * read break
access to dn.regex="^([^,]+),(.*)$" attrs=telephoneNumber
  by group/groupOfUniqueNames/uniqueMember.expand="cn=local admins,$1,$2" write
  by group/groupOfUniqueNames/uniqueMember.expand="cn=local admins,$2" write
  by group/iNetOrgPerson/secretary.expand="$1,$2" write
  by * read
```

Employees and faculty are recorded as entries of *inetOrgPerson* object class, students belong to *organizationalPerson* class and guests are recorded under *person* class. The policy of the university allows everybody except guests to manage his/her own telephone and mobile number, postal address and other contact informations. If an assistant is assigned to a person, the assistant's DN is registered in the attribute *secretary* of the person entry, and he/she can also update the same data for that person. Finally, every department or organisational unit can have local administrators (a group named *cn=local admins*) which can overwrite contact information for everybody in the department as well as for the department building, for internal workgroups, etc....

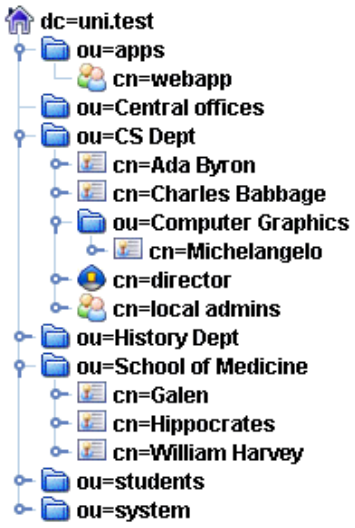

Figure 2: A sample Directory Information Tree (DIT).

These rules can be easily implemented as a set of ACI (*Access Control Item*) on LDAP system. Implementations that are sensitive to ACIs evaluation order organize items in ACL (*Access Control List*). Since configuration is not a part of LDAP standard, each implementation uses a specific mechanism to define and handle ACIs. Without ignoring general aspects, we will mainly consider examples derived from Sun Directory Server[1] version 5.2 and OpenLDAP version 2.2, two of the most used LDAP implementations. Programming examples will be developed in PHP, due to its compactness, its widespread and the built-in LDAP support.

Sun Directory Server stores ACIs directly in entries as values of attribute *aci*. ACIs are evaluated accumulating them from DIT root to target entry [4]. Example 1 shows ACIs that implement the university policy.

With OpenLDAP, the same result can be obtained writing an ACL, as shown in Example 2, in the configuration[2] file *slapd.conf*. In both cases, policies are applied when an user connects directly to the LDAP system.

However, it is easy to think to additional rules; we can imagine to allow only the user to update his/her home phone number, to hide mobile numbers to the colleagues from other department, to permit access in working hours only, etc.... In multi-tier environments, only access control based on the client network address could be ineffective. In fact, from the point of view of the LDAP server, all connections are originated by the middle-tier application server.

To consider a different usage of proxy authorization, suppose you want to delegate some people to administer specific entries subsets. You do not want to grant high privileges to their ordinary identities (the ones they use for everyday tasks) nor to create a new functional identity specific for such administration tasks, because it requires additional management and update.

[1] Sun Directory Server's syntax applies also to Fedora Directory Server, which share a common origin. OpenDS, which main sponsor Sun Microsystems considers its next-generation directory server, has some incompatibility [2]; specifically, it requires an additional *privilege* attribute.

[2] Example ACL is referred to OpenLDAP version 2.2; previous releases use a slightly different syntax. Newer versions can optionally save ACLs in a special *cn=config* entry. For the sake of clarity, this ACL is not exactly equivalent to the one used in Sun Directory Server example.

In these situations, proxy authorization can be seen as a sort of *setuid()* function on POSIX systems. A management entry, as *cn=delegation,ou=system,dc=uni.test*, can be entitled to use high privileges on directory, but can be set up in a way that cannot really connect, e.g. leaving it without password. Delegate managers will connect using their own identity and then perform operations as proxy for the management identity. If one delegate retires, only his main identity should be disabled, without any intervention on service accounts or shared passwords.

LDAP AUTHORIZATION ON MULTI-TIER APPLICATIONS

Which strategy for authentication can the web address-book application use? Normally, this class of applications is designed to connect and to operate on the *back-end* using a service identity. Although widely used, also with other systems such as databases, this solution has several drawbacks:

- Middle-tier tends to have too much privileges, because it must have at least as many privileges as the union of all users to whom it performs actions for. In our example, the identity *cn=webapp,ou=apps,dc=uni.test*, used by web application, must be enabled to modify the telephone number at least for every user and every organizational unit, while there is nobody in the university with such high privileges.
- At the same time, the application cannot rely on already defined ACIs and developers must reimplement an authorization mechanism to cope with single user capabilities. These mechanisms must remain synchronized with possible reconfiguration of LDAP access control.
- Application deploy requires to the system manager a considerable knowledge of how the application works, to apply suitable ACL, instead of focusing on what users are allowed to do.
- Real user activity is hidden to default auditing and logging systems of the directory system because each operation appears as it has been done by the service identity.

Problems arise because for a long time in computer software authentication identity and authorization identity have been mixed up. Nowadays, a clear separation between these identities is supported by many protocols and applications such as IMAP4 (RFC 3501), XMPP (RFC 3920) or latest versions of Oracle database. LDAP implements this feature with the "Proxied Authorization" mechanism discussed in the next sections.

PROXIED AUTHORIZATION

Since version 3 of the LDAP protocol, server designers can extend capabilities of their software using a mechanism known as *Controls*. When a client invokes an operation, it may attach one or more controls to the LDAP message in order to modify the semantics of the request. Controls sent by clients are termed "request controls", but also the server can add controls to the response. Controls should not be confused with *Extensions*, which define a completely new operation not provided by protocol standards.

Formal definition of LDAP controls in ASN.1 notation is:

```
Controls ::= SEQUENCE OF Control
Control ::= SEQUENCE {
    controlType   LDAPOID,
    criticality   BOOLEAN DEFAULT FALSE,
    controlValue  OCTET STRING OPTIONAL
}
```

A control type is identified by an Object IDentifier (OID), a string of numbers in the form *number.number....number* [1]. Through a mechanism of registration and delegation controlled by international organisations ISO, ITU and IANA, OID assignment is guaranteed as universally unique. A control has also a *criticality* flag; controls marked as critical must be recognized by the server, that must performs the appropriate action or return an error. Non critical controls, which have the corresponding field set to *false*, can be ignored by the server. The content of the *controlValue* field depends on the control itself.

Proxied Authorization Control [11] is identified by the OID *2.16.840.1.113730.3.4.18* and requires critical flag set to *true*. It can be attached to any *search()*, *compare()*, *modify()*, *add()*, *delete()* or *modRDN()* LDAP operation and its content value is the identifier of the proxied entry. Possible identifier forms can be one of the following [5]:

```
dn: <proxied user dn>
u: <proxied user ID UTF-8 encoded>
```

The first form (*dn:*) directly identifies the entry on behalf which operate. The second one (*u:*) specifies a user identification referred to the underlying authentication mechanism if present, such as external database; this solution results less portable because every LDAP implementation uses a different way to map these types of user identifiers on DIT entry.

Some servers can also support the control with OID *2.16.840.1.113730.3.4.12*, an older and deprecated draft proposal [10] of Proxied Authorization Control.

At the time of writing, Proxied Authorization control is supported at least[3] by OpenLDAP since version 2.1, various renaming of Sun™ Directory Server, Fedora Directory Server, IBM® Tivoli® Directory Server and OpenDS.

Oracle® Internet Directory seems to support a non standard proxy authorization mechanism, identified by the OID *2.16.840.1.113894.1.8.1* [3] that works as a sort of "identity switch" on the connection. Thus, this product will not be further considered.

However, it is possible to dynamically check if a server supports a Control by verifying if Root DSE (*DSA-Specific Entry*), which has an empty DN (""), contains the corres-

[3] Notably exception of servers not supporting Proxied Authorization are current versions of Microsoft Active Directory, Apache Directory Server, Novell eDirectory and Lotus Domino.

```
~$ ldapsearch -h <ldapserver> -s base -b "" \
  "(objectClass=*)" supportedControl
dn:
supportedControl: 2.16.840.1.113730.3.4.18
supportedControl: 2.16.840.1.113730.3.4.2
supportedControl: 1.3.6.1.4.1.4203.1.10.1
supportedControl: 1.2.840.113556.1.4.1413
...
```

Example 3: Query to check Proxied Authorization support using command line tools.

ponding OID string in *supportedControl* attribute, as shown in the Example 3.

Reading Root DSE, LDAP specification does not permit to filter nor compare attribute values directly, because Root DSE is usually generated on-the-fly from server configuration; hence it is not possible to apply a filter as *(supportedControl=2.16.840.1.113730.3.4.18*). Example 4 shows a program snippet to check server capabilities.

```
define("LDAP_PROXIED_CONTROL",
    "2.16.840.1.113730.3.4.18");
$ds = ldap_connect($server);
// Only LDAPv3
// support controls
ldap_set_option($ds,
    LDAP_OPT_PROTOCOL_VERSION, 3);

$sr = ldap_read($ds, "", "(objectClass=*)");
$rootDSE = ldap_first_entry($ds, $sr);
$sc = ldap_get_values($ds, $rootDSE,
    "supportedcontrol");
if (!$sc
    || in_array(LDAP_PROXIED_CONTROL, $sc)) {
  die("Proxied access unsupported");
}
```

Example 4: PHP code snippet to check Proxied Authorization server support.

SERVER CONFIGURATION

In our scenario, we want the service identity *cn=webapp,ou=apps,dc=uni.test* to impersonate users. As other security aspects, also this type of configuration is implemented in different ways on different products.

On Sun Directory Server, the result is achieved by adding the new ACI below, which permits to the service identity to proxy a user.

```
dn: dc=uni.test
aci: (targetfilter=(objectClass=person))
  (version 3.0;acl "Proxy access";
  allow (proxy)
  (userdn=
  "ldap:///cn=webapp,ou=apps,dc=uni.test");)
```

The ACI, reading it from the end, means that the LDAP identity *cn=webapp,ou=apps,dc=uni.test* can act as proxy for each entry belonging to object class *person* starting from the DIT root.

OpenLDAP server is slightly more complex to configure, since it uses the *Simple Authentication and Security Layer* (SASL) framework [8] for all authentication mechanisms except simple bind with clear text password. Therefore proxy authorization requires a special credential configuration. Although SASL permits to user credentials to reside outside of LDAP system, we will consider the usage of the same passwords recorded in LDAP entries. Moreover, SASL uses as default DIGEST-MD5 authentication mechanism, which requires that the server can access to the plaintext password. This can be risky, because it prevents password interception on the wire, but an attack to the server or even a simple administrative routine can reveal users' passwords altogether.

Fortunately, for our usage, we suppose that only few applications need service identity to access LDAP directory, where normal users can use simple bind and be protected by SSL/TLS channel encryption. If all applications identities are registered under branch *ou=apps,dc=uni.test*, mapping between SASL identification and LDAP DN requires these lines in the configuration file:

```
sasl-regexp
  uid=(.*),cn=digest-md5,cn=auth
  cn=$1,ou=apps,dc=uni.test
```

Application will authenticate using its *cn* instead of full DN. After setting a cleartext value in *userPassword* attribute for *cn=webapp,ou=apps,dc=uni.test* entry, we can test configuration with search tool provided with OpenLDAP package:

```
~$ ldapsearch -U "webapp" -W -s base \
  -b "" "(objectClass=*)"
```

The proxy authorization rules processing is disabled by default and can be activated setting a value from Table 1 to the directive *sasl-authz-policy* in *slapd.conf* file.

Table 1: OpenLDAP values for proxy authZ policy.

none	disable proxy authorization (default)
from	enable source rules, proxy authorization is allowed by rules in the proxy entry
to	enable destination rules, proxy authorization is allowed by rules in the proxied entry or in its ancestor
any	proxy authorization is allowed by a source or by a destination rule
all	proxy authorization requires either a source and a destination rule to be allowed

Then permission checking on proxy authorization is performed by using the multivalued attributes *saslAuthzTo* and *saslAuthzFrom*[4] into LDAP entries; both attributes can target DNs using a filter in URL form, a regular expression or a group specification. When assigned to an entry, the first attribute says that the entry can impersonate target DNs, the second says each DN satisfying the filter can impersonate the entry. Both *saslAuthzTo* and *saslAuthzFrom*, as operational attributes, are not automatically returned by LDAP queries. To see their values, you must explicitly require the attributes in the query.

[4] For OpenLDAP 2.3 and newer, directives and attributes cited are named *authz-regexp, authz-policy, authzFrom* and *authzTo* respectively.

In our example, could be sufficient to add:

```
dn: cn=webapp,ou=apps,dc=uni.test
saslAuthzTo: ldap:///dc=uni.test??sub?
  (objectclass=person)
```

to grant the application service identity *cn=webapp, ou=apps, dc=uni.test* to operate as any user, providing that *sasl-authz-policy* enables source rules.

The choice between *saslAuthzTo* and *saslAuthzFrom* depends on several factors. The *saslAuthzTo* attribute enables as proxy only the entry is assigned to, while authorization given by *saslAuthzFrom* is propagated to the descending entries. Moreover, *saslAuthzTo* requires more attention, because if ACIs allow users to write it on their own entries, they can gain more privileges enabling themselves to act as administrative identities.

SENDING PROXIED AUTHORIZATION CONTROL

Classic command line tools have an option to use proxy authorization, but each implementation has a slightly different syntax. For instance, *ldapsearch* can use the following forms, depending on which server it is bundled with (differences are highlighted in boldface):

```
// Sun Directory Server
~$ ldapsearch -Y "dn: <proxied dn>"
// OpenLDAP
~$ ldapsearch -X "dn: <proxied dn>"
// IBM Tivoli Directory Server
~$ ldapsearch -y "<proxied dn>"
```

Next examples will rely to Sun tools syntax, omitting arguments not related with the explained issue. It is worth knowing that there are some incompatibilities in mixing client from a vendor with server from another one, as it will be discussed later.

To test configuration we can proceed as illustrated in Example 5. After preparing a modification file *changes.ldif*, we try to modify *Ada Byron* entry using the web application identity. The operation should fail returning an *Insufficient access* error; none of the rules saved in ACIs enable the web server identity to modify the entry. Then we have to specify to login again as *cn=webapp* but operate on behalf of *cn=Ada Byron* privileges; since *cn=Ada Byron* is entitled to modify her own telephone number, the operation will complete successfully.

```
// Initialize a Proxied AuthZ control
// Generic version
function newProxiedCtrl($dn) {
  return array(
    "oid" => LDAP_PROXIED_CONTROL,
    "iscritical" => true,
    "value" => "dn: " . $dn
  );
}

// Initialize a Proxied AuthZ control
// Sun and Fedora Directory Server version
function newProxiedCtrlSun($dn) {
  $authzid = "dn: " . $dn;
  return array(
    "oid" => LDAP_PROXIED_CONTROL,
    "iscritical" => true,
    "value" => "\x04"
        . chr(strlen($authzid))
        . $authzid
  );
}

$proxyDn = "cn=webapp,ou=apps,dc=uni.test";
ldap_bind($ds, $proxyDn, $proxyPw);

$proxiedCtrl = newProxiedCtrl($proxiedDn);
ldap_set_option($ds,
    LDAP_OPT_SERVER_CONTROLS,
    array($proxiedCtrl));
$newvalues = array(
    "telephoneNumber" => array($newPhone));
ldap_modify($ds, $proxiedDn, $newvalues);

ldap_unbind($ds);
```

Example 6: PHP code snippet to send Proxied Authorization in a modify request.

Example 6 shows as an application can send Proxied Authorization control by adding it to the operation message.

Fedora Directory Server and some version of the Sun implementation suffer of a specification misunderstanding; instead of a string with authorization identity, they require in the field *controlValue* a BER-encoded SEQUENCE, as it was specified in an old draft [10]. With those servers, the request need to be changed as the alternative function in

Example 5: Test Proxied Authorization using command line tools.

```
~$ cat > changes.ldif <<END
> dn: cn=Ada Byron,ou=CS Dept,dc=uni.test
> changetype: modify
> replace: telephoneNumber
> telephoneNumber: +00 100 000 000
> END
~$ ldapmodify -D "cn=webapp,ou=apps,dc=uni.test" -w - -f changes.ldif
ldap_modify: Insufficient access
~$ ldapmodify -D "cn=webapp,ou=apps,dc=uni.test" -w - -f changes.ldif \
  -Y "dn: cn=Ada Byron,ou=CS Dept,dc=uni.test"
modifying entry cn=Ada Byron,ou=CS Dept,dc=uni.test
~$ tail /slapd-test/log/access
 [17:28:07] conn=112 op=0 msgId=1 - BIND dn="cn=webapp,ou=apps,dc=uni.test" method=128 version=3
 [17:28:07] conn=112 op=0 msgId=1 - RESULT err=0 nentries=0 etime=0
 dn="cn=webapp,ou=apps,dc=uni.test"
 [17:28:07] conn=112 op=1 msgId=2 - MOD dn="cn=Ada Byron,ou=CS Dept,dc=uni.test" authzid="dn:cn=Ada
  Byron,ou=CS Dept,dc=uni.test"
~$ ldapsearch -b "cn=Ada Byron,ou=CS Dept,dc=uni.test" "(objectClass=*)" modifiersName
dn: cn=Ada Byron,ou=CS Dept,dc=uni.test
modifiersName: cn=ada byron,ou=cs dept,dc=uni.test
```

Example 7: Different implementations of GetEffectiveRights. Admin checks *cn=Ada Byron* rights on her own entry.

```
# Fedora Directory Server returns entry level and attribute level rights in separate fields
~$ ldapsearch -h fedora.uni.test -D "cn=Directory Manager" -w - \
  -b "cn=Ada Byron,ou=CS Dept,dc=uni.test" \
  -J "1.3.6.1.4.1.42.2.27.9.5.2:true:dn:cn=Ada Byron,ou=CS Dept,dc=uni.test" "(objectClass=*)"
dn: cn=Ada Byron,ou=CS Dept, dc=uni.test
telephoneNumber: +...
...
entryLevelRights: v
attributeLevelRights: telephoneNumber:rscwo, objectClass:rsc, sn:rsc, cn:rsc, userPassword:wo

# Sun Directory Server returns both entry level and attribute level rights in the same field
~$ ldapsearch -h sun.uni.test -D "cn=Directory Manager" -w - \
  -b "cn=Ada Byron,ou=CS Dept,dc=uni.test" \
  -c "dn:cn=Ada Byron,ou=CS Dept,dc=uni.test" "(objectClass=*)" telephoneNumber aclRights
dn: cn=Ada Byron,ou=CS Dept, dc=uni.test
aclRights;attributeLevel;telephoneNumber: search:1,read:1,compare:1,write:1,se
 lfwrite_add:1,selfwrite_delete:1,proxy:0
telephoneNumber: ...
...
aclRights;entryLevel: add:0,delete:0,read:1,write:1,proxy:0

# IBM Tivoli Directory Server returns ACIs and computation of rights is a client task
~$ ldapsearch -h ibm.uni.test -D "cn=root" -w - \
  -b "cn=Ada Byron,ou=CS Dept,dc=uni,dc=test" "(objectClass=*)" ibm-effectiveAcl
dn: cn=Ada Byron,ou=CS Dept,dc=uni,dc=test
ibm-effectiveAcl: group:CN=local admins,OU=CS Dept,DC=uni,DC=test:at.telephoneNumber:rwsc
ibm-effectiveAcl: access-id:CN=THIS:at.telephoneNumber:w
```

Example 6, where *chr(strlen($authzid))* is a workaround which forces the string length in the BER-encoded structure. It is valid only if *$authzid* length is shorter than 127 bytes; dealing with other cases requires a deeper knowledge of ASN.1 encoding rules [6].

Although LDAP controls are sent on per-operation basis, some high level interface to LDAP, such as PHP and Java JNDI, set them up per-connection basis. This behaviour can be prone to side effects when authorization identity can be switched in several application points over the same connection as in complex function frameworks or Java multithread applications [7]. For example, a PHP application can use a service identity to find user DN from *uid* passed by Apache container, an operation usually denied to normal user due to security reasons. Then the application uses the returned DN as proxy authorization identity to update user's profile. If a new search is required, e.g. to update group membership, according to the new profile, it will fail unless authorization identity is not reset, with a new *bind()* operation or recovering a saved connection state.

For some old versions of OpenLDAP, authorization identity switch is one-way. After an operation on behalf of an identity, the DN used for authentication will be lost and the authorization to proxy for a new identity will be checked against the last identity and not the authentication one. Applications that use several identities during one LDAP session will need to re-authenticate each time before changing authorization identity. OpenLDAP since version 2.2 seems not affected by this problem.

LOGGING AND PERFORMANCE ISSUE

The usage of a proxied identity will clearly result in log system as shown in the Example 5. From the point of view of a client application, it would be more interesting if the modifier's identity would results accessible in a standard way, specifically in the entry's *modifiersName* operational attribute. This is the behaviour of OpenLDAP, Tivoli Directory Server and Sun Directory Server 6.x[5]. Instead, Sun Directory Server 5.x and Fedora Directory Server still report the authentication identity as entry modifiers.

As other security features, proxy authorization can have an impact on performances, which is strictly related to how security mechanisms have been implemented.

For example, with OpenLDAP when a LDAP URL is used in *saslAuthzTo* attribute, access verification will enquire for the set of DNs satisfying that URL. Then each returned DN is checked for matching with authorization identity. If search returns a large set, the authorization process can take an annoying long time, especially if the query contains unindexed attributes.

Anyway, proxy authorization results useful especially to control writing operations, which in LDAP systems are usually rare, with negligible total effect.

CAVEATS

Misinterpretations concerning the format of *controlValue* of Proxied Authorization control previously discussed make useless some utilities offered to the programmer by software libraries. Several libraries, such as Netscape/Sun/Mozilla LDAP C client SDK, Perl Net::LDAP library or Java JNDI Booster Pack, offer functions to initialize most commons additional LDAP controls. If both server and client come from the same origin, they will work together without any problem, but using Proxied Authorization on a

[5] By default, newer version of Sun Directory Server records the authentication identity for backward compatibility, but recording of authorization identity can be enabled with parameter *useAuthzIdForAuditAttrs*.

Sun server from a program linking OpenLDAP libraries, such as PHP, or *vice versa,* is a tricky task. The best solution is to initialize the control structure from scratch as previously shown. The choice of sending one or other version of the control can be defined in the application configuration or the application can exploits a sort of "server sniffing"; in fact, Root DSE usually contains two attributes *vendorName* and *vendorVersion* that can be used to detect the producer and the version of the LDAP server.

The main disadvantage of relying on LDAP access control system instead of rebuilding it from the ground into middle-tier application is that there is not a convenient way an application can use to make its user interface reactive to the user capability. For instance, there is nothing in basic LDAP which helps to enable or disable a contextual menu item that invokes a field modification, e.g. inserting a new telephone number, according to the user capability of effectively updating it. The user should try and expect an error if the operation fails because of insufficient privileges.

This issue also affects directory management activities, because admins need a tool to verify the correctness of access control configurations. To cope with this problem, a standard instruction, called *GetEffectiveRights,* was proposed in the past [9]. The proposal never passed the draft status, but some vendors have implemented an equivalent functionality, unfortunately in reciprocally incompatible ways. As result, this task requires a lot of "ad-hoc" solutions.

Either Sun Directory Server and Fedora Directory Server support an additional control both named "GetEffectiveRights" with OID *1.3.6.1.4.1.42.2.27.9.5.2*. They work under the same general concept: by adding GetEffectiveRights control to a *search()* operation, results will report additional operational attributes which describe the effective right on entries returned. But all other details are implemented in a quite different way, as shown in Example 7.

Fedora Directory Server requires the subject identity for which compute effective rights is specified in the value of GetEffectiveRights control in the same way of DN identification in Proxied Authorization control (but spaces between "*dn:*" and DN string seems to cause troubles). Then, it returns effective rights on targets in attributes *entryLevelRights*, for entry level privileges, and *attributeLevelRights*, for attribute level privileges.

Sun Directory Server admits a null value in GetEffectiveRights, in which case effective rights are evaluated using the bind DN as subject. But, if a valued should be passed, it must be in the form of BER-encoded SEQUENCE of strings. Resulting attribute is named *aclRights* and reports both entry and attribute level privileges the subject own on the target. For diagnostic purpose, another attribute, *aclRightsInfo* explains how effective rights have been computed.

IBM Tivoli Directory Server offers a similar feature consisting in an operational attribute named *ibm-effectiveAcl*, that reports the effective ACL for an entry, where "effective" means all accumulated access controls that applies to the target object based on how ACIs have been distributed in the DIT. The main difference[6] compared to Fedora and Sun servers is they report privileges of an entry over another one as computed by the server, whereas IBM Tivoli reports ACIs, thus effective privileges must be resolved by client. This task needs other queries and can require higher privileges to be accomplished.

Usually, a normal user can only retrieve effective rights on his personal entry. To get effective rights on target entries different from own identity an administrative level account is required. Moreover, the computation of effective right is in general quite expensive, and therefore it should be limited only when effectively needed.

CONCLUSIONS

It is a common practice to develop multi-tier and web-based applications that connect to databases, directory or other resources using privileged account. How many web applications still use *sa*, *dba* or an equivalent account to access to database? Sometimes the technology involved offers no other choice, sometimes it is required because the middle-tier must perform operation not allowed to normal users, often it can be avoided but alternatives are not well known.

Many LDAP implementations offer a feasible mechanism as explored in this article. This mechanism delivers a wide range of security benefits: commits to LDAP system all control on the roles the middle-tier can assume for the user, preserves the identity of the real user through to the middle-tier, and enables auditing of actions taken on behalf of real users.

ACKNOWLEDGMENTS

Thanks to Marcello Cerruti and Marina Ribaudo for feedback and suggestions.

Figure 2 has been produced using a customized version of JXplorer, http://www.jxplorer.org/, using icons from David Vignoni's Nuvola theme.

REFERENCES

[1] *Abstract Syntax Notation One (ASN.1)* "Information Object Specification", ITU-T Rec. X.681 (2002)

[2] *OpenDS project wiki* "Compatibility With Sun Java System Directory Server (revision-4)" available at https://www.opends.org/wiki//page/CompatibilityWithSunJavaSystemDirectoryServer

[3] *Oracle Identity Management Application Developer's Guide 10g*, Oracle Part Number B15997-01 (2006), available at

[6] The access control system and other details of Tivoli Directory Server are quite different from those in other products. The effort of design a service equivalent to the example scenario under Tivoli was beyond the purpose of the article.

http://download.oracle.com/docs/cd/B28196_01/id-manage.1014/b15997/ext_ldap.htm

[4] *Sun ONE Directory Server 5.2 Administration Guide*, Sun Microsystems, Inc., ISBN 0595732232 (2003), available at http://docs.sun.com/app/docs/doc/816-6698-10 .

[5] Harrison R., *Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms*, IETF RFC 4513 (2006).

[6] Larmouth, J., *ASN.1 Complete*, Morgan Kaufmann Publishers, ISBN 0-12233-435-3 (1999).

[7] Lee R., The JNDI Tutorial, Sun Microsystems, Inc. (2002), available at http://java.sun.com/products/jndi/tutorial/index.html

[8] Myers J. G., *Simple Authentication and Security Layer (SASL)*, IETF RFC 2222 (1997)

[9] Stokes E., Blakley B., Byrne R., Huber R. and Rinkevich D., *Access Control Model for LDAPv3*. IETF draft (2001), draft-ietf-ldapext-acl-model-08.txt.

[10] Weltman R., *LDAP Proxied Authorization Control*. IETF draft (2000), draft-weltman-ldapv3-proxy-05.txt.

[11] Weltman R., *Lightweight Directory Access Protocol (LDAP) Proxied Authorization Control*, IETF RFC 4370 (2006).

[12] Zeilenga K., *Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map*, IETF RFC 4510 (2006)

[13] Zeilenga K., *Lightweight Directory Access Protocol (LDAP): Directory Information Models*, IETF RFC 4512 (2006).