Perl for System Administration

Networking Power Hour: LDAP

David N. Blank-Edelman
Northeastern University CCIS
USENIX 2003
Directories

• Not your grandparents’ phone books any more!
  – Many kinds of directories (phone/email/PKI)
  – Resource discovery
  – User policy information (Active Directory)
• Specially tuned networked databases:
  – Optimized for read-many, write few traffic patterns
  – Simple queries, simple storage
  – Hierarchical access models

What is the Lightweight Directory Access Protocol?

• Defined in:
  – v3: RFCs 2251-2256, 2829, 2830, 3377
  – LDIF: RFC2849
  – inetOrgPerson: RFC2798
  – NIS: RFC2307
  – SASL: RFC2222
  – Controls (more info later)
• But that’s not the answer you really want…
LDAP Terminology

Begin with an Entry

- Has a name (important! more info soon)
- Set of attributes
- Attributes are:
  - Set of type-value(s) pairs
    (some types can have multiple values)
More About Types

- Type is:
  - OID & description
  - Multi-valued setting
  - Syntax (RFC2522)
  - Matching rules

```plaintext
# from OpenLDAP core.schema
attributetype ( 0.9.2342.19200300.100.1.25
  NAME ('dc' 'domainComponent')
  DESC 'RFC1274/2247: domain component'
  EQUALITY caseIgnoreIA5Match
  SUBSTR caseIgnoreIA5SubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 SINGLE-VALUE )
(SYNTAX URL: http://www.alvestrand.no/objectid/1.3.6.1.4.1.1466.115.121.1.26.html
or: http://ldap.akbkhome.com/syntax/1.3.6.1.4.1.1466.115.121.1.26.html)
```

objectClass & schema

- Every entry has to have an objectClass attribute
- Dictates what attributes MUST and MAY be in that entry
- Can inherit into definition from other classes
- Schema = attribute types, objectClass(es), etc. used to determine how search/compare/add/modify operations work or don't work

```plaintext
objectclass ( 2.5.6.6 NAME 'person'
  DESC 'RFC2256: a person'
  SUP top STRUCTURAL
  MUST ( sn $ cn )
  MAY ( userPassword $ telephoneNumber $ seeAlso $ description ) )

objectclass ( 2.5.6.7 NAME 'organizationalPerson'
  DESC 'RFC2256: an organizational person'
  SUP person STRUCTURAL
  MAY ( title $ x121Address $ registeredAddress $
    destinationIndicator $ preferredDeliveryMethod $
    tel exNumber $ teletexTerminalIdentifier $
    telephoneNumber $ internationaliSDNNumber $
    facsimileTelephoneNumber $ street $ postalOfficeBox $
    postalCode $ postalAddress $ physicalDeliveryOfficeName $
    ou $ st $ ) )
```

4
Tree Time

- Directory Information Tree
- Every entry in a DIT has a Relative Distinguished Name
- RDN unique on that level
- DN == collection of RDNs from current level to top (like a postal address), for example:

1. cn=Robert Smith, l=main campus, ou=CCS, o=Hogwarts School, c=US
2. uid=rsmith, ou=systems, ou=people, dc=ccs, dc=hogwarts, dc=edu

More Tree Stuff

- Naming context/suffix
- Root DSE
  (DSE = DSA-specific Entry
  (DSA = Directory System Agent
  (Directory System Agent = server)))
- Aliases
- Referrals (uses LDAP URIs, more later)
Bring on the Perl Code!

- Ok, but we’ll need to sneak some more terminology in as we go...

```
use Net::LDAP;
$c = Net::LDAP::new($server, port => $port,
              version => 3) or
die "Unable to connect to $server: $@
;#
use no parameters to bind() for anonymous bind
$c->bind($binddn, password => $passwd) or
die "Unable to bind: $@
;
...
$c->unbind();
```

- Three flavors of binds: simple, SASL, anonymous
- SASL EXTERNAL allows for SSL/TLS/LDAPS
- RFC2252/2307 define a userPassword attribute
  - e.g. userPassword: {crypt}X5/ bdDiEfSS2F
Simple Search Background

- Basedn – where we start
- Scope – where we look
  - base, one, sub
- Filter – what we want
  - Remember your LISP?

Search Filters

- Examples (from RFC2254):
  - (cn=Babs Jensen)
  - (!((cn=Tim Howes))
  - (&(objectClass=Person)((!(sn=Jensen)(cn=Babs J*))))
  - (o=univ*of*mich*)
- Comparison operators: = =* ~= >= <= *
- Logical operators: & | !
- attr[:dn][:mrule]:=value or [:dn]:mrule := value
  - (cn:1.2.3.4.5:=Fred Flintstone)
  - (sn:dn:2.4.6.8.10:=Barney Rubble)
  - (o:dn:=Ace Industry)
  - (:dn:2.4.6.8.10:=Dino)
- Must quote: * ( ) \ NUL
Search from the Command Line

• ldapsearch
  – -h hostname
  – -x (for simple bind)
  – -D binddn
  – -w password
  – -b basedn
  – -s scope
  – (filter)
  – attr attr attr...

Search from Perl

use Net::LDAP;
# bind here...
$searchobj = $c->search(base => $basedn,
                        scope  => $scope,
                        filter => $filter);

die "Bad search:" . $searchobj->error() if $searchobj->code();

• Other modifiers for the search:
  - sizelimit => # entries
  - timelimit => # secs
  - typesonly => T/F
  - attrs => ['name','name','...'] or ['*','opname','opname','...']
Searchobj Operations

# return a specific entry number
$entry   = $searchobj->entry($entrynum);

# acts like Perl shift() on entry list
$entry   = $searchobj->shift_entry;

# acts like Perl pop() on entry list
$entry   = $searchobj->pop_entry;

# return all of the entries as a list
@entries = $searchobj->entries;

# return the attributes of a specific entry
@attributes = $entry->attributes;

# get a specific value from the first entry
$value   = $searchobj->entry(1)->get(cn)

Referrals & References

• Uses LDAP URLs (defined in RFC2255):

ldap://host.com:6666/o=Univ%20of%20Michigan,c=US??sub?(cn=Babs%20Jensen)

use Net::LDAP::Constant qw(LDAP_REFERRAL);
use URI;

# bind and search here...
my @referrals = $searchobj->referrals
if ($searchobj->code() == LDAP_REFERRAL);

my @references = $searchobj->references;
foreach my $refrl (@referrals){
  my $uri = URI->new($refrl);
  my $dn = $uri->{dn}; # ->filter, ->scope, ...
  # go forth and do something with this...
}
Compare

- Uses matching rules from earlier
- Useful for things like passwords

```perl
use Net::LDAP;
use Net::LDAP::Constant qw(LDAP_COMPARE_TRUE);

# bind here...
my $result = $c->compare($dn,
    attr  =>'userPassword',
    value => $passwd);

die "bad pwd" if ($result->code() != LDAP_COMPARE_TRUE);
```

Adding Entries – Bare Metal

```perl
use Net::LDAP;

# bind here...
$res = $c->add(
    dn => 'uid=jay, ou=systems, ou=people, dc=ccs,
    dc=hogwarts, dc=edu',
    attr => [ 'cn'  => 'Jay Sekora',
              'sn'  => 'Sekora',
              'mail' => 'jayguy@cs.hogwarts.edu',
              'title' => [ 'Sysadmin','Lecturer' ],
              'uid'  => 'jayguy',
    ]);

die "add error:" . $res->error() if $res->code();
```
Adding Entries – OOP

```perl
use Net::LDAP;
use Net::LDAP::Entry;

# bind here... get back $c
my $entry = Net::LDAP::Entry->new();
$entry->dn('uid=jay, ou=systems, ou=people,
   dc=cs, dc=hogwarts, dc=edu');
$entry->add('cn' => 'Jay Sekora', 'sn' => 'Sekora',
   'mail' => 'jayguy@cs.hogwarts.edu',
   'title' => ['Sysadmin', 'Lecturer'],
   'uid' => 'jayguy');
$res = $c->update($entry);
die "add error: ". $res->error() if $res->code();
```

Delete and Rename Whole Entries

- **Delete:**
  $res = $c->delete($dn);

- **Rename:**
  $res = $c->moddn($oldDN,
   newrdn => $newRDN,
   del eteol drdn => 1);
Entry Modification

```perl
$searchobj = $c->search(base => $basedn,
    filter => "(l=Boston)",
    scope => 'sub',
    attrs => [''],
    typesonly => 1);

if $searchobj->code();
    die "Bad search: " . $searchobj->error();

if ($searchobj){
    @entries = $searchobj->entries;
    for (@entries){
        $res=$c->modify($_->dn(),
            delete => {"l" => "Boston"},
            add => {"l" => "Indiana"});
        warn "Bad modify: ". $res->error() if $res->code();
    }
}
```

Dealing with Groups

- **Static (RFC2256)**
  - objectType is:
    - `groupOfUniqueNames`
    - `groupOfNames` (old)
  - Attribute is:
    - `uniquemember`
    - `member` (old)
  - Value is:
    - DN (person or group)

- **Dynamic**
  - (Netscape/iPlanet/Sun One)
  - objectType = `groupOfUrls`
  - attr = `memberurls`
Leftovers: Bulk Additions

- LDIF
  - Defined in RFC2849

version: 1
# Add a new entry
dn: cn=Fiona Jensen, ou=Marketing, dc=airius, dc=com
changetype: add
objectclass: top
objectclass: person
objectclass: organizationalPerson
cn: Fiona Jensen
sn: Jensen
uid: fiona
telephonenumber: +1 408 555 1212

LDIF from Perl

use Net::LDAP::LDIF;

# can also pass new() an open filehandle
$ldifobj = Net::LDAP::LDIF->new("fn", "w",
  encode=>'canonical');
$ldifobj->write_entry(@entries);
$ldifobj->done();

OR

$ldifobj = Net::LDAP::LDIF->new("fn", "r");
until ($ldifobj->eof()) {
  $entryobj = $ldifobj->read_entry();
}
$ldifobj->done();
DSML

- www.oasis-open.org/committees/dsml/
- “still in development”
- Like using LDIF, with a few changes:

```perl
use Net::LDAP::DSML;
open(FH, ">filename") or die $!
$dsmlobj = Net::LDAP::DSML->new (output => *FH,
  pretty_print => 1);
$dsmlobj->start_dsml; # not documented yet
$dsmlobj->write_entry(@entryobjs);
$dsmlobj->end_dsml;   # not documented yet
close(FH);
```

Advanced Leftovers: Controls

- “adjectives” for operations (G. Carter’s definition)
- Extensions to basic protocol, not all servers implement even RFC’d ones (e.g. OpenLDAP)
- Examples supported in Net::LDAP:
  - Simple paged results (RFC2696)
  - Proxied Authorization Control
    (draft-weltman-ldapv3-proxy-11.txt)
  - Server Side Sorting of Search Results (RFC2891)
  - Scrolling View Browsing of Search Result/VLV
    (draft-ietf-ldapext-ldapv3-vlv-08.txt)
Controls from Perl

- Most commands take a control argument
- Arguments can be simple, or constructed for you

```perl
use Net::LDAP::Control::Sort;
use Net::LDAP::Constant qw(LDAP_CONTROL_SORTRESULT);

$sort = Net::LDAP::Control::Sort->new(
    order => "cn-phone",
);

$mesg = $ldap->search( @args, control => [ $sort ]);

($resp) = $mesg->control( LDAP_CONTROL_SORTRESULT );
print "Results are sorted\n"
    if $resp and !$resp->result;
```

Advanced Leftovers: rootDSE & Schema

```perl
# bind and then
$entryobj = $c->root_dse();

- Get back an entry with these attributes:
  - subschemaSubentry, namingContexts, altServer,
    supportedExtension, supportedControl,
    supportedSASLMechanisms, supportedLDAPVersion

$schemaobj = $c->schema();

- Call methods for schema information:
  - must()/may(), all_{attributes, matchingrules, syntax,
    objectclasses,}, attribute("name/oid"), …
```