U-Bolt: Campus Identity Integration for Decentralized Systems Webinar

David Champion
Computation Institute, Enrico Fermi Institute
US ATLAS, UC3
University of Chicago
UC3 Identity

Where it started
UC3 Identity Goals

• UC3 is an open platform for connecting research to distributed HTC resources across campus.
  » Condor cluster that can flock to other Condor clusters on campus
  » 4-5 other facilities on campus, upwards of 10,000 job slots accessible

• Users could be anyone on campus.

• Users should be validated as legitimate campus personnel.

• Shared facilities should have a common basis for identifying owners of data and users of resources.
UC3 Identity Goals

• We want to get potential users online:
  1. quickly
     » minimally operational within 60 minutes
  2. simply
     » use existing connection tools and identity frameworks
  3. cheaply
     » no new username and passwords
     » no complicated registration process, when the University already knows all users

• Campus LDAP is the obvious solution, but not all the pieces were there. To integrate our local access, we needed to improvise.
UC3 Onboarding Demo
Login Failure

bash ttys007 09:39:41 ~ [3/0]:
bash ttys007 09:39:42 ~ [3/0]: ssh uc3-sub.uchicago.edu
You must be a uniqueMember of cn=uc:org:uc3:users,ou=groups,dc=uchicago,dc=edu to login.
Connection closed by 128.135.158.243
bash ttys007 09:39:54 ~ [4/0]:
U-Bolt
What is U-Bolt?

• U-Bolt is identity integration middleware for the campus

• it aims to be a flexible toolkit for addressing identity integration problems for distributed environments embedded within, or with access to, larger campus infrastructures
What U-Bolt Provides

• U-Bolt currently consists of two NSS modules to address problems arising from limitations in a campus LDAP environment
  1. nss_identity to provide forward and reverse group mapping for artificial groups not in LDAP
  2. nss_filter to provide unique home directory mapping when LDAP does not
     » nss_filter also allows optional mapping of pw_gecos and pw_shell

• It is a work in progress; contributions are welcomed
Measures of Completion

• U-Bolt is already a success in that it has addressed identity integration for our site

• Major objective is to be able to piggyback on any LDAP or AD authentication service without any attribute visibility whatsoever:
  - no uid • no gid • no gecos • no home directory • no shell

• However, when such attributes are visible, we should use them
Account Services
Technical Overview
How does it work?
1. Two key service departments:
   a. Nameservice Switch (NSS)
      • *identity* service (ID): what users exist?
      • *directory* service (DS): what are a user’s attributes (uid, home directory, shell, groups, etc)
      • limited *authorization* controls
         » existence = access
         » valid shell = access
1. Two key service departments (cont’d):
   b. Pluggable Authentication Modules (PAM)
      • **authentication** service (authN): is this user (agent) who she (it) claims to be?
      • **authorization** service (authZ): is this user permitted to use this system?
2. NSS and PAM are both generalized frameworks implemented within libc

3. Both are extensible under a plugin architecture
   a. plugins are dynamic objects (DLL/DSO) that load into the address space of the process that needs their service
   b. that process can be any program that calls into the NSS or PAM framework
   c. most commonly that process for NSS is nscd, which proxies/caches lookups for other processes
   d. PAM modules are loaded directly by e.g. sshd, httpd
Study Example: Basic local ssh login

1. sshd prompts for username and password

2. sshd looks up user via getpwnam(), a libc function
   a. getpwnam(), passes request to nsswitch (nss) framework
   b. nss checks /etc/nsswitch.conf
   c. nsswitch loads libnss_files.so to resolve request
   d. libnss_files.so:_getpwnam_r() resolves name via /etc/passwd
   e. struct passwd is constructed (almost like /etc/passwd)

```
alice:$1$YeNsbWdH$wvOF...:2049:500:Alice:/home/alice:/bin/bash
```
3. `sshd` asks PAM to validate user using the password

   a. PAM checks `/etc/pam.conf`, `/etc/pam.d/*` to find the applicable module stack

   b. `pam_unix.so` retrieves `struct passwd` from NSS; it contains a DES, MD5, or SHA hashed password

   
   `alice:$1$YeNsbWdH$wvOF...:2049:500:Alice:/home/alice:/bin/bash`

   c. `pam_unix.so` validates the password by hash compare, and returns success value to `sshd`

4. if authN succeeded, other PAM modules may refuse login on a policy basis (authZ)
1. Minimum requirements:
   a. bind as user; read user attributes as user
   b. cn or uid contains a unique identifier
   c. all users in a single directory service
      » it’s technically possible to work around this, but it’s difficult, and there are multiple risks to negotiate
   d. everything else is on you, and not all tools exist for making the translation (but they are feasible to create)
2. Ideal scenario (cumulative):
   a. all of theposixAccount MUST attributes: `cn`, `uid`, `uidNumber`, `gidNumber`, `homeDirectory`
   b. two of theposixAccount MAY attributes: `loginShell`, `gecos`
   c. sensible and distinct values for `uidNumber` and `homeDirectory`
   d. group mapping from `gidNumber` to group names

Your situation is probably somewhere between minimum and ideal.
1. If your site has all the requirements of the ideal scenario, *all you need may be to convert to LDAP!*
   
   » Active Directory is a special case of LDAP, and can also work.

2. NSS must be augmented:

   a. nsswitch *stacks*: if a user is not found in the first listed plugin, it falls through to the second, third, fourth...

   ```
   # /etc/nsswitch.conf
   passwd: files ldap
   group: files ldap
   hosts: files ldap
   hosts: files dns
   ...
   ```

   b. users not listed locally can now be found in LDAP
      (configuring LDAP access not described here)
Use Case: Moving to LDAP/AD

3. PAM must be updated:

a. `/etc/pam.d/system-auth` (Red Hat derivatives); `/etc/pam.d/sshd` (other):

<table>
<thead>
<tr>
<th>auth</th>
<th>required</th>
<th>pam_env.so</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth</td>
<td>sufficient</td>
<td>pam_unix.so nullok try_first_pass</td>
</tr>
<tr>
<td>auth [default=ignore success=done]</td>
<td>pam_ldap.so use_first_pass</td>
<td></td>
</tr>
<tr>
<td>auth</td>
<td>requisite</td>
<td>pam_succeed_if.so uid &gt;= 500 quiet</td>
</tr>
<tr>
<td>auth</td>
<td>required</td>
<td>pam_deny.so</td>
</tr>
</tbody>
</table>

- this tells PAM to try standard UNIX authentication (hashed passwords); if that doesn’t work, then try binding to LDAP as the candidate user using the password presented

b. if you need Active Directory authentication, use `pam_winbind.so` instead of `pam_ldap.so`, and separately configure Samba’s `winbindd`. 
4. What about home directories?
   a. if your users’ home directories are pre-created (e.g. reside on an extant network share), you’re done
   b. otherwise, you can add PAM configuration to create home directories on demand. Two options:
      • `pam_mkhomedir.so`: copies a skeleton home from `/etc/skel` when home is absent
        ‣ maximally efficient, completely rigid
      • `pam_exec.so`: executes an arbitrary script as root for each login; this script can check whether a home is needed and create it in any arbitrarily complex fashion
        ‣ very inefficient, infinitely flexible
1. Mixing local and directory-sourced users is an old problem.
   
a. old NIS (SUN) system managers will recall this pattern:

   ```
   root:x:0:0:System Administrator:/root:/bin/sh
   bin:x:1:0::/:/bin/false
   alice:x:2049:500:Alice:/home/alice:/bin/tcsh
   +bob:::::::
   +::::::::/bin/nologin
   ```

b. the + lines means: map into this system’s user list a user who appears in my (NIS) directory service, and treat them as though they were local

c. a blank field is inferred from the directory

d. an explicit field supersedes the one in the directory
Use Case: Restricted LDAP integration

2. nss_compat brings this to a modern nsswitch — *whether using NIS or any other directory service*

a. suppose the following:

```plaintext
# /etc/nsswitch.conf
password: files ldap
group: files ldap
hosts: files dns
...
```

```plaintext
# /etc/passwd
root:x:0:0:System Administrator:/root:/bin/sh
bin:x:1:0::/bin/false
alice:x:2049:500:Alice:/home/alice:/bin/tcsh
+bob::::::::
+:::/bin/nologin
```

b. all LDAP users are now known to the local system, but only “bob” may log in
How can we tap into this?

Your site is probably not so lucky. But how can you extend this to make it work for you?

- NSS and PAM together provide an extremely flexible mechanism for semi-arbitrary governance of identity, authentication, and macro-level authorization.
- You can insert authNZ policies and mechanisms into PAM using a simple API.
- You can insert directory services or wrappers for directories into NSS using an even simpler API.
- The plumbing is there; all you need is code.
  (but documentation and examples are very limited)
Case Study: UC3
Recall our required and preferred components:

- bind as user; read user attributes as user
- cn or uid contains a unique identifier
- all users in a single directory service
- all of the posixAccount MUST attributes: cn, uid, uidNumber, gidNumber, homeDirectory
- two of the posixAccount MAY attributes: loginShell, gecos
- sensible values for uidNumber and homeDirectory
- group mapping from gidNumber to group names
Recall our required and preferred components:

- bind as user; read user attributes as user
- cn or uid contains a unique identifier
- all users in a single directory service
- all of the posixAccount MUST attributes: cn, uid, uidNumber, gidNumber, homeDirectory
- two of the posixAccount MAY attributes: loginShell, gecos
- sensible values for uidNumber and homeDirectory
- group mapping from gidNumber to group names
UC3’s Advantages

Authentication and authorization:

☑ bind as user; read user attributes as user
☑ cn or uid contains a unique identifier
☑ all users in a single directory service

➡ We can use pam_ldap for authentication
➡ Authorization works through pam_ldap also — more on this later
UC3’s Advantages

☑ all of the posixAccount MUST attributes: cn, uid, uidNumber, gidNumber, homeDirectory

☑ two of the posixAccount MAY attributes: loginShell, gecos

☑ sensible values for uidNumber

→ nss_ldap gets us most of the way to an identity and directory solution
UC3’s Challenges

Two problems to solve:

1. no POSIX groups in LDAP
   » user’s gidNumber does not reverse map to any group name.

2. homeDirectory value is an artifact of a past age:
   /nfs/harper/ha0/usernameA
   /nfs/harper/ha1/usernameB
   /nfs/harper/hb0/usernameC ...
   » We want to replace with a simple /home/username.

Each of these is a case for an NSS plugin.
UC3: Solving gidNumber

The problem:

1. In uchicago LDAP, gidNumber == uidNumber.
2. In uchicago LDAP, there are no POSIX groups.

```
$ id -a
uid=2052(dgc) gid=2052
$ getent group 2052
(no result)
$ ls -ld ~
drwxr-x--x 135 dgc 2052 421 2013-02-21 12:03 /nfs/harper/hc0/dgc
```
The solution:

1. `nss_identity.so` is an NSS module to manufacture such groups on demand. Pseudocode:
   a. `getgrgid(gid=2052) ⇒`
      (i) `gid = uid = 2052`
      (ii) `pw = getpwuid(uid=2052)`
      (iii) `return new group(gid=2052, name=pw.name)`
2. Added to nss stack:

```bash
# /etc/nsswitch.conf
passwd: files ldap
group: files identity
hosts: files dns
...
```

3. Done!

```bash
$id -a
uid=2052(dgc) gid=2052(dgc)
$ getent group 2052
dgc::2052:dgc
$ ls -ld ~
drwxr-x--x 135 dgc dgc 421 2013-02-21 12:03 /nfs/harper/hc0/dgc
```
UC3: Solving homeDirectory

A little more complex. The problem:

1. In uchicago LDAP, homeDirectory is a valid and distinct value, and technically works, but

2. It’s ugly, unpredictable, a bit surprising and confusing to users.

   $ cd; pwd
   /nfs/harper/hc0/dgc
   $ getent passwd dgc
dgc:x:2052:2052:David Champion:/nfs/harper/hc0/dgc:/bin/bash

3. We can’t just stack onto NSS, because we receive and use the rest of the nss_ldap response.
UC3: Solving homeDirectory

The solution:

1. `nss_compat.so` is an NSS passthrough or proxy module. We can use this technique. (Twice!)

   ```
   # /etc/nsswitch.conf
   passwd: files compat
   passwd_compat: ldap
   ```

a. `getpwnam(name=“dgc”)` checks with `nss_files`
b. `getpwnam(name=“dgc”)` cascades to `nss_compat`
c. `nss_compat` makes a “back door” call through `nss_ldap`
d. `nss_ldap` returns dgc’s LDAP POSIX account
e. `nss_compat` updates this data and returns it
1. `nss_filter.so` is an NSS passthrough or proxy module that we compound with `nss_compat.so`. This is unusual, but it solves our problem exactly.

```bash
# /etc/nsswitch.conf
passwd: files filter
passwd_filter: compat
passwd_compat: ldap

# /etc/passwd
root:x:0:0:System Administrator:/root:/bin/sh
bin:x:1:0:::/bin/false
alice:x:2049:500:Alice:/home/alice:/bin/tcsh +::::::/home/&:
```
UC3: Solving homeDirectory

2. Logic flow:
   a. `getpwnam(name="dgc")` checks with `nss_files`
   b. `getpwnam(name="dgc")` cascades to `nss_filter`
   c. `nss_filter` makes a “back door” call through `nss_compat`
   d. `nss_compat` makes a “back door” call through `nss_ldap`
   e. `nss_ldap` returns dgc’s LDAP POSIX account, with `pw_dir="/nfs/harper/hc0/dgc"`
   f. `nss_compat` replaces with `pw_dir="/home/&"`
   g. `nss_filter` replaces `/home/&` with `/home/dgc` and returns the full passwd entry
UC3: Solving homeDirectory

2. Added to nss stack:

```plaintext
passwd: files filter
passwd_filter: compat
passwd_compat: ldap
```

3. Added to /etc/passwd:

```plaintext
+:-----:/home/&:
```

4. Done!

```plaintext
$ cd; pwd
/home/dgc
$ getent passwd dgc
dgc:x:2052:2052:David Champion:/home/dgc:/bin/bash
```
UC3 Onboarding Demo
Integrating Your Site
Integration Profile (redux)

Checklist of required and preferred components:

☐ bind as user; read user attributes as user
☐ cn or uid contains a unique identifier
☐ all users in a single directory service
☐ all of the posixAccount MUST attributes: cn, uid, uidNumber, gidNumber, homeDirectory
☐ two of the posixAccount MAY attributes: loginShell, gecos
☐ sensible values for uidNumber and homeDirectory
☐ group mapping from gidNumber to group names
## What ideals are you missing, and what do you do?

<table>
<thead>
<tr>
<th>Missing feature / Problem</th>
<th>How you handle it</th>
<th>Works Today?</th>
</tr>
</thead>
<tbody>
<tr>
<td>• cn or uid missing</td>
<td>talk to IdM - you need one of these</td>
<td></td>
</tr>
<tr>
<td>• cannot bind as user</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• loginShell is not present or not useful</td>
<td>supersede with nss_compat</td>
<td>✓</td>
</tr>
<tr>
<td>• gecos is not present or not useful</td>
<td>configure nss_ldap to use cn or displayName instead</td>
<td>✓</td>
</tr>
<tr>
<td>• gidNumber does not map to a named group</td>
<td>not strictly required, but you can map it in the nsswitch stack using nss_identity</td>
<td>✓</td>
</tr>
<tr>
<td>• homeDirectory is not sensible or not unique</td>
<td>mapped through multiple layers of nsswitch: nss_compat to supersede pw_home, with nss_filter to perform user substitutions</td>
<td>✓</td>
</tr>
<tr>
<td>• uidNumber is not sensible or not unique</td>
<td>You can’t work with this. Ignore it and resolve as if there’s no uidNumber.</td>
<td>←</td>
</tr>
<tr>
<td>• no gidNumber</td>
<td>either supersede with nss_compat, or treat like missing uidNumber</td>
<td></td>
</tr>
<tr>
<td>• no uidNumber</td>
<td>need to manufacture this on demand; this will in turn require stateful user caching</td>
<td>×</td>
</tr>
</tbody>
</table>
Solving loginShell

Problem:

- loginShell is not present or not useful in LDAP

Solution:

1. use nss_compat to supersede locally

```sh
# /etc/nsswitch.conf
passwd: files compat
passwd_compat: ldap

# /etc/passwd
root:x:0:0:System Administrator:/root:/bin/sh
bin:x:1:0:::/bin/false
alice:x:2049:500:Alice:/home/alice:/bin/tcsh
+::=/bin/bash
```
Solving gecos

Problem:

- gecos is not present or not useful in LDAP

Solution:

1. configure nss_ldap to use cn or displayName instead

   » usually nss_ldap shares configuration with openldap and pam_ldap
   » configuration file is typically /etc/ldap.conf or /etc/openldap/ldap.conf
   » see nss_ldap(5), RFC2307

```
# /etc/ldap.conf
# For generic LDAP, map cn onto gecos
nss_map_attribute gecos cn
# For Active Directory, map displayName onto gecos
nss_map_attribute gecos displayName
```
Solving gidNumber

Problem:
  • gidNumber does not map to a named group in LDAP

Solution:

1. map in nsswitch using nss_identity
   » see UC3 case study

```bash
# /etc/nsswitch.conf
passwd: files ldap
group: files identity
hosts: files dns
...
```
Solving homeDirectory

Problem:
  • homeDirectory is not sensible or not unique

Solution:
  1. supersede in nsswitch using nss_compat and nss_filter
     
        » see UC3 case study

        # /etc/nsswitch.conf
        passwd: files filter
        passwd_filter: compat
        passwd_compat: ldap

        # /etc/passwd
        +++++:/home/&:
Solving gidNumber

Problem:

- gidNumber is not present in LDAP

Solution:

1. this may be a small problem that can be solved with nss_compat supersession

```
# /etc/group
users::1001:
```

```
# /etc/passwd
+::1001::
```

2. otherwise this is akin to solving uidNumber; see below
Future Challenge: Solving uidNumber

Problem:

- uidNumber is not present in LDAP

Solution:

1. forward development in U-Bolt will address this:
   a. an nss_http module bind to an HTTP-based directory service
      
         » GET /ubolt/0.1/passwd/byuid/2052
         
            
      
      b. U-Bolt will provide a plugin-based reference implementation that can be tuned or extended to meet local needs
      c. service will provide stateful storage for manufactured data
      d. can be run locally or centrally
Why the HTTP approach?

• We want to simplify client configuration as much as possible, while providing solutions to any problem sites are likely to encounter

1. There are two approaches to this:
   a. let the client talk to an extant DS (e.g. LDAP) and teach that extant DS to incorporate complexity
      » this either involves a lot of continuous feed processing — the kind of thing we’d need extensive cooperation from IdM to do — or hacks to provide configurable backends to an LDAP/NIS frontend
   b. invent a shim for the client that lets us talk to a DS that anyone can implement, or that they can borrow from us and adjust

2. The latter is simpler and more maintainable in the long term than trying to graft complex dynamic backends onto code projects (e.g. OpenLDAP) managed by an upstream host.
Why the HTTP approach?

• We need a protocol for the exchange between the nss module and the service. HTTP is:

  1. widely implemented
     » anyone can build their own service, or use our reference implementation

  2. scalable as needs change
     » can run as a local standalone service, or under Apache, etc.

  3. easily extensible
     » structure, scope, and hierarchy already present in standard HTTP WS idioms
Questions?

https://uc3.uchicago.edu/

https://wiki.uchicago.edu/display/uc3/

https://wiki.uchicago.edu/display/uc3/Presentations+about+UC3

https://github.com/DHTC-Tools/ubolt
• What about federated identity?

• Probably possible to perform some or nss_compat functions in ldap.conf with nss_* parameters; this might save some headache

• where do nss and pam modules install?
  1. /lib/security/pam_xyz.so[.2]
  2. /lib64/security/pam_xyz.so[.2]
  3. /usr/lib/libnss_xyz.so[.2]
  4. /usr/lib64/libnss_xyz.so[.2]