Kerberos on Servers

- "host" means ssh/telnet login to the server itself
- "service" means applications like HTTP, POP3
- In both cases you need to:
  - 1. Enable Kerberos authentication in the software
  - 2. Create a principal in the KDC
  - 3. Put the corresponding key in a keytab file
- What Microsoft calls "joining a domain"
- Not much harder than adding clients

Kerberised sshd

```
# editor /etc/ssh/sshd_config
...
GSSAPIAuthentication yes
GSSAPIKeyExchange yes       # when available
...
```

- Note: don't set "KerberosAuthentication yes"
  - That really means password login, with the password checked against KDC
  - True Kerberos doesn't send the password at all
  - When properly deployed you can turn off ssh password authentication completely!

Creating the keytab

- Option 1: run kadmin on the target itself, using kerberos administrator account.
    - strong random key; copy across net is encrypted

```
# kadmin -p username/admin
addprinc -randkey host/pcN.ws.nsrc.org
ktadd host/pcN.ws.nsrc.org
```

- Option 2: extract keytab on another machine, copy to target e.g. with scp
- Option 3: set passphrase on KDC, use ktutil on target with same passphrase (awkward in practice)

Kerberised Apache

- mod_auth_kerb in Ubuntu 8.04, RHEL 4 &up

```
<Location /secure>
  AuthName "Hello Kerberos World"
  AuthType Kerberos
  # Allow fallback to Basic Auth?
  KrbMethodK5Passwd Off
  KrbAuthRealms WS.NSRC.ORG
  Krb5Keytab /etc/apache2/krb5/krb5.keytab
  # TODO: LDAP authorisation
  # require user testuser@WS.NSRC.ORG
  require valid-user
</Location>
```

Kerberised Apache

- Create a service principal and a keytab which is readable to the Apache user ("www-data")
    - Depending on clients, may need to include principals for both virtual server name and real server name

```
# mkdir /etc/apache2/krb5
# kadmin -p username/admin
addprinc -randkey HTTP/noc.ws.nsrc.org
ktadd -k /etc/apache2/krb5/krb5.keytab \
         HTTP/noc.ws.nsrc.org
^D
# chown -R www-data:www-data /etc/apache2/krb5
# chmod 550 /etc/apache2/krb5
# chmod 440 /etc/apache2/krb5/krb5.keytab
```

Authorization

Tickets aren't authorization

- A ticket is proof of your identity to a particular endpoint - nothing more (*)
- You can ask for tickets to prove your identity to any principal you like. KDC doesn't care.
- Sounds a bit like certificates? It is!
    - Uses symmetric cryptography instead of public/private
    - Hence you need a separate ticket for each endpoint
    - But symmetric crypto is cheap and fast

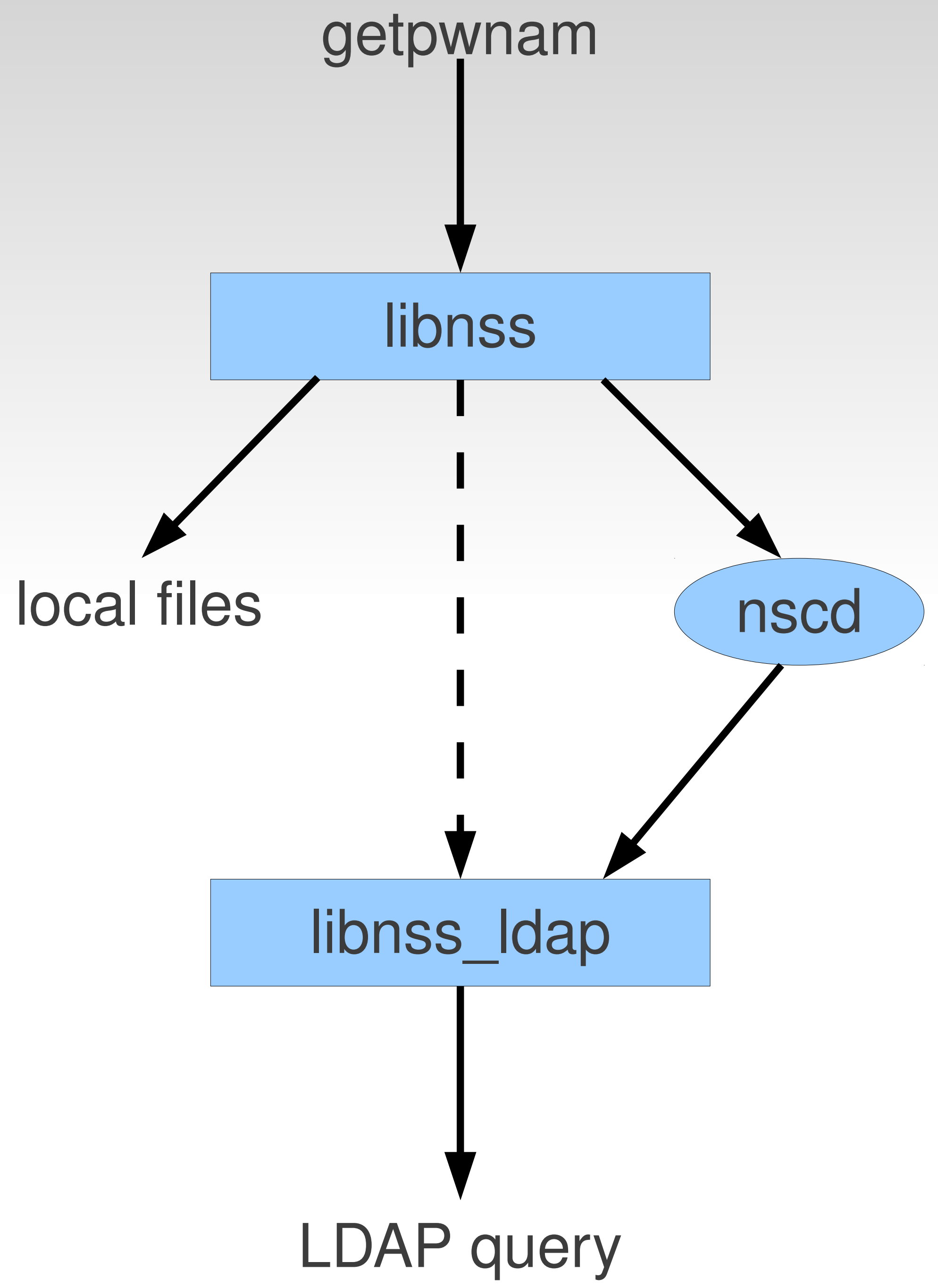
(*) Microsoft has bastardized the concept by including "Privilege Access Certificates" (PACs) in tickets. In large AD deployments tickets can become huge, and hence logins slow.

Login authorization

- sshd needs to decide whether to allow a particular principal to login as a particular user
- Default rule: map foo@THIS.HOSTS.REALM to system user "foo"
- Default denies all users in other realms
- You can add explicit authorization by putting principal name(s) into ~/.k5login
    - Like adding a key to ~/.ssh/authorized_keys, but simpler

Login authorization (cont)

- But we also need to know what uid for user "foo", what groups they are in, their home directory etc
- We don't want to distribute /etc/passwd files!
- So configure system to use LDAP database for passwd and group info
- Can restrict logins to particular groups (*pam_access*)
- LDAP communication needs to be strongly protected, by Kerberos or TLS
    - LDAP is controlling privileges, so it's very important that it's secure


getpwnam
libnss
local files
nscd
libnss_ldap
LDAP query

Configuring LDAP

- /etc/ldap.conf *[man nss_ldap]*
    - LDAP server and base DN; attribute mapping
- /etc/nsswitch.conf
    - use LDAP for passwd, shadow, group
- /etc/nscd.conf
- /etc/cron.hourly/kerberos
    - obtain Kerberos ticket for name service caching daemon to be able to query LDAP
- Make your own tarball to deploy

Exercise

- Part one: set up your machine to accept Kerberos authenticated logins
- Part two: set up your machine to use LDAP for uid/gid mapping
- We're doing it manually, but remember in real life you'd deploy a tarball/package/script etc

More authorization scenarios

Kerberos admin (kadmin)

- Certain people are authorized to add/modify/remove other principals via kadmin
- This is security critical
- How do we control it?

Option 1

- List all the authorized entities in the KDC ACL
  - brian@REALM   *
  - carlos@REALM   *
  - hervey@REALM   *
- Advantages:
  - Clean separation of "authentication" and "authorization"
- Disadvantages:
  - Need to edit a file on the KDC to amend the ACL

Option 2

- Admin users have a second identity:
  - username/admin@REALM
- Pattern-matching ACL in the KDB
  - */admin@REALM   *
- Advantages:
  - The ACL never needs adjusting
  - You have to enter a different password when doing "admin" things (more secure??)
  - Some tools like kadmin have this as default behaviour

System root access

- Some documents suggest having separate principals for superuser access, e.g. username/root@REALM
- Authorize */root to login as root (or ksu)
- Again, user has multiple identities
- I think this muddles authentication vs authorization
- Can use sudo, but don't want to expose password
- Can allow sudo with NOPASSWD for wheel group
  - membership of this group is my authorization

Other services

- HTTP (Apache)
  - Authorize users via LDAP groups
  - Doesn't work with apache 2.0 / mod_auth_ldap
  - Use apache 2.2 / mod_authnz_ldap (Ubuntu, RHEL5+)
- Access to LDAP database itself
  - OpenLDAP can be configured with rules to map principal to DN (olcAuthzRegexp)
  - Then has its own ACL for DN authorization

Switches and Routers

- Some Cisco IOS images support Kerberos
  - You need an alternative, e.g. RADIUS or TACACS+, for those which don't
- I tried it, and couldn't make it work :-(
- Supports authorization via instance mapping, e.g. myname/enable@FOO gives enable mode
- Otherwise need to a static table in TACACS+ for authorisation, or build a TACACS+ to LDAP bridge
- There are also commercial solutions (SecureACS)