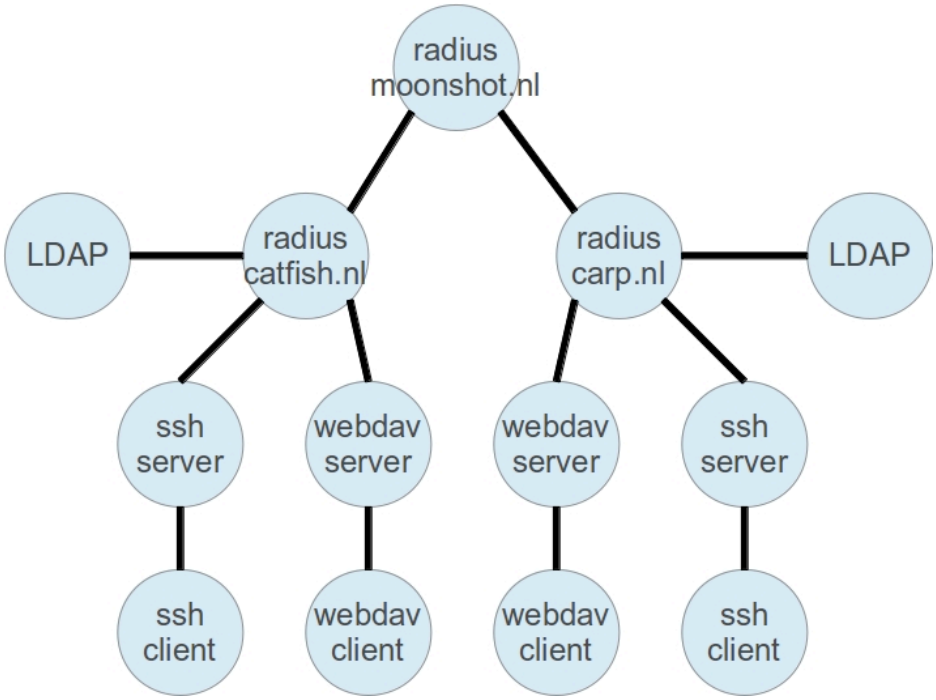
The Project Moonshot

Sebastiaan Heuft
Killian Hesterman
Rikkert ten Klooster
Bart Stokman

Versie 0.1.1

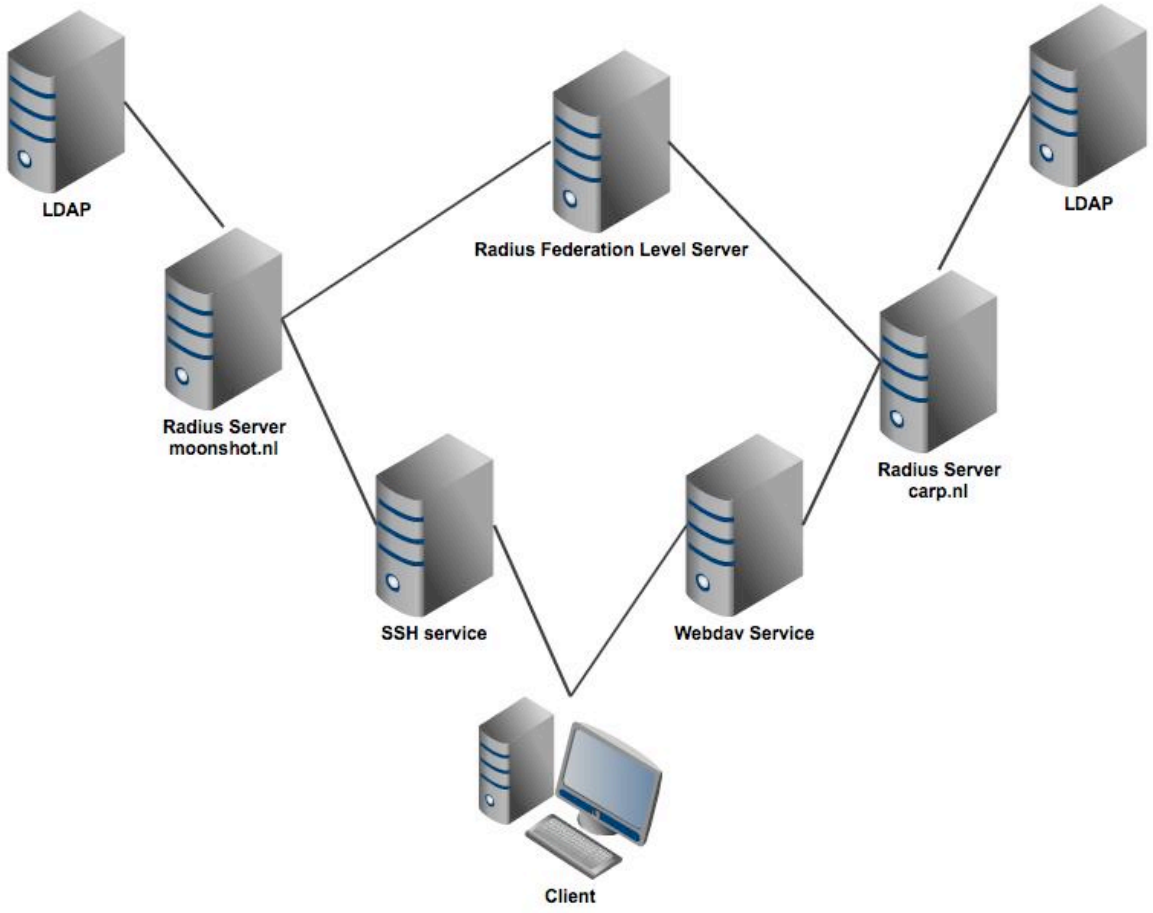
Preface

Welcome to the documentation of Project Moonshot. The project started at the Hogeschool of Amsterdam where the minor Forensic Intelligence and Security (FIS) is given. Our team includes Sebastiaan Heuft, Killian Hesterman, Rikkert ten Klooster and Bart Stokman. All these team members follow the same minor FIS. In every minor there is a project with the duration of about 6 months. After our team was made, we were assigned to The Project Moonshot of the Nikhef. Nikhef is the Dutch National Institute of Subatomic Fysics which has many big and interesting projects. Our client O. Koeroo will accompany us during this project and at the end we will present him our final product.
The idea of Project Moonshot is based on Eduroam. Eduroam is an world-wide roaming access service for international research and education communities. Eduroam stands for education roaming. The main goal of eduroam is to give students, researchers or teachers from participating institutions internet in every institutions who is a member of the eduroam federation. Eduroam is currently available in 54 countries all over the world.
Eduroam is has a lot more potention then to only use it for Internet. Therefore our assignment is to build federated Webdav-server and a federated SSH-server. So every user from a federated institution has immediate access to a shell as well as federated storage with authorization options.



1 Introduction to Project Moonshot

The idea of this Project Moonshot is to research the possibilities of federated authentication and authorization. The core idea is that a user should be able to login on services from different organizations with one user account. The service can then give the user access to different resources according to the attributes provided by the user account. To build this infrastructure we used different components. The purpose is to improve the user experience and enhance the management of all user accounts. The main focus was to extend the Project Moonshot with authorization options. When we started, there were quite some demonstrations about authentication via Moonshot but less about authorization. After recreating some demonstrations of SSH authentication over RADIUS with federated servers the focus moved to the point how we could extend this with authorization.



1.1 Authorization

Authorization is the process of specifying access to resources related to the specific user accessing the service. To extend Moonshot with authorization we had a couple of core challenges to attend. Where in the process should the authorization take place? Where should the attributes be stored? How can we keep it secure? We'll try to answer these questions with the approach we used.

1.1.1 Where in the process should the authorization take place?

You can choose to perform the authorization in several parts of the process. The choice would be at the authentication server (RADIUS) or near / at the Service Provider. When authenticating at the authentication server the process would be controllable near the holder of the user accounts. This would be preferable if the organization holding the user account has to be in charge of the decisions. For a school that makes use of storage provided by a storage company and where the school has to pay by the amount of storage used. In that case the authorization process should take place at the school so the school can decide which resources the user gets. In our case the authorization at the service end was more preferable. We want the system administrators of the service to be able to have control over which resources a user gets. We had to find out how to get the attributes from where the user accounts were stored to the service. To do so the attributes from passed through from LDAP to FreeRADIUS. This was an already supported feature, the attributes only had to be mapped. We chose to do WebDAV and SSH authentication over PAM and then let PAM authenticate via RADIUS. The reason for that was so that we could extend it with our own module. The module maps the attributes from FreeRADIUS to unix groups. The unix groups gets added to the user that gets logged in to either WebDAV or SSH. WebDAV and SSH then can give the user the privileges according to the groups. The choice to do it via PAM was that SSH and WebDAV both already have the ability to authenticate by PAM. WebDAV even offers authorization options based on unix groups. By choosing for PAM we could build a relative easy demonstration on how we thought authorization is preferable.

1.2 Where should the attributes be stored?

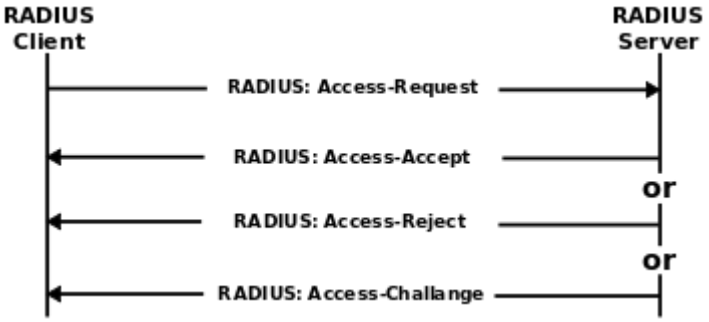
This problem is a bit similar to the one above, it would be nice to keep all the user data at the same place but this is not always possible. In our case we had full control over all servers, but there might be situation where the Service Provider wants to store information about a user but does not have control over the account data (LDAP). We could store additional data in LDAP, a good solution to expand this would be by storing the additional data near the authorization application (PAM). Through XACML (eXtensible Access Control Markup Language) you can define authentication and authorization policies. A PAM can send a request with the user object and its attributes, including the resource to be used and action. XACML will based on the defined policies answer with accept / reject / don’t know or an obligation. The obligation can contain instructions for the PAM module to perform. For example: Add the user A to group x,y,z. We have not tested this but there is space for further research on this part.

1.3 How can we keep it secure?

As described in the RADIUS section, RADIUS supports 802.1x including EAP-TTLS. Right now it is possible to keep the end from PAM through RADIUS to LDAP secure with EAP-TTLS. From the moment that this tunnel is opened and data is sent through a system administrator will be denied from sniffing data. Right now the authentication from SSH to PAM is not tunneled. If this would be deployed in one company without sensitive information this might be acceptable. But the use case would be a federation where it would not be acceptable that one organization can sniff user passwords from another organization. WebDAV currently is not secured with HTTPS but this is possible, however WebDAV doesn't support EAP-TTLS right now it wouldn't be secure enough for a federation either. For Moonshot it is possible to use a modified SSH-client, SSH-server and GSS for authentication over radius, this support EAP-TTLS which would be secure but does not offer the wished authorization possibilities.

2 Introduction to RADIUS

Remote Authentication Dial In User Service (RADIUS) is a networking protocol that provides centralized Authentication, Authorization, and Accounting (AAA) management for computers to connect and use a network service. RADIUS is used to interact between different domains for exchanging account data. It is the link between a service making the request and the account data.



The diagram shows an example of an Access-request. The service asks access for user credentials and the RADIUS service either accepts the request, rejects the request or challenges the user to respond with additional credentials.

2.1 What are the possibilities of RADIUS?

RADIUS offers authentication, authorization and accounting. RADIUS handles access requests from a user through a service. The RADIUS server checks that the information is correct using authentication schemes like PAP, CHAP or EAP. The user's proof of identification is verified, along with, optionally, other information related to the request, such as the user's network account status and specific network service access privileges. This information can be later on used by the service to give the user privileges. Modern RADIUS servers can refer to external sources for the user data such as MySQL or LDAP as in our case was used.

2.2 How did we use RADIUS?

It is a core part in the moonshot project we worked on. It’s the link between a Service Provider and the account data. We’ve used three RADIUS servers, two connected to a different LDAP containing the user data. And one central RADIUS server. The server was the link between the two RADIUS servers containing user data. The two different RADIUS servers were running under the domain carp.nl and moonshot.nl. When a request comes in for a carp.nl account on the moonshot.nl it gets redirected through the central server to the carp.nl server which then handles the authentication request. We used FreeRADIUS for the servers.

2.3 Realms

Example config of the server on realm moonshot.nl

```
#Used when there is no domain present in the authentication request.
realm NULL {
	type		= radius
	authhost	= LOCAL
	accthost	= LOCAL
	secret		= testing123
}

#Used when the domain for the request is moonshot.nl
realm moonshot.nl {
	type		= radius
	authhost	= LOCAL
	accthost	= LOCAL
	secret		= testing123
}

#Used when the domain for the request is anything but moonshot.nl
realm DEFAULT {
	authhost	= 10.198.5.39:1812
	accthost	= 10.198.5.39:1813
	secret		= testing123
	nostrip
}
```

In order for RADIUS to respond with attributes the LDAP attributes needed to be mapped to FreeRADIUS attributes. After mapping these attributes when a LDAP attribute is present in the user account the RADIUS server will pass any of these attributes on to the service.

2.4 Protocols

2.4.1 802.1x

IEEE 802.1X defines the encapsulation of the Extensible Authentication Protocol (EAP) over IEEE 802 (LAN). RADIUS supports the 802.1x standard for authentication. The authentication involves three parties. The supplicant, authenticator and authentication server. The supplicant is the client requesting access to the authenticator. The authenticator is a service offering resources to the supplicant. The authentication server runs software supporting the RADIUS and EAP protocols. When a supplicant requests access the service/authenticator the tunnel forwards the request to the authentication server. A secure SSL tunnel is set up between the supplicant and the authentication server. When the secure tunnel is set up then the credentials will be requested by the authentication server. If the user provides the right credentials the server will accept the user and the tunnel will be closed again. Upon success the authenticator can give the user access to the requested resource.

3 Introduction to LDAP

3.1 What is LDAP?

The Lightweight Directory Access Protocol (LDAP) is an application protocol for accessing and maintaining distributed directory information services over an IP network. The directory service may provide sets of records often in a hierarchical structure. The service is often used for storing customer data, email, course data etc.

3.2 What are the possibilities of LDAP?

LDAP is a good way to store user information in a hierarchical structure. The LDAP protocol is readily available across the web. It has a well defined API which makes it easy for applications to implement LDAP integration.

3.3 What are the limitations of LDAP?

If you want to use LDAP you'll need LDAP-enabled applications. While LDAP usage should increase it is not sure that the application of choice supports LDAP.

3.4 How did we use LDAP?

We used LDAP to store the account data of users. To create a realistic environment we used the eduperson Object Classes. It's a schema designed to include widely-used person and organizational attributes in higher education. This standard got us started with several fields for the users objects. These attributes can be used in a service to give a user privileges to the service. For the purpose of this project we created an attributed called unixGroup, this unixGroup would later added to the user object by PAM. See the config below:

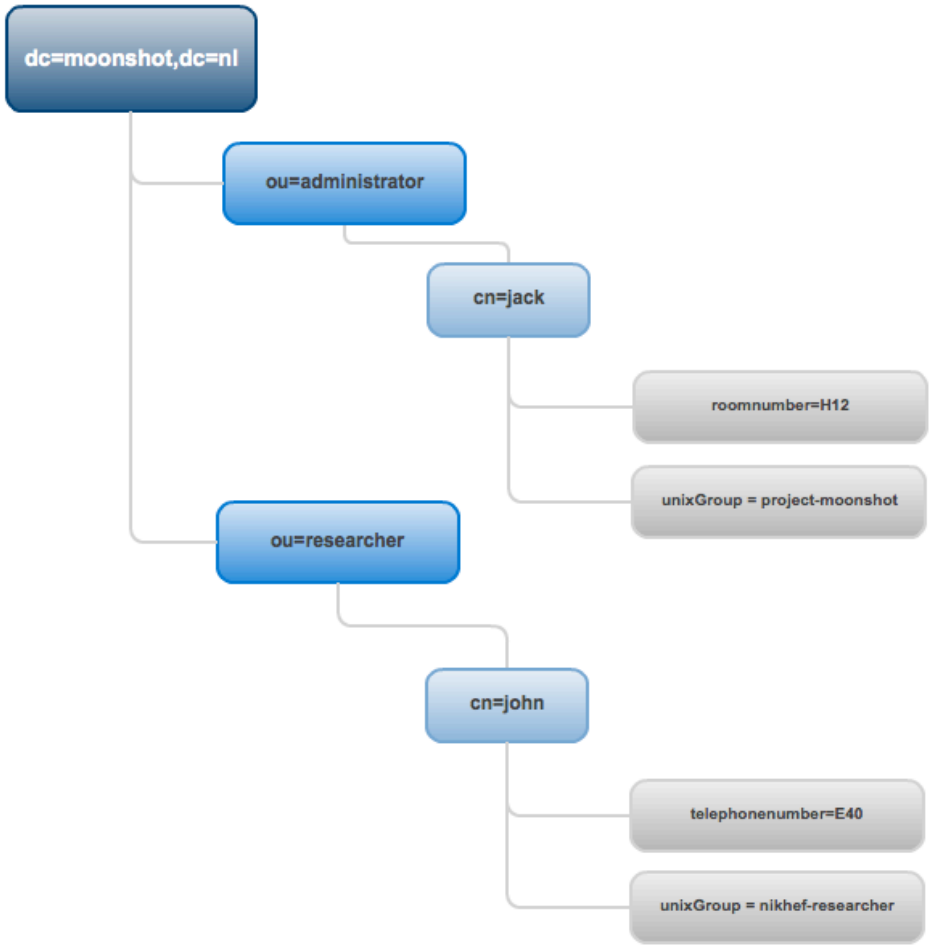
```
attributetype (1.3.6.1.4.1.3317.4.3.200.1
    NAME 'unixGroup'
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{1024})

objectclass ( 2.16.840.1.113730.3.200.1
    NAME 'unixUser'
    SUP 'inetOrgPerson'
    DESC 'Unix User'
    STRUCTURAL
    MAY (unixGroup))
```

OpenLDAP is the application we chose to use. It's a free and open source implementation of LDAP. It runs on well on linux and is stable in its development. It meets all the requirements we have of storing data and it interacts well with freeradius.

3.5 What problems did we have?

OpenLDAP worked well out of the box. It is easy to expand the schema, create objects and perform requests.

3.5 Example of a LDAP



structure

4 Introduction to Pluggable Authentication Modules (PAM)

4.1 What is PAM?

PAM is software that applies an additional layer to Unix authentication. Most standard Unix-applications (su, sudo) are authenticating via PAM. The application uses pam_start() to start the authentication process. If not already set, PAM asks for a username and password and tries to authenticate the user. Fortunately PAM can do more than authenticating only.

PAM has 4 tasks:

- Authentication, establish the user who they claim to be.

- Account, provide account verification types of service.
- Password, update authentication mechanisms.
- Session, provides both opening and closing hook for modules to affect the service.

PAM can be extended by adding modules. As a system administrator you can add modules to extend the tasks of PAM. For example, you can add a module that requires the user to enter an extra authentication token.

This is the default PAM stack for the OpenSSH daemon on a Linux-system (CentOS 6.2):

```
#%PAM-1.0
auth        required      pam_sepermit.so
auth        include       password-auth
account     required      pam_nologin.so
account     include       password-auth
password    include       password-auth
# pam_selinux.so close should be the first session rule
session     required      pam_selinux.so close
session     required      pam_loginuid.so
# pam_selinux.so open should only be followed by sessions to be executed in the user context
session     required      pam_selinux.so open env_params
session     optional      pam_keyinit.so force revoke
session     include       password-auth
```

As you can see, a PAM-entry consists of three fields. The PAM-task, control and the path to the module. Optionally you can pass module-specific arguments to the module. The ‘control’-field is used the control the PAM-stack.

There are 4 types of control:

- Requisite, upon failure the authentication process will be terminated.
- Required, the module is required for the process to succeed.
- Sufficient, the module is sufficient for the process to succeed.
- Optional, if the module fails PAM will go further.

Other PAM-stacks can also be included. PAM will execute the PAM-stack in the included file.

4.2 Advantages and Disadvantages

Advantages:

- PAM provides an authentication scheme that can be used by many applications
- PAM allows system administrators and developers to control over authentication
- PAM allows developers to develop applications without creating their own authentication mechanism.
- PAM is well documented.

Disadvantages:

- Users with less Unix experience may find the documentation difficult to understand.
- Users with less Unix experience may find the PAM-stack difficult to fully understand.

4.3 PAM & Moonshot

4.3.1 Why PAM & Moonshot?

As you probably have read in Chapter 1, the goal of Project Moonshot is to enable management of access to multiple services using a single technology and infrastructure. There are already a lot of applications that can use PAM for authentication. Our goal was to make OpenSSH & WebDAV to work with Moonshot, without patching the software. Moonshot has to be easy to implement in existing systems and federations like the worldwide eduroam-federation. So patching OpenSSH or WebDAV is not the best option.

4.3.2 Possibilities of PAM & Moonshot

- Single sign-on for users with ‘username’@’domain’ for multiple services (SSH, WebDAV, NFS).
- Authorization based on the user’s LDAP-attributes.
    - Map users to (temporary) pool-accounts.
    - Map users to (temporary) Unix-groups.
    - Manage data storage per user.
    - Manage connection speed per user.
    - etc.

4.3.3 Our goal

Our goal is to implement a PAM that can at least authenticate a user via Freeradius. We will try to use the pam_radius_auth.so module. This module is developed by the Freeradius Project to do Freeradius authentication and accounting.
The PAM-stack can be used by OpenSSH and WebDAV for authentication. Users can log in to SSH and WebDAV using the same credentials.

Because we have limited time for this project, we probably can’t implement full authorization via PAM. However, if there is enough time, we will try to implement little authorization based on one or more attributes. Just to demonstrate possibilities.

4.3.4 Authorization by PAM

PAM can also do the authorization for users. An existing PAM for use with LDAP (without RADIUS) does authorization for users based on LDAP-attributes. There is no PAM yet that does authorization via RADIUS-attributes.
We did a little research on XACML. XACML stands for eXtensible Access Control Markup Language. XACML is primarily an Attribute Based Access Control (ABAC) system where attributes associated with a user are inputs into the decision of whether a given user may access a given resource in a particular way. There is already a PAM for XAMCL.

Examples for types of authorization:

- SSH: access to specific files, root privileges, tunneling, etc.
- NFS: amount of storage, access to specific directory’s.
- FTP: upload files, edit/delete files.

4.3.5 Authorization using a Radius-attribute - Demonstration

We have researched the possibility to develop a PAM to assign a supplementary Unix-group to a user that authenticates via PAM. After doing some research we concluded that it is possible to create a working demonstration. The code of the PAM can be found in Appendix B. Keep in mind that this module is purely for demonstration. It only demonstrates that Authorization based on Radius-attributes can work with PAM. Our knowledge of the programming language C is also not good.

Because we installed an NSS-module that maps all unknown users to a ‘pool-account’, we can use 2 usernames in the PAM. With getpwnam() we get the username of the pool-account, and with pam_get_user() we get the username given by the authenticating user.
The PAM-module has a function that does a radtest to the authentication server with the given username (‘username’@’domain’). Because the PAM is executed in the account-section of PAM, we can not get the password the user has entered. Therefore we are doing a radtest with an empty password which also replies the Unix-group as an attribute. We strip the Unix-group from the RADIUS reply and return the value.
The second function (pam_sm_acct_mgmt()) is called by PAM. In that function we check if the Unix-group exists on the system, if so we add the group as supplementary group to the pool-account-user.

After the user logged in, he is added to the Unix-group.

In this PAM the user is added to the group. This is also saved in the /etc/group file. It is also possible to make a temporary group mapping. By using the functions getgroups() and setgroups() you can give the process a membership to one or more groups. After the process (SSH session) is shut down the mapping is gone.

To achieve that the user is also removed from the group after he closes the session we used another PAM. You can find this PAM in Appendix B2. This PAM hooks in on the pam_close_session() function called by the service. PAM executes the pam_sm_close_session() function in the PAM-stack. Our module (pam_radius_removegroup.so) removes all supplementary groups from the user. For our demonstration it is just using the ‘usermod’ command.

This is our PAM-stack for our demonstration:

```
#%PAM-1.0
auth       required     pam_radius_auth.so debug try_first_pass

account    required     pam_radius_group.so
account    required     pam_radius_auth.so

# delete group-mapping done by pam_radius_group.so
session    required     pam_radius_removegroup.so
session    required     pam_radius_auth.so debug

session    required     pam_loginuid.so
# pam_selinux.so open should only be followed by sessions to be executed in the user context
session    required     pam_selinux.so open env_params
session    optional     pam_keyinit.so force revoke
```

4.3.6 Security Considerations

PAM also has some security issues. The credentials that are sent to the RADIUS-server are sent in plain text. A man-in-the-middle can sniff all passwords that are being sent to the RADIUS-server. A solution to this problem is SSL. With SSL all data sent over the network can be sent encrypted. Another solution is to create an IPSEC connection between PAM and the RADIUS-server. This will also send all data encrypted over the network.

The system administrator of the server doing the request to PAM can also steal credentials. In the case of OpenSSH, the credentials are sent locally from SSHd to PAM. This security issues is not only applicable to Project Moonshot. As a user you have to trust the authority which you are giving your credentials.

5 Introduction to Name Service Switch (NSS)

5.1 What is NSS and what is NSS used for?

NSS provides sources for common configuration databases and name resolution. For example, if an application wants to check whether a user exists on the system it uses the function getpwnam(). This function asks NSS for the user.

/etc/nsswitch.conf

```
passwd:         files moonshot
shadow:         files
group:          files
```

In the configuration file above you can see that the ‘passwd’ entry has 2 modules that search for the given username. If, for example, your application checks a username that does not exist on the system, the first NSS-module will return false. Therefore, the second NSS-module is executed.

Finally NSS returns a pointer to the passwd object. The pointer points to the struct passwd which looks like this:

```
#include <pwd.h>
struct passwd {
        char *pw_name;
        char *pw_passwd;
        uid_t pw_uid;
        gid_t pw_gid;
        time_t pw_change;
        char *pw_class;
        char *pw_gecos;
        char *pw_dir;
        char *pw_shell;
        time_t pw_expire;
};
```

NSS simplifies the search for entries is common database files. Applications can use the function getpwnam() to get the passwd-struct of a user for example. Databases which can be searched by NSS include:

- /etc/passwd
- /etc/group
- /etc/shadow
- /etc/services
- /etc/hosts

5.2 Why NSS & Moonshot?

NSS can be used to map user accounts to 'pool-accounts'. For example, bas@moonshot.nl is mapped to user pool0001. The mapping is needed on Unix systems because you can not create a user with an '@'-sign in the username on a Unix system. When connecting to a service in a federation you usually log in with a username like 'username'@'domain'. For the SSH-service, a custom NSS-module is required. At first, OpenSSH will set the password to '\010\n\r\177INCORRECT' if the user does not exist on the system. OpenSSH uses the getpwnam() function to check if a user exists. Secondly, you can not get a shell with a non existing account.

5.3 Our goal with NSS

Our goal is to create a working demonstration for federated login to OpenSSH and WebDAV. Therefore we need an NSS-module which maps the authenticating user to an existing Unix-account on the system. These mappings can be stored in a database to keep track of logged in users. For our demonstration we are not storing these mappings in a database.

In Appendix A, you can find the NSS-module we used to create the simple mapping. This module simply maps any user to an existing user based on a user-id. To let the NSS-module work, make sure the user-id exists on your local system.

5.4 NSS & LDAP

With NSS you can also for example connect an NSS-module to an LDAP-server. When an application calls for example getpwnam(), NSS will look in the LDAP-database to check if the user exists and returns the pointer to the passwd-struct of the LDAP-user on the local system. The goal of the Moonshot Project is to use federated login, therefore this module can not be used. For the Moonshot Project we need RADIUS-servers for authentication. The RADIUS-servers can use LDAP as their database.

6 Introduction to SSH

6.1 What is SSH?

The SSH-protocol encrypts all data that is communicated between servers and clients. It uses port 22 by default but as an extra security measurement this is changed frequently changed. The protocol uses public-key cryptography. This basically means that you need a public key on the remote machine you want to access and you need a matching private key on the machine you want to access the remote machine from. The private key is never send true the network while authenticating but is only used for verifying a match with the public key. If so the connection will be established.

6.2 What is SSH used for?

SSH is typically used to log into a remote machine and execute commands. Therefore it is used a lot in the cloud for a secure connection with the cloud. SSH allows tunneling and port forwarding. You can transfer files using the SSH file transfer protocol (SFTP) or the secure copy protocol (SCP).

6.3 Why OpenSSH?

OpenSSH is an open-source implementation of the SSH-protocol and is developed by the OpenBSD project and supports all SSH versions. At OpenBSD there are two teams working on OpenSSH. One team is working on OpenSSH for OpenBSD. They make the code so clean, simple and secure as possible. The other team is then porting the OpenSSH to other operating systems. OpenSSH comes with a lot of operating systems by default. First we used OpenSSH_5.3p1 which is the standard version of SSH installed on CentOS 6.2. Because of some issues which will be explained later we updated it to OpenSSH_5.8p2 which is the default from OpenSUSE 12.1.

6.4 OpenSSH & Moonshot

OpenSSH is one of the applications that is successfully tested with moonshot. They used a moonshot enabled version of OpenSSH and they use the GSSAPI for authorisation and authentication. We decided that we want to use a stock version of OpenSSH but still we want all the benefits of moonshot. So we combined a stock version of OpenSSH with pluggable authentication modules (PAM) and a name service switch (NSS) module. Therefore you do not need a moonshot enabled SSH server only make some configurations changes. The NSS module does a user lookup in the ldap from the domain from the user. The pluggable authentication modules do the authentication but also the authorisation.

6.5 Specific configurations for OpenSSH

For authenticating to SSH with PAM you need the following configuration in the the sshd_config file in the /etc/ssh folder.

PasswordAuthentication yes
ChallengeResponseAuthentication yes
UsePam yes

The PAM-stack for SSHd can be found in /etc/pam.d/sshd. Here you can configure which Pluggable Authentication Modules SSH has to use. For authentication via Radius you have to add the pam_radius_auth.so module to the PAM-stack.

```
auth        required    pam_radius_auth.so try_first_pass
session     required    pam_radius_auth.so
```

An NSS-module is also required. OpenSSH checks if a user exists on the system using getpwnam(). If the user does not exist, authentication is not possible. An NSS-module that maps any username to a pool-account tells OpenSSH that the user exists on the system. Even if the user is not found in the /etc/passwd file.

/etc/nsswitch.conf

```
passwd:     files moonshot
```

The NSS-module we used for our demonstration can be found in the Appendixes. In the NSS-chapter we have explained how the NSS-module works.

6.6 Step through

We first started with the OpenSSH_5.3p1 version a logged in from the localhost with a local user. This was no problem so we could continue with adding our RADIUS server. When we added our radius server it was no problem to log in if the user existed locally. We did not tested it yet with domain names after the username. Once we tried to log in with an username and a domain name we could not log in. In the RADIUS debug we saw there was something wrong with the User-Password which was “\010\n\r\177INCORRECT”. After debugging we found out that the OpenSSH server first checks if an user exist locally if this was not the cast the password was set to “\010\n\r\177INCORRECT”. We thought to quick fix this to just add the user with a script. We quickly discovered that you can not add an user name with an @ in the name. So our solution with creating the users locally was not going to work. While doing some research we came up with the idea of using name service switch module. We wrote a NSS module which would check if an user exist in the LDAP from the domain of the user. But still the SSH daemon checks if the user exist locally if not the password will be set to “\010\n\r\177INCORRECT”. When we tried the same on an OpenSUSE machine it worked. So we checked the differences between the OpenSSH server from CentOS 6.2 and the OpenSSH server from OpenSUSE 12.1. We saw that OpenSUSE had the OpenSSH_5.8p2 installed and CentOS had the OpenSSH_5.3p1 installed. After installing the OpenSSH_5.8p2 on our CentOS machine and made changes to the sshd_config. We could login with an username and domain name.

6.7 Problems

6.7.1 Problem password incorrect

OpenSSH checks if the user local exist if not set password to incorrect.

auth.h (line 49-75):

```
struct Authctxt {
        int                 valid;          /* user exists and is allowed to login */
        struct passwd  *pw;             /* set if 'valid' */
}
```

This is the stripped definition of the Authctxt struct in the auth.h file from the source code of OpenSSH_5.3p1. Only the variables we need are shown. The Authctxt->valid variable is set to 1 if the user exists on the local system. If the user does not exist, Authctxt->valid is not set.

auth1.c (line 401-407):

```
/* Verify that the user is a valid user. */
if ((authctxt->pw = PRIVSEP(getpwnamallow(user))) != NULL)
        authctxt->valid = 1;
else {
        debug("do_authentication: invalid user %s", user);
        authctxt->pw = fakepw();
}
```

auth_pam.c (r. 234, 834-839):

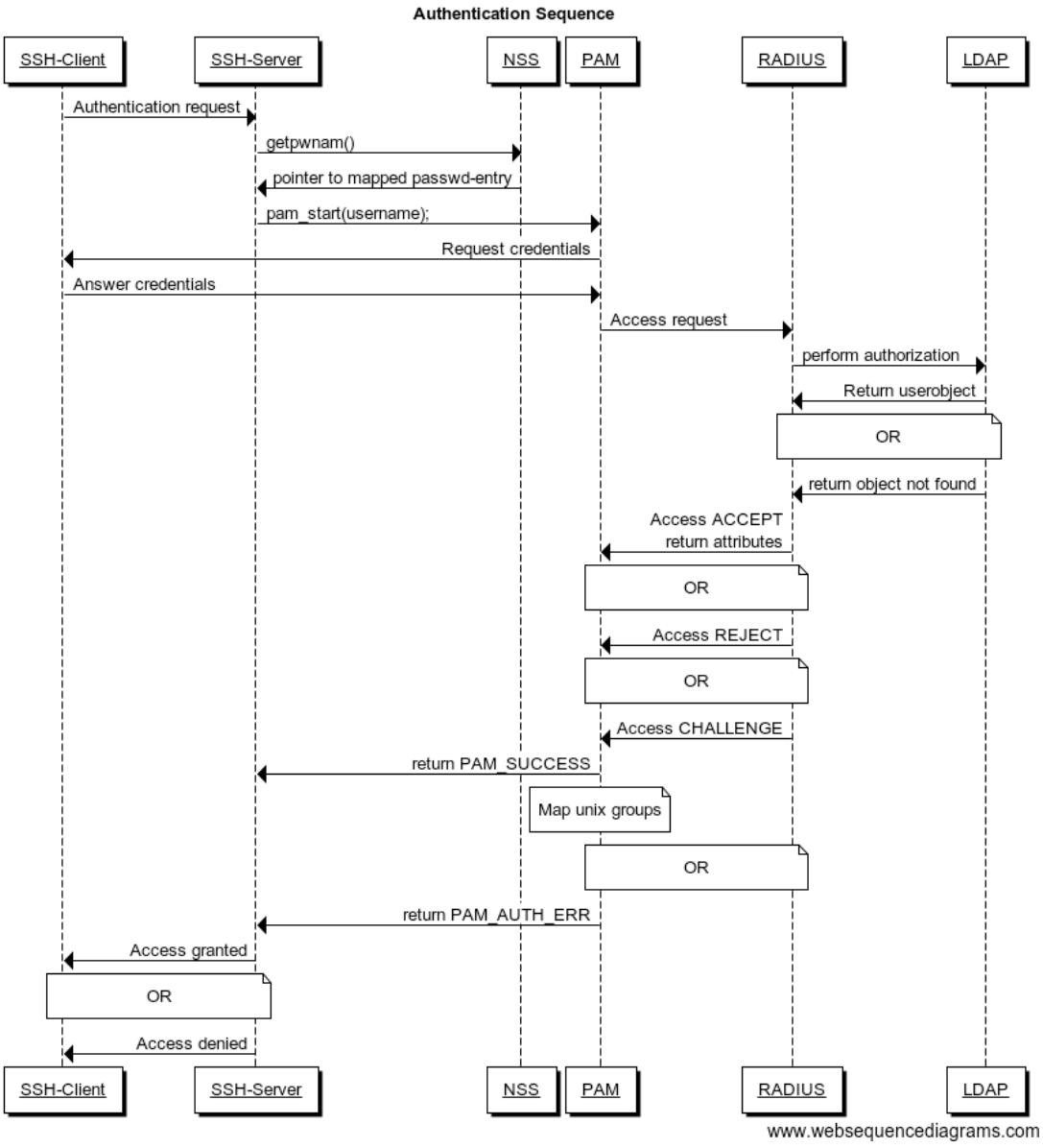
```
static char badpw[] = "\b\n\r\177INCORRECT";

if (sshpam_authctxt->valid &&
        (sshpam_authctxt->pw->pw_uid != 0 ||
        options.permit_root_login == PERMIT_YES))
                buffer_put_cstring(&buffer, *resp);
        else
                buffer_put_cstring(&buffer, badpw);
```

Here is shown that if sshpam_authctxt is not valid the badpw array will be sent. It is not valid if the user does not exist or is not allowed to login.

6.7.2 Solution password incorrect

Updating our version of OpenSSH from version 5.3p1 to 5.8p2. For some reason it worked then.

6.8 Sequence Diagram



SSH

6.9 Future aspects

Checks if it still works with OpenSSH_6.0p1
Checks why OpenSSH_5.3p1 did not support PAM usage
Support for GSSAPI with a stock version of OpenSSH

7 Introduction to WebDAV

7.1 WebDAV

Everybody knows the problems of collaborating on the internet. Sending emails over and over again, notifying each other when a new update is ready of a document. Constantly have the urge to let everybody know what is done and what has to be done. The intention of people is to make things as easy as possible. WebDAV simplifies that problem. WebDAV makes it possible to edit web documents or other files on the spot. These documents can be protected so they could not be overwritten and it is possible to limit access rights. Versions can be stored for later retrieval and relevant metadata can be viewed on the fly.

7.2 Initiation Project Moonshot - WebDAV

Project Moonshot is a single unifying technology for extending the benefits of federated identity to a broad range of non-Web services, including Cloud infrastructures, High Performance Computing & Grid infrastructures and other commonly deployed services including mail, file store, remote access and instant messaging. The goal of the technology is to enable the management of access to a broad range of services and applications, using a single technology and infrastructure. The Nikhef extends this project in a collaboration with the Hogeschool van Amsterdam. Nikhef is the Dutch National Institute of Subatomic Fysics. The Nikhef was interested in the possibilities of several applications provided with federated login. One of the applications that came by was WebDAV. Our team of students of the HvA were commissioned by the Nikhef to research and develop the possibilities of SSH and WebDAV combined in a Project Moonshot environment.

7.3 What is WebDAV?

WebDAV stands for Web Distributed Authoring and Versioning and is an extension for the Hypertext Transfer Protocol (HTTP) that facilitates collaboration between users in editing and managing documents and files stored on web servers. More specifically, it is an extension of the HTTP/1.1 protocol. WebDAV adds new HTTP methods and headers. In addition, DAV specifies how to use the new extensions, how to format request and response bodies, how existing HTTP behavior may change, etc. WebDAV is mostly called DAV in short.

The WebDAV protocol makes the web a readable and writable medium. It provides a framework for users to create, edit, delete and move documents and files on a server. With server we mean a web server or web share.

The most important features of the WebDAV protocol include the maintenance of properties about an author or modification date, namespace management, collections and overwrite protection.

7.3.1 Maintenance of properties

Maintenance of properties includes such things as the creation, removal, and querying of file information. Properties are also wel known as metadata. Files or data published to the web contain a lot of information associated with it. This information about the files or data are very

usefull when searching through the Web resources. Particular WebDAV properties are name and value pairs. The name here is a URL and the value is a sequence of well formed XML. The DAV Searching and Locating (DASL) effort is working on a standard mechanism for searching WebDAV servers. The value of properties is very important here.

7.3.2 Namespace management

Namespace management deals with the ability to copy and move web pages within a server's namespace. If the Web is to successfully support distributed authoring, users must be able to manage the namespace of the Web server they are writing to. Discovery of what resources currently populate the relevant portion of the server name space, plus the ability to copy, move, and delete those resources, are the key elements required.

7.3.3 Collections

Collections deals with the creation, removal, and listing of various resources. Collections provide direct containment for local resources and referential containment for resources located anywhere on the Web. The WebDAV server maintains consistency in the direct containment relationships. When a new resource is PUT into the namespace of a collection, it is automatically added to the collection. Listing the contents of a collection is very similar to listing a directory of a file system. Most existing operations for listing a directory, such as "ls" or "dir," provide the name of a file, and an option for retrieving limited sets of properties about the file, such as its size, owner, and access permissions

7.3.4 Overwrite protection

Overwrite protection handles aspects related to locking of files. When two or more people write to the same document, changes can easily be lost as first one collaborator and then another one makes changes without first merging their documents. WebDAV has an exclusive write lock. This lock garantees that only the owner of the lock can overwrite the locked source. WebDAV also has a shared write lock. This lock allows two or more collaborators to work on a source together. Shared locks can only work when collaborators are aware of each others actions.

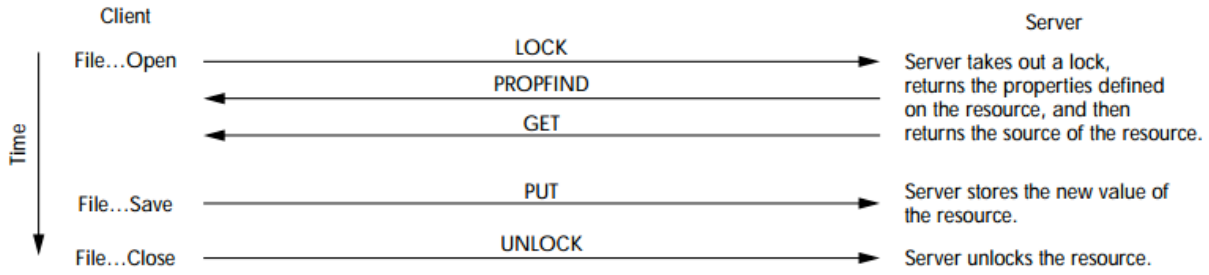
At first WebDAV was focused on only handling distributed authoring. Versioning was added later by an extension. Subversion is also compatible with WebDAV.

7.3.5 Methods

The protocol consists of a set of new methods and headers for use in HTTP. The methods include:

PROPFIND: used to retrieve properties, stored as XML, from a resource.
PROPPATCH: used to change and delete multiple properties on a resource.
MKCOL: used to create collections.
COPY: used to copy a resource.
MOVE: used to put a lock on a resource.
UNLOCK: used to remove a lock from a resource.

7.3.6 Example by figure



This figure shows a typical web client opening a file through WebDAV for an editing process. The Protocol requests that a generic WebDAV client would make to carry out specific client actions. Arrows indicate the predominant information flow associated with a request. All requests use the WebDAV plus HTTP 1.1 protocol. In this scenario, the user of a WebDAV-capable application selects the resource they want to edit using a standard File Open... The application then uses WebDAV LOCK to lock the resource and thus prevent modifications by other applications. Next, WebDAV PROPFIND retrieves the resource's properties; and HTTP GET retrieves its contents, which are then displayed for editing. Once all edits have been completed, HTTP PUT saves the resource back to the Web, and WebDAV UNLOCK removes the lock, allowing collaborators to access the resource

7.4 The WebDAV goal

The goal is to provide WebDAV with federated login (Authentication) like the eduroam network and provide Authorizing options that gives a user the properties provided for his account. WebDAV makes the web a writable, collaborative medium.

7.5 How did we use WebDAV?

We expended authorization options so that WebDAV can authorize over PAM. With the module authorization would be made possible over RADIUS. Therefore we used the authnz_external_module. This module makes webdav authorization via pwauth possible. pwauth will authenticate via PAM. Also did we use the authz_unixgroup_module. With this module you can make decisions based on the unix groups mapped to the user. The PAM module will map the unix groups from radius attributes to the unix user. See the PAM and RADIUS chapters for more information regarding this matter.
After that you can make authorization decisions based on the username and unix groups of the user account.

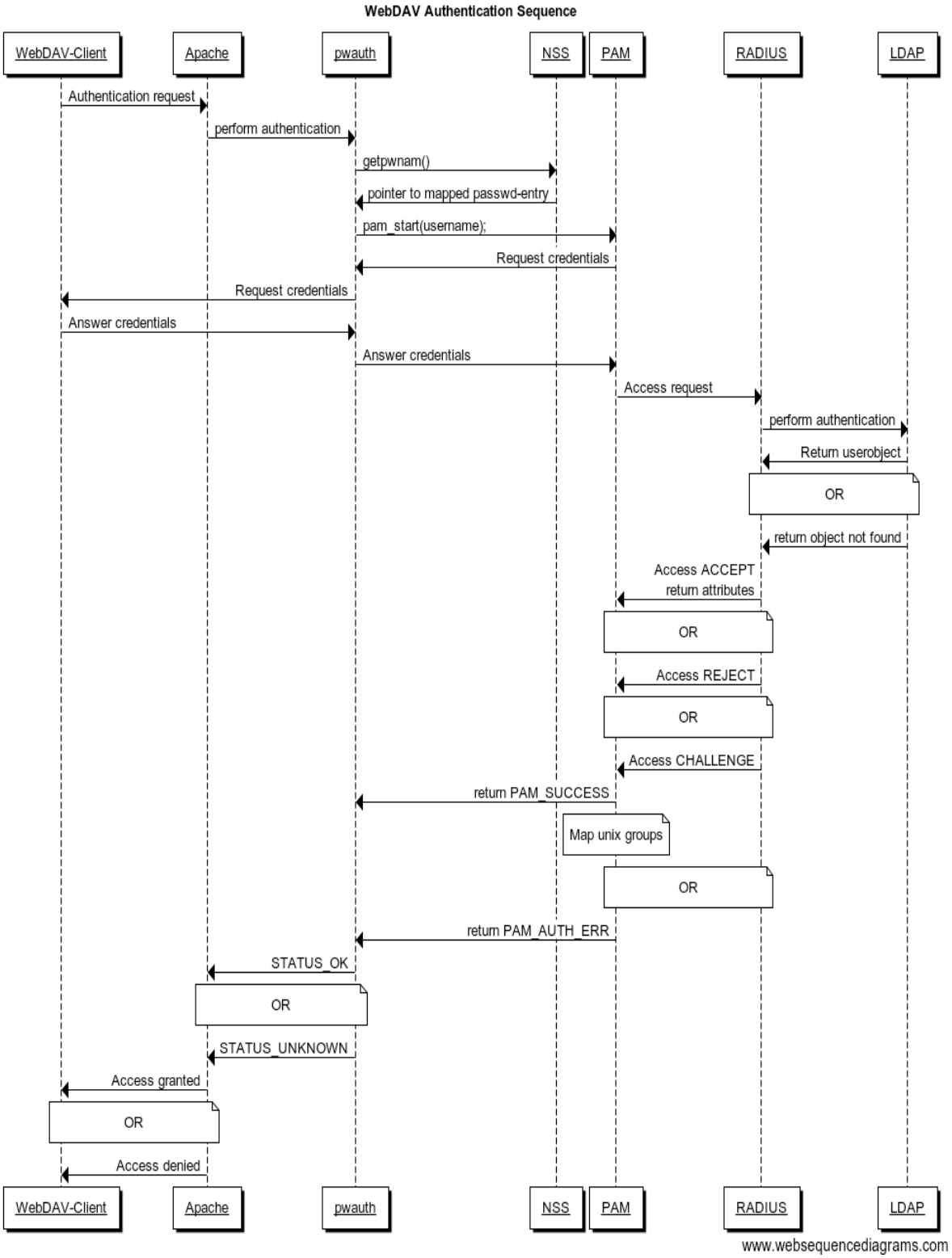
for example in an .htaccess file of a folder you can add the following lines:

```
require user john@moonshot.nl
require group nikhef-researcher
```

With the limit directive you can make a configuration for that user.
For example:

```
<Limit POST PUT DELETE>
        require group admin
</Limit>
```

Also see the sequence diagram of how the webdav authorization will work.


WebDAV Authentication Sequence
WebDAV-Client
Apache
pwauth
NSS
PAM
RADIUS
LDAP
Authentication request
perform authentication
getpwnam()
pointer to mapped passwd-entry
pam_start(username);
Request credentials
Request credentials
Answer credentials
Answer credentials
Access request
perform authentication
Return userobject
OR
return object not found
Access ACCEPT
return attributes
OR
Access REJECT
OR
Access CHALLENGE
return PAM_SUCCESS
Map unix groups
OR
return PAM_AUTH_ERR
STATUS_OK
OR
STATUS_UNKNOWN
Access granted
OR
Access denied
WebDAV-Client
Apache
pwauth
NSS
PAM
RADIUS
LDAP
www.websequencediagrams.com

7.6 Our project steps

Step 1: Research the WebDAV possibilities for the purpose of the project description
What do we need to use, patch or develop in order to reach a federated login for WebDAV with authorizing options. (Identification stage)

Step 2: Install and configure all the applications that are needed
Looking at our project we came up with the following: we had to install and configure centOS 6, WebDAV, radius, LDAP, PAM and several mods to binds those. (Development stage)

Step 3: Bug Fix and Test
Whether the configuration works or not, this phase is important. During this phase you do everything you can to fix problems or to test it so you´d be sure that no errors can occur. (Testing phase)

Step 4: Check if the requirements have been met
When the requirements are met we can go to the next point if available of completing our goal. When the requirements aren't met we have to go back to step 1 and find another combination to complete our goal. (Checking stage)

All these steps were taken many times. For example WebDAV wasn´t installed and configured at the same time as PAM. When WebDAV works like a charm, we can repeat the steps with the next installation and configuration.

7.7 Problems during the project

We encountered several problems during this project with the WebDAV progress.

7.7.1 The problems and how they were solved

Cadaver rewrite_mod problem:
Cadaver and many other clients don't support redirects in Apache. This is a common problem with WebDAV. To solve this problem we tested many clients and versions of several clients. We weren't able to get cadaver redirect ready.

Some clients don´t support redirects, others have the redirection in an experimental stage and some do actually support redirects.

mod_auth_pam problem:
At first we tried to connect WebDAV - mod_auth_pam - radius. But the connection didn´t work out. We got error after error. During the search for a solution we found that mod_auth_pam is no longer supported and/or developed. Because of this fact we had to search for something else to make a pam - radius connection. The solution was to use pwauth and mod_authnz_external instead of mod_auth_pam. These are still supported and developed. After installation the pam-radius worked. (installation of pwauth and mod_authnz_external are in the installation guide of WebDAV)

Quota problem:
Normally in WebDAV it is not possible to add a quota to a location. Mod_dav is a module that adds some more options, but doesn´t add the option to add an quota. There is a patch which provides this option to WebDAV. It doesn´t look like an official patch but it works. After patching apache it is possible to add a quota with the line “DAVSATMaxAreaSize 256000”.

(13) Permission denied problem:
After logging in to WebDAV at a moment we got a (13) permission denied. For this problem there was no easy solution. But after hours of research on the internet we found that the problem had to do with SElinux. Security-Enhanced Linux is a Linux feature that provides the mechanism for supporting access control security policies.

If you get the following error:
[Wed Sep 21 09:04:50 2011] [error] [client 192.168.10.89] (13)Permission denied: Could not open password file: /etc/shadow
[Wed Sep 21 09:04:50 2011] [error] [client 192.168.10.89] PAM: user 'root' - not authenticated: Permission denied

Open the directory /var/log/secure then install semanage by yum install libsemanage and execute: semanage permissive -a httpd_t

All folders visible problem:
A user that is logged in can see all the folders that are in the root directory, but we don´t want the user to see the folders of others. The solution to this problem is to add mod_rewrite. This adds the functionality that a user will be redirected to the folder that belongs to him.

Open the directory /var/www/webdav/data and then vim .htaccess

Add the following:

```
<IfModule mod_rewrite.c>
  RewriteEngine on
  RewriteRule ^$ /%{REMOTE_USER}/ [R=302,L]
</IfModule>
```

Next, Open the directory /etc/httpd/conf.d and then vim pam_auth.conf

Add the following between <Directory /var/www/webdav/data> and </Directory>:

RewriteEngine on
Options Indexes ExecCGI FollowSymLinks
AllowOverride all

Important: The user directory where the rewrite is going must exist! To protect a folder for users or groups place a .htaccess file in that directory with the line require user ´test´ or require group ´test´ !

7.7.2 The problem that couldn't be solved

One of the problems that we encountered during this project is the fact that we are not able to hide folders that a user has no access to in WebDAV. The user logged in to WebDAV enters the root of the directories. All the folders inside that directory are visible to the user. The user can only enter the folder in which he has the rights. To solve this problem we tried to redirect the user to his own folder and not the root directory. This can be done by apache's rewrite mod. Problem solved, at least we think. But not all of the WebDAV clients support apache's rewrite mod. One of the clients we use, ´Cadaver´ doesn´t allow us to rewrite a user to another directory. Looking at the code of a new version of cadaver this should be possible, but after testing it seems otherwise. Clients that support redirects are the built in webdav client of windows 7 and osx lion (tested). The fusedav 0.2.6 client seems closest to supporting redirects to what we've found, you only have to enter the folder with cd folder after you've been redirected.

7.8 Our WebDAV Configuration

Centos
Centos is the operating system we use for our project. CentOS stands for the Community ENTerprise Operating System and is an Enterprise-class Linux Distribution derived from sources freely provided to the public by a prominent North American Enterprise Linux vendor. CentOS exists to provide a free enterprise class computing platform and strives to maintain 100 % binary compatibility with it´s upstream source. CentOS is bases on Red Hat Enterprise Linux (RHEL). CentOS is developed by a small but growing team of core developers. In turn the core developers are supported by an active user community including system administrators, network administrators, enterprise users, managers, core Linux contributors and Linux enthusiasts from around the world. The biggest competitor of CentOS is Debian.

Source: http://www.centos.org
Version we use: Centos 6.2

Apache Webserver
The Apache HTTP Server Project is an effort to develop and maintain a open-source HTTP server for modern operating systems including UNIX and windows NT. The goal of the project is to provide a secure , efficient and extensible server that provides HTTP services in sync with the

current HTTP standards. Apache httpd is one of the most popular web servers of the internet. The Apache HTTP Server (httpd) is a project of The Apache Software Foundation.

Source: http://httpd.apache.org
Version we use: Apache 2.2

Pwauth
Pwauth is an authenticator designed to be used with mod_auth_external or mod_authnz_external and the Apache HTTP daemon to support reasonably secure web authentication out of the system password database on most versions of Unix.

Source: http://code.google.com/p/pwauth
Version we use: Pwauth 2.3.10

Mod_authnz_external
Mod_authnz_external is a flexible tool for building custom basis authentication systems for the Apache HTTP Daemon. Basic Authentication is a type of authentication build into the HTTP protocol, in which a login pops up when the user requests a protected resourse, and the login and passwords are checked by Apache. Mod_authnz_external allows the password checking normally done inside Apache to be done by an seperate external program running outside of Apache. One of the most common secure databases that people want to authenticate out of is the Unix system password database. The open sourse pwauth program is a mod_authnz_external authenticator that can do this. It can also authenticate from any PAM Authentication source.

Source: http://code.google.com/p/mod-auth-external
Version we use: Mod_authnz_external 3.2.4

Pam_radius_auth
Pam_radius_auth is the PAM to RADIUS authentication module. It allows any PAM-capable machine to become a RADIUS client for authentication and accounting requests. You will need a RADIUS server to perform the actual authentication.

Source: http://www.freeradius.org/pam_radius_auth
Version we use: Pam_radius_auth 1.3.17

7.9 Pro´s and Con´s of our work

The big pro of our work is that we have a well federated way of logging in with WebDAV through PAM with mostly standard configurations. We didn´t have to code, patch or develop our own add-ons. The things we did had to do with installing and configuring. So it is possible to federate WebDAV with authorizing options when we only use existing applications and protocols. One of the con´s has to do with lack of security. In our configuration we are not using GSS yet, due to a shortage in time.

8 Future Research

For the purpose of this project we have created a demonstration and thought several things through. This however opens doors for more possibilities and things which are left to research. These things will be discussed in this chapter and we'd like to encourage you to dig into it.

8.1 GSS

With the use of GSS (GNU Generic Security Service) it is currently possible to authenticate from the SSH client to the SSH server through a EAP-TTLS connection. However this would require a patch SSH client and server. This lacks of support of most SSH clients and servers. This method also doesn't offer authorization options. This only regards the ssh service other services are left to research or patch. Things to expand in the future would be:

- Broader support for SSH clients and servers
- Ability to use authorization modules on SSH server, possibly through PAM though keeping it secured with EAP-TTLS
- Broaden the support of a secure connection for other services such as Webdav / SFTP by adding support for GSS or other security methods.

8.2 WebDAV redirect support

A lot of WebDAV clients don´t support redirects. In order to make the WebDAV federated authentication more properly redirects must be supported. Because it is neatly that the folders of others won´t be visible. Also the authorization in WebDAV can get more valuable. With redirects you can give certain groups their own directory more easily based on unix groups. To live up to that expectation it is necessary to patch or upgrade some of the available WebDAV clients.

8.3 PAM module expansion

At the moment we can only give small authorization options through PAM to SSH and WebDAV. For the future an PAM module expansion can give full authorization options to different applications at the same time. Ideality would be to have one PAM module that works for a lot of different applications. Which can handle multiple attributes and map users to a pool account based on those attributes.

8.4 XACML support

XACML authorization options can be added. The XACML allows for generic access control policies in XML format. Authorization policies today are formulated for individual services. This makes it difficult to state the access rights for individual users. XACML allows for unified authorization policies: Use one consistent XACML policy for multiple services!

8.5 Kerberos

In the future Kerberos might be added to our configuration to make the connection more secure. Kerberos is a computer network authentication protocol which works on the basis of "tickets" to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner. Its designers aimed primarily at a client–server model, and it provides mutual authentication—both the user and the server verify each other's identity. Kerberos protocol messages are protected against eavesdropping and replay attacks. Kerberos builds on symmetric key cryptography and requires a trusted third party, and optionally may use public-key cryptography by utilizing asymmetric key cryptography during certain phases of authentication.

Appendix A

```
#include <nss.h>
 #include <string.h>
 #include <stdlib.h>
 #include <pwd.h>

 enum nss_status _nss_moonshot_getpwnam_r(
        const char *name,
        struct passwd *result,
        char *buffer,
        size_t buflen,
        int *errnop
        )
{
        struct passwd *respointer=NULL;
        getpwuid_r(509, result, buffer, buflen, &respointer);

        return NSS_STATUS_SUCCESS;
}

enum nss_status _nss_moonshot_getpwuid_r(
        uid_t uid,
        struct passwd *result,
        char *buffer,
        size_t buflen,
        int *errnop
        )
{
        struct passwd *respointer=NULL;
        getpwuid_r(509, result, buffer, buflen, &respointer);

        return NSS_STATUS_SUCCESS;
}
```

Appendix B

```
#include <grp.h>
#include <pwd.h>

#include <stdio.h>
#include <string.h>
#include <syslog.h>

#define PAM_SM_AUTH

#include <security/pam_appl.h>
#include <security/pam_modules.h>

char *ltrim(char *trim) {
	while (isspace(*trim) || !isalnum(*trim)) {
		trim++;
	}
	return trim;
}

char *rtrim(char *trim){
	char* end = trim + strlen(trim);
	while(!isalnum(*end--)){

	}
	*(end+2) = '\0';
	return trim;
}

char *lrtrim(char *trim) {
	return rtrim(ltrim(trim));
}

void baslog (char *str)
{
	openlog ("pam_radius_group", LOG_PERROR, 1);
	syslog (LOG_INFO, str);
	closelog ();
}

char *get_radius_unixgroup(char *username, char *password) {
	FILE *in;
	char buffer[256];
	char temp[11];

	char unixGroup[256];

	char connectionString[256];
```

```
	sprintf(connectionString, "radtest %s %s 10.198.5.33 1812 testing123", username,
password);

	in = popen(connectionString, "r");
	if (in == NULL) {
		return;
	}
	while (fgets(buffer, 256, in) != NULL) {
		strncpy(temp, ltrim(buffer), 10);
		temp[10] = '\0';
		if (strcmp(temp, "Unix-Group") == 0) {
			strcpy(unixGroup, lrtrim(strstr(buffer, "=")));
		}
	}

	return unixGroup;
}

PAM_EXTERN int
pam_sm_acct_mgmt(pam_handle_t *pamh, int flags, int argc, const char *argv[])
{
	const char *group, *user;
	char **members;

	struct passwd *pwd;
	struct group *grp;

	/* get target account */

	if (pam_get_user(pamh, &user, NULL) != PAM_SUCCESS ||
	    user == NULL || (pwd = getpwnam(user)) == NULL)
		return (PAM_AUTH_ERR);
	/*
	if (pam_get_item(pamh, &authtok, NULL) != PAM_SUCCESS) {
		return (PAM_AUTH_ERR);
```

```
}
	*/
	/*
	if (pam_get_item(pamh, PAM_AUTHTOK, (void **) &authtok) != PAM_SUCCESS) {
		baslog(pam_strerror(pamh, pam_get_item(pamh, PAM_AUTHTOK, (void **)
&authtok)));
	}
	*/

	char ugrp[256];
	strcpy(ugrp, get_radius_unixgroup(user, "a"));

	grp = getgrnam(ugrp);

	if (grp != NULL) {
		char buf[256];
		sprintf(buf, "/usr/sbin/usermod --append -G %s %s", grp->gr_name, pwd->pw_name);
		baslog(buf);
		system(buf);
		return (PAM_SUCCESS);
	} else {
		baslog("Group not found!");
		return (PAM_IGNORE);
	}
}
```

Appendix B2

```
#include <grp.h>
#include <pwd.h>

#include <stdio.h>
#include <string.h>
#include <syslog.h>

#define PAM_SM_SESSION

#include <security/pam_appl.h>
#include <security/pam_modules.h>

void baslog (char *str)
{
	openlog ("pam_radius_removegroup", LOG_PERROR, 1);
	syslog (LOG_INFO, str);
	closelog ();
}

PAM_EXTERN int
pam_sm_open_session(pam_handle_t *pamh, int flags, int argc, const char *argv[])
{
	baslog("Session opened");
	return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_close_session(pam_handle_t *pamh, int flags, int argc, const char *argv[])
{
	const char *user;
	struct passwd *pwd;

	baslog("Close session");

	pam_get_user(pamh, &user, NULL);
	pwd = getpwnam(user);

	char buf[256];
	sprintf(buf, "/usr/sbin/usermod -G \"\" %s", pwd->pw_name);
	baslog("Session closed");
	system(buf);
	return (PAM_SUCCESS);
}
```

Appendix C

Debug log of a RADIUS Request

```
rad_recv: Access-Request packet from host 10.198.5.35 port 20703, id=103, length=97
        User-Name = "john@moonshot.nl"
        User-Password = "moonshot"
        NAS-IP-Address = 10.198.5.35
        NAS-Identifier = "sshd"
        NAS-Port = 19678
        NAS-Port-Type = Virtual
        Service-Type = Authenticate-Only
        Calling-Station-Id = "localhost"
# Executing section authorize from file /etc/raddb/sites-enabled/default
+- entering group authorize {...}
++[preprocess] returns ok
++[chap] returns noop
++[mschap] returns noop
++[digest] returns noop
[suffix] Looking up realm "moonshot.nl" for User-Name = "john@moonshot.nl"
[suffix] Found realm "moonshot.nl"
[suffix] Adding Stripped-User-Name = "john"
[suffix] Adding Realm = "moonshot.nl"
[suffix] Authentication realm is LOCAL.
++[suffix] returns ok
[eap] No EAP-Message, not doing EAP
++[eap] returns noop
++[files] returns noop
[ldap] performing user authorization for john
[ldap]   expand: %{Stripped-User-Name} -> john
[ldap]   expand: (uid=%{%{Stripped-User-Name}:-%{User-Name}}) -> (uid=john)
[ldap]   expand: dc=moonshot,dc=nl -> dc=moonshot,dc=nl
 [ldap] ldap_get_conn: Checking Id: 0
 [ldap] ldap_get_conn: Got Id: 0
 [ldap] performing search in dc=moonshot,dc=nl, with filter (uid=john)
[ldap] checking if remote access for john is allowed by uid
[ldap] looking for check items in directory...
 [ldap] unixGroup -> Unix-Group == "project-moonshot"
 [ldap] userPassword -> Cleartext-Password == "moonshot"
 [ldap] userPassword -> Password-With-Header == "moonshot"
[ldap] looking for reply items in directory...
 [ldap] unixGroup -> Unix-Group = "project-moonshot"
 [ldap] radiusTunnelPrivateGroupId -> Tunnel-Private-Group-Id:0 = "10"
 [ldap] radiusTunnelMediumType -> Tunnel-Medium-Type:0 = 802
 [ldap] radiusTunnelType -> Tunnel-Type:0 = VLAN
[ldap] user john authorized to use remote access
 [ldap] ldap_release_conn: Release Id: 0
++[ldap] returns ok
++[expiration] returns noop
++[logintime] returns noop
[pap] Config already contains "known good" password.  Ignoring Password-With-Header
```

```
++[pap] returns updated
Found Auth-Type = PAP
# Executing group from file /etc/raddb/sites-enabled/default
+- entering group PAP {...}
[pap] login attempt with password "moonshot"
[pap] Using clear text password "moonshot"
[pap] User authenticated successfully
++[pap] returns ok
# Executing section post-auth from file /etc/raddb/sites-enabled/default
+- entering group post-auth {...}
++[exec] returns noop
Sending Access-Accept of id 103 to 10.198.5.35 port 20703
        Unix-Group = "project-moonshot"
        Tunnel-Private-Group-Id:0 = "10"
        Tunnel-Medium-Type:0 = 802
        Tunnel-Type:0 = VLAN
Finished request 12.
Going to the next request
Waking up in 4.9 seconds.
rad_recv: Access-Request packet from host 10.198.5.35 port 33569, id=8, length=68
        User-Name = "john@moonshot.nl"
        User-Password = "a"
        NAS-IP-Address = 10.198.5.35
        NAS-Port = 1812
# Executing section authorize from file /etc/raddb/sites-enabled/default
+- entering group authorize {...}
++[preprocess] returns ok
++[chap] returns noop
++[mschap] returns noop
++[digest] returns noop
[suffix] Looking up realm "moonshot.nl" for User-Name = "john@moonshot.nl"
[suffix] Found realm "moonshot.nl"
[suffix] Adding Stripped-User-Name = "john"
[suffix] Adding Realm = "moonshot.nl"
[suffix] Authentication realm is LOCAL.
++[suffix] returns ok
[eap] No EAP-Message, not doing EAP
++[eap] returns noop
++[files] returns noop
[ldap] performing user authorization for john
[ldap]   expand: %{Stripped-User-Name} -> john
[ldap]   expand: (uid=%{%{Stripped-User-Name}:-%{User-Name}}) -> (uid=john)
[ldap]   expand: dc=moonshot,dc=nl -> dc=moonshot,dc=nl
 [ldap] ldap_get_conn: Checking Id: 0
 [ldap] ldap_get_conn: Got Id: 0
 [ldap] performing search in dc=moonshot,dc=nl, with filter (uid=john)
[ldap] checking if remote access for john is allowed by uid
[ldap] looking for check items in directory...
 [ldap] unixGroup -> Unix-Group == "project-moonshot"
 [ldap] userPassword -> Cleartext-Password == "moonshot"
 [ldap] userPassword -> Password-With-Header == "moonshot"
```

```
[ldap] looking for reply items in directory...
 [ldap] unixGroup -> Unix-Group = "project-moonshot"
 [ldap] radiusTunnelPrivateGroupId -> Tunnel-Private-Group-Id:0 = "10"
 [ldap] radiusTunnelMediumType -> Tunnel-Medium-Type:0 = 802
 [ldap] radiusTunnelType -> Tunnel-Type:0 = VLAN
[ldap] user john authorized to use remote access
 [ldap] ldap_release_conn: Release Id: 0
++[ldap] returns ok
++[expiration] returns noop
++[logintime] returns noop
[pap] Config already contains "known good" password.  Ignoring Password-With-Header
++[pap] returns updated
Found Auth-Type = PAP
# Executing group from file /etc/raddb/sites-enabled/default
+- entering group PAP {...}
[pap] login attempt with password "a"
[pap] Using clear text password "moonshot"
[pap] Passwords don't match
++[pap] returns reject
Failed to authenticate the user.
Using Post-Auth-Type Reject
 WARNING: Unknown value specified for Post-Auth-Type.  Cannot perform requested action.
# Executing group from file /etc/raddb/sites-enabled/default
Delaying reject of request 13 for 1 seconds
Going to the next request
Waking up in 0.9 seconds.
Sending delayed reject for request 13
Sending Access-Reject of id 8 to 10.198.5.35 port 33569
        Unix-Group = "project-moonshot"
        Tunnel-Private-Group-Id:0 = "10"
        Tunnel-Medium-Type:0 = 802
        Tunnel-Type:0 = VLAN
Waking up in 3.9 seconds.
rad_recv: Accounting-Request packet from host 10.198.5.35 port 20701, id=140, length=84
        User-Name = "john@moonshot.nl"
        NAS-IP-Address = 10.198.5.35
        NAS-Identifier = "sshd"
        NAS-Port = 19676
        NAS-Port-Type = Virtual
        Acct-Status-Type = Start
        Acct-Session-Id = "00019676"
        Acct-Authentic = RADIUS
# Executing section preacct from file /etc/raddb/sites-enabled/default
+- entering group preacct {...}
++[preprocess] returns ok
[acct_unique] Hashing 'NAS-Port = 19676,Client-IP-Address = 10.198.5.35,NAS-IP-Address =
10.198.5.35,Acct-Session-Id = "00019676",User-Name = "john@moonshot.nl"'
[acct_unique] Acct-Unique-Session-ID = "3d2b27370402f017".
++[acct_unique] returns ok
[suffix] Looking up realm "moonshot.nl" for User-Name = "john@moonshot.nl"
[suffix] Found realm "moonshot.nl"
```

```
[suffix] Adding Stripped-User-Name = "john"
[suffix] Adding Realm = "moonshot.nl"
[suffix] Accounting realm is LOCAL.
++[suffix] returns ok
++[files] returns noop
# Executing section accounting from file /etc/raddb/sites-enabled/default
+- entering group accounting {...}
[detail] expand: /var/log/radius/radacct/%{Client-IP-Address}/detail-%Y%m%d ->
/var/log/radius/radacct/10.198.5.35/detail-20120626
[detail] /var/log/radius/radacct/%{Client-IP-Address}/detail-%Y%m%d expands to
/var/log/radius/radacct/10.198.5.35/detail-20120626
[detail] expand: %t -> Tue Jun 26 20:22:05 2012
++[detail] returns ok
++[unix] returns ok
[radutmp]       expand: /var/log/radius/radutmp -> /var/log/radius/radutmp
[radutmp]       expand: %{User-Name} -> john@moonshot.nl
++[radutmp] returns ok
++[exec] returns noop
[attr_filter.accounting_response]       expand: %{User-Name} -> john@moonshot.nl
attr_filter: Matched entry DEFAULT at line 12
++[attr_filter.accounting_response] returns updated
Sending Accounting-Response of id 140 to 10.198.5.35 port 20701
Finished request 14.
```

FreeRADIUS Installation Manual

Installation and configuration of the radius server

Install freeradius:

yum install freeradius.x86_64

Version check of the radius server:

radiusd -v

We install the freeradius utilities to test the radius configuration, :

yum install freeradius-utils.x86_64

We install the freeradius-ldap module to connect the radius server to ldap:

yum install freeradius-ldap.x86_64

The freeradius server gets connected to LDAP, so you have to look at the 'userPassword' field in the LDAP-object. To do this you have to add "checkItem Cleartext-Password userPassword" to the '/etc/raddb/ldap.attrmap':

vim /etc/raddb/ldap.attrmap

At the end of the list we add the checkItem of our new check:

checkItem Cleartext-Password userPassword

Now we are going to add the ldap/module to the authorize-section of '/etc/raddb/sites-available/inner-tunnel':

vim /etc/raddb/sites-available/inner-tunnel

Search for ldap:

/ldap

When found "uncomment" this and search for the next ldap, you have to "uncomment" this function also:

```
Auth-Typ LDAP{
        ldap
}
```

We still have to add the LDAP module to the authorize-section of /etc/raddb/sites-available/default:

vim /etc/raddb/sites-available/default

Search for ldap:

/ldap

When found "uncomment" this and search for the next ldap, you have to "uncomment" this function also:

```
Auth-Typ LDAP{
        ldap
}
```

The next step is to configure the LDAP-module:

```
vim /etc/raddb/modules/ldap
ldap {
        server = "{IP LDAP-server}"
        identity = "cn=Manager,dc=moonshot,dc=nl"
```

password = slapwachtwoord
basedn = “dc=moonshot,dc=nl”

The freeradius daemon needs to be configured that it only listens to one ip-address instead of all ip-addresses:

vim /etc/raddb/radiusd.conf

Search for ipaddr under the listen{}

/ipaddr
*Replace * by the ip-address of the radius server (two times)*

Add a radius client:

vim /etc/raddb/clients.conf

At client localhost we add the client radtest:

client localradtest{
ipaddr = 10.198.5.137 (van de client)
secret = testing123
reguire_message_authenticator = no
nastype = other
}

Radius and ldap testing by a radtest with the client we made before:

radtest bas hans 10.198.5.37 12 testing123

This is the response:

Sending Access-Request of id 211 to 10.198.5.37 port 1812
User-Name = "bas"
User-Password = "hans"
NAS-IP-Address = 10.198.5.37
NAS-Port = 12
rad_recv: Access-Accept packet from host 10.198.5.37 port 1812, id=211, length=36
Tunnel-Private-Group-Id:0 = "10"
Tunnel-Medium-Type:0 = 802
Tunnel-Type:0 = VLAN

At last we configure the radius daemon that it restarts at a reboot::

chkconfig radiusd on

If done correctly, you know made a radius - ldap connection.

Add the realms in the proxy.conf to connect to radius servers to each other:

vim /etc/raddb/proxy.conf
realm moonshot.nl{
type = radius
authhost = 10.198.5.33:1812
accthost = 10.198.5.33:1813
secret = duitsland
nostrip
}

realm carp.nl{
type = radius

```
        authhost = LOCAL
        accthost = LOCAL
        secret = testing123
    }
```

To test this do a radtest again:

```
radtest "bas@moonshot.nl" "hans" 10.198.5.37 1812 testing123
```

The response of the radtest:

```
Sending Access-Request of id 169 to 10.198.5.37 port 1812
        User-Name = "bas@moonshot.nl"
        User-Password = "hans"
        NAS-IP-Address = 10.198.5.37
        NAS-Port = 1812
rad_recv: Access-Accept packet from host 10.198.5.37 port 1812, id=169, length=36
        Tunnel-Private-Group-Id:0 = "10"
        Tunnel-Medium-Type:0 = 802
        Tunnel-Type:0 = VLAN
```

Make a central point

vim /etc/raddb/proxy.conf

```
realm moonshot.nl{
        type = radius
        authhost = 10.198.5.33:1812
        accthost = 10.198.5.33:1813
        secret = testing123
        nostrip
}

realm carp.nl{
        type = radius
        authhost = LOCAL
        accthost = LOCAL
        secret = testing123
        nostrip
    }
```

Default ip address of our central radius server added to the other radius servers and null on the local radius server.

```
vim /etc/raddb/proxy.conf
```

WebDAV Installation Manual

Installation

Install Centos 6.2

For download and installation see http://www.centos.org/.

Install Apache (HTTPD) 2.2

yum install httpd

service httpd start

Turn on and configure WebDAV in Apache
cd /etc/httpd/conf
vim httpd.conf
where the other modules, before are loaded
before: “Include conf.d/*.conf” add:

LoadModule authnz_external_module /usr/lib64/httpd/modules/mod_authnz_external.so
LoadModule authz_unixgroup_module /usr/lib64/httpd/modules/mod_authz_unixgroup.so
AddExternalAuth pwauth /usr/local/libexec/pwauth
SetExternalAuthMethod pwauth pipe
---

cd /etc/httpd/conf.d

vim auth_pam.conf

<Directory /var/www/webdav/data>
DAV ON

Options Indexes ExecCGI FollowSymLinks
AllowOverride all

AuthType Basic
AuthName “PAM Authentication”
AuthBasicProvider external
AuthExternal pwauth

AuthzUnixgroup on

require valid-user

IndexIgnore *.bak .??* *~ *# HEADER* README* RCS CVS *,v *,t
</Directory>

Install Radius and LDAP

See the Radius and LDAP installation manual.

NSS
Get libnss_moonshot.c

Edit the libnss_moonshot.c file and add the preferred name, uid and gid.

Compile using:
gcc -shared -o libnss_moonshot.so.2 -Wl,-soname,llibnss_moonshot.so.2 OBJECTS -fPIC

Move the libnss_moonshot.so.2 to /lib64

Edit /etc/nsswitch.conf to
passwd: files moonshot

Install pam_radius_auth

Download and extract

wget ftp://ftp.freeradius.org/pub/radius/pam_radius-1.3.17.tar.gz
tar -zxvf pam_radius-1.3.17
cd pam_radius-1.3.17
make
mv pam_radius_auth.so /lib/security/pam_radius_auth.so

Configure pam_radius_auth
vi /etc/raddb/server:
---
server[:port] shared_secret timeout (s)
127.0.0.1 secret 1
other-server other-secret 3
---

chown root /etc/raddb
chmod go-rwx /etc/raddb
chmod go-rwx /etc/raddb/server

Install mod_authnz_external

Check if Dynamically Loaded Modules are supported

httpd -l

Download and extract

wget http://mod-auth-external.googlecode.com/files/mod_authnz_external-3.2.6.tar.gz
tar -zxvf mod_authnz_external-3.2.6.tar.gz

Compile the module

```
# cd mod_authnz_external-3.2.6
# apxs -c mod_authnz_external.c
```

Install the module

```
# apxs -i -a mod_authnz_external.la
```

Download mod_authz_unixgroup
http://code.google.com/p/mod-auth-external/downloads/list

Compile the module

```
# apxs -c mod_authz_unixgroup.c
```

Install the module

```
# apxs -i -a mod_authz_unixgroup.la
```

Install pwauth

Download and extract

```
# wget http://pwauth.googlecode.com/files/pwauth-2.3.10.tar.gz
# tar -zxvf pwauth-2.3.10.tar.gz
```

Edit config.h

```
# cd pwauth-2.3.10.tar.gz
# vim config.h
```

Uncomment the line
#define PAM /* Linux PAM or OpenPAM */

optionally define from which userid the users may login
#define MIN_UNIX_UID 500 /**/

Uncomment the line: and change the userid to the one of wwwrun
#define SERVER_UIDS 48 /* user "wwwrun" on the author's system */

Pam configuration

```
# cd /etc/pam.d
# vim pwauth
```

```
---------------------------------------------------------------------------------------------------------------
auth            required        pam_radius_auth.so debug try_first_pass
account         include         system-auth
session         include         password-auth
---------------------------------------------------------------------------------------------------------------
```

install pwauth

chown root pwauth
chmod u+s pwauth

(13) Permission denied fix

SElinux disable httpd_t

cd /var/log/secure
yum install semanage
semanage permissive -a httpd_t

Quota apache patch

wget http://leche.goodcrew.ne.jp/webdav/webdav-2.2.11-quota-2.4any.txt
wget http://archive.apache.org/dist/httpd/httpd-2.2.3.tar.gz
tar -xvzf httpd-2.2.3.tar.gz
cd httpd-2.2.3
patch -p2 < /home/user/webdav-2.2.11-quota-2.4any.txt
./configure --enable-modules=most --enable-mods-shared=all
mv /home/user/httpd-2.2.3/modules/dav/main/.libs/mod_dav.so
/usr/lib/httpd/modules/mod_dav.so
mv /home/user/httpd-2.2.3/modules/dav/fs/.libs/mod_dav_fs.so
/usr/lib/httpd/modules/mod_dav_fs.so

Now you can use DAVSATMaxAreaSize 256000 in the apache config in the location but also the subdirectories. Where the size is in kb.

Troubleshooter

For extra hints, tips or help see these websites:

Centos - http://www.centos.org
Apache - http://httpd.apache.org
Pwauth - http://code.google.com/p/pwauth
Mod_authnz_external - http://code.google.com/p/mod-auth-external
Pam_radius_auth - http://www.freeradius.org/pam_radius_auth

Windows 7

The default webdav client supports redirects. If there is a problem with authenticating (basic auth) in windows 7 there might have to be some changes in the registry. Under

HKLM\SYSTEM\CurrentControlSet\Services\WebClient\Parameters:

Type: DWORD (32-bit) Value

Value: 2

If you do this, you should make sure the Web-application uses SSL since the Basic Auth password is sent in cleartext.

Note that the possible values you can assign to this new registry entry are:

0 - Basic authentication disabled

1 - Basic authentication enabled for Secure Sockets Layer (SSL) shares only

2 or greater - Basic authentication enabled for SSL shares and for non-SSL shares

And under HKLM\SYSTEM\CurrentControlSet\Services\WebClient\Parameters\

ServerNotFoundCacheLifeTimeInSec set the registry entry to 0 (default is 60 or 3c in hex) now adding the webdav in windows 7 should work properly supporting redirects.

Mac OS X
At least 10.7 supports redirects.

OpenLDAP Installation Manual

Download and install OpenLDAP.

yum install openldap-servers

Create password for LDAP-administration.

slappasswd

We used the standard FreeRADIUS schema to insert in the LDAP configuration. Therefore we copied radius.schema from our FreeRADIUS server by issueing a Secure Copy command.

scp user@domain:filename local_destination

Create the configuration file /etc/openldap/slapd.conf. This file is not created by the installation but it is an option in the slapd init-script.

vim /etc/openldap/slapd.conf

Configure /etc/openldap/slapd.conf. Include the standard schema's, allow LDAPv2, configure database, configure suffix, checkpoint, rootdn

include /etc/openldap/schema/radius.schema
include /etc/openldap/schema/unix.schema
database bdb
suffix "dc=moonshot,dc=nl"
checkpoint 1024 15
rootdn "cn=Manager,dc=<domain>,dc=<toplevel-domain>"

directory /var/lib/ldap
index objectClass eq,pres
index uid,memberUid eq,pres,sub

Make sure that the owner of the directory /var/lib/ldap/ is user ldap.

chown -R ldap:ldap /var/lib/ldap/

Start OpenLDAP

service slapd start

If slapd can not start due to a "Permission Denied"-error maybe the selinux context of the radius.schema file is not correct. You can restore it with this command:

restorecon /etc/openldap/schema/radius.schema

Install OpenLDAP-clients

yum install openldap-clients

You can now insert .ldif files with the ldapadd command. An .ldif file can look like this:

 # moonshot.nl
dn: dc=moonshot,dc=nl
objectClass: dcObject
objectClass: organization
o: moonshot
dc: moonshot

```
# people, moonshot.nl
dn: ou=people,dc=moonshot,dc=nl
objectClass: organizationalUnit
ou: people
description: people

# bas, people, moonshot.nl
dn: uid=bas,ou=people,dc=moonshot,dc=nl
objectClass: top
objectClass: radiusprofile
objectClass: inetOrgPerson
cn: bas
sn: bas
uid: bas
description: user Bas
radiusTunnelType: VLAN
radiusTunnelMediumType: 802
radiusTunnelPrivateGroupId: 10
userPassword:: aGFucw==
```

The command:

```
ldapadd -h localhost -W -D "cn=Manager,dc=moonshot,dc=nl" -f initial.ldif
```

Make sure that the firewall is not blocking the LDAP ports. You can allow ports with the iptables command.

If you want to start OpenLDAP at system startup issue the following command:

```
chkconfig slapd on
```