Hogeschool van Amsterdam
Amsterdam University of Applied Sciences

Project From Moonshot to Mars

Research Report JaNET Moonshot

Remy Bien, Ruben Bras, Marvin Hiemstra,
Sebastiaan Groot, Wouter Miltenburg, Koen Veelenturf

10 June 2013,
Version 0.7

NIKHEF

Table of contents

Introduction

This research document was created in order to give an insight into the research done by ITopia. The chapter ‘JaNET LiveDVD’ describes the modifications JaNET has made to create their Moonshot environment. It also contains a description of usable parts for our own version of the Moonshot project. The chapter ‘Port JaNET CentOS’ will focus on the known issues and if a port to CentOS is possible. Furthermore, the way JaNET’s Moonshot project works will be discussed. This document is part of a set of documents, the other document describes the FreeRADIUS module, our proposed solution to make Moonshot secure.

JaNET LiveDVD

JaNET

JaNET is a private organisation, which provides a computer network and related collaborative services to UK research and education. The UK government funds the organisation. All further- and higher-education organisations in the UK are connected to JaNET's network, as are all the research councils. The network also carries traffic between schools within the UK. The name of the organisation was originally a contraction of *Joint Academic NETwork* but it is now known as JaNET.

Moonshot

The goal of the Moonshot technology is to enable the mangement of access to a broad range of services and applications, using a single technology and infrastructure. Moonshot is expected to significantly improve the delivery of these services by providing users with a common single sign-on, for both internal and external services. Service providers will be able to (more) easily offer their services to users from other organisations using a single common authentication mechanism.

LiveDVD

The JaNET LiveDVD contains a setup and a live operating system – Debian Wheezy – that contains the Moonshot packages and libraries. This is the easiest way to play with the Moonshot environment. The latest release of the Moonshot LiveDVD (2013.04.21) contains FreeRADIUS 3.x (including the trust_router client support) and significant updates to the UI. It also uses RADSEC, and therefore a small amount of manual certificate configuration is necessary. The team used the first release of the JaNET Moonshot LiveDVD (2013.03.07) for its research.

Difference in code

JaNET needed to make some changes in the source code of the SSH daemon in order to make federated login possible. They created some extra files in the source code in order to make it work, and they made some adjustment to the current source code. In the rest of the chapter we will highlight some of those changes and some of the new files.

We tried to compare JaNET's source code to the official OpenSSH 5.9p1 source from OpenBSD. By comparing the different sources we were able to find out the differences between them.

One of the interesting differences in the source code is located in the "auth2-gss.c" file. The file itself exists on every OpenSSH installation, but JaNET made some changes. They added two extra functions of code for GSSAPI key exchange, named: "userauth_gsskeyex", "gssapi_set_username(Authctxt *authctxt)". And they added a new authentication method (struct): "Authmethod method_gsskeyex". The first new part of code is the new userauth mechanism for the GSSAPI key exchange. When a user wants to login, it will return "authenticated=1" or "authenticated=NULL". This piece of code will call the gssapi_set_username. This function will set the GSSAPI context for the username and password to the PAM module authctxt, if the option USE_PAM is enabled.

Another file where some major adjustments have been made is the file “ssh-gss.h”. The original file contains a part for MIT Kerberos. In the version of JaNET’s source code, this part is gone. A lot of new code has been added to the file. This is especially for the SSH / GSSAPI key exchange. This is probably because JaNET made its own GSSAPI exchange, instead of using the standard Kerberos GSSAPI exchange method. The following code is new in the *ssh-gss.h* file:

```
…
#define SSH2_MSG_KEXGSS_INIT                30
#define SSH2_MSG_KEXGSS_CONTINUE            31
#define SSH2_MSG_KEXGSS_COMPLETE            32
#define SSH2_MSG_KEXGSS_HOSTKEY             33
#define SSH2_MSG_KEXGSS_ERROR               34
#define SSH2_MSG_KEXGSS_GROUPREQ            40
#define SSH2_MSG_KEXGSS_GROUP               41
#define KEX_GSS_GRP1_SHA1_ID                "gss-group1-sha1-"
#define KEX_GSS_GRP14_SHA1_ID               "gss-group14-sha1-"
#define KEX_GSS_GEX_SHA1_ID                 "gss-gex-sha1-"
…
/* in the server */
typedef int ssh_gssapi_check_fn(Gssctxt **, gss_OID, const char *,
   const char *);
char *ssh_gssapi_client_mechanisms(const char *, const char *);
char *ssh_gssapi_kex_mechs(gss_OID_set, ssh_gssapi_check_fn *, const char *,
   const char *);
gss_OID ssh_gssapi_id_kex(Gssctxt *, char *, int);
int ssh_gssapi_server_check_mech(Gssctxt **,gss_OID, const char *,
   const char *);
…
char *ssh_gssapi_server_mechanisms(void);
int ssh_gssapi_oid_table_ok();
…
```

This code will probably be the initiation of the variables for the SSH Moonshot-GSSAPI server mechanism. There is an indication that this code is made for the SSH server side, because of a comment in the code (*in the server*).

Changed files OpenSSH

JaNET has changed quite a lot of the OpenSSH directory in version 5.9. They have removed the sandbox files and the gss-serv-krb5.c and added files like changeLog.gssapi, kexgsss.c and kexgssc.c. JaNET edited a lot of files as well like gss-serv-krb5.c, auth2-gss.c and ssh-gss.h. For the full overview of changes, please see the appendix. Due to the amount of changes JaNET did to their Test Environment and without their documentation, it is hard to pinpoint the essence of their work. We could use some of their work as an example but it takes quite some time to implement it for our purpose. Because of this we had to consider if we would put considerable time in analysing their work, at the cost of having less time for our own solution.

Interesting files

In this paragraph we list a number of interesting files:

auth2-gss.c
(Include) monitor_wrap.h
Line 40 deleted
Added: OM_uint32 mm_ssh_gssapi_localname / OM_uint32 mm_ssh_gssapi_sign
(Include) packet.h
Line added: "packet_connection_is_ipv4(void)"

mod_auth_kerb
mod_auth_gssapi.c
ssh-gss.h
radius_example.c
eap_example
kexgsss.c

Unlocatable files

Here you will find a list of files, which are not present on the JaNET LiveDVD installation but are mentioned in the original installations or referred to in the code.

Security/AuthSession.h

Points of attention

A list of things that caught our attention while looking through the JaNET LiveDVD installation.
/moonshot/moonshot/libeap/radiusexample (radius_example.c)
Radius configuration build in c. Not yet analysed.

/moonshot/moonshot/libeap/eap_example (certificates (.pem))
They have experimented with certificates as well, unknown how far they have got or what the results are.

Network traffic

This chapter describes the observed network traffic between the SSH client, identity card application, SSH server and radius server on the Moonshot LiveDVD.

The SSH client first sets up a TCP connection with the SSH server, after which they complete the SSH key exchange. Seemingly directly after the key exchange, the SSH client opens the Moonshot identity card application in which the user selects an identity to "send". At this point, either the identity card application or the SSH client itself, connects to the FreeRADIUS server and initiates an EAP-TTLS inner-tunnel. Once the secure tunnel is set up, authentication against the radius server is performed and an Access-Accept is returned with a vendor-specific attribute containing a SAML assertion. This is used to finalize the authentication and the SSH server gives the user (in this case *steve*) the generic "Moonshot" account on the server.

Network Flow

Scenario 1

This chapter will describe how a connection is made using gss-server and gss-client according to the Janet online how-to (https://community.ja.net/groups/moonshot/article/moonshot-pilot-release-1-dvd). We will check if this solution uses proven technologies like GSS and EAP-TTLS to make secure authentication possible.

Test scenario 1:

- Janet moonshot LiveDVD
  - Hostname: debian

Used commands:
gss-server -verbose gss@debian (in pts/0)
gss-client -mech '{1.3.6.1.5.5.15.1.1.18}' debian gss@debian "message" (in pts/1)
freeradius -X (in pts/2, freeradius must be halted if this isn't the case you can execute /etc/init.d/freeradius stop)

In the RADIUS output you will see that there is an EAP-TTLS tunnel where all information is securely transported and handled. In the output that RADIUS produces you can deduce that the local realm is first transported without a username. RADIUS will then check if the realm exists in its configuration and if so, it will set up an EAP-TTLS connection with the client. When the connection is set-up the username will be send from the client to the RADIUS server. There it will authenticate the user.

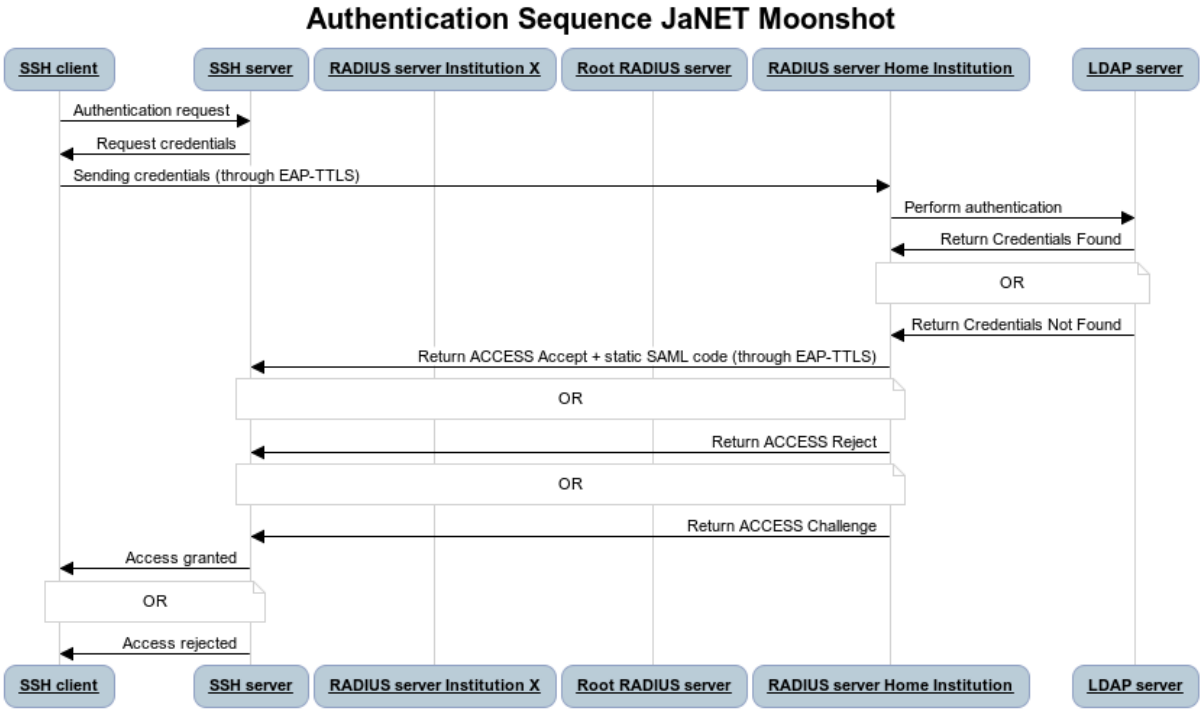


Figure 1 Authentication Sequence Diagram JaNET Moonshot

The RADIUS log shows the information sent in the outer- and inner-tunnel. The realm will be transported through the outer-tunnel to the un-authenticated server. The username will be transported in the inner-tunnel, which the Home Institution RADIUS server will be able to see

when the EAP-TTLS tunnel is set up. Figure 1 shows the authentication sequence of the JaNET Moonshot environment, when trying to access a SSH server.

If the authentication process is completed a local account named *moonshot* will be mapped to the user.

Scenario 2

This chapter describes the scenario of using a multiple system Moonshot environment, instead of using only the JaNET LiveDVD. The RADIUS servers, LDAP server and SSH server are all located on dedicated servers. Figure 2 shows the network topology of the JaNET Moonshot environment in scenario 2.

Scenario 2 contains the following servers:

- JaNET LiveDVD (virtual machine)
- Root RADIUS server
- RADIUS server moonshot.nl
- OpenLDAP server moonshot.nl
- RADtest (program to test radius network connectivity)

Changing the configuration on the JaNET LiveDVD made it possible to successfully authenticate against another RADIUS server in a radius chain. However, at the moment it's not possible to securely authenticate against a RADIUS server in a RADIUS chain. One side note, we did not make use of the GSS integrated in the JaNET pilot. Further research is needed to figure out if a default RADIUS server can be used in a RADIUS chain using GSS technology inside OpenSSH.

When using RADtest, running all software in debug mode makes it possible to see the user information of the user that wants to login. For security and privacy, it is not desirable that all information is viewable in clear text.

A possible solution to address this problem is using GSS; further research will provide the necessary information to determine if RADIUS or OpenSSH need changes in order to accomplish this goal.

The possibility to authenticate works but mapping is not done automatically. Mapping might be possible by running a script that combines users to a unique local account.

Every scenario is captured in a tcpdump/PCap file. They are located on the MoonshotNL Github: https://github.com/MoonshotNL/moonshotdocuments/tree/master/Research/Dump%20v2/dump_2

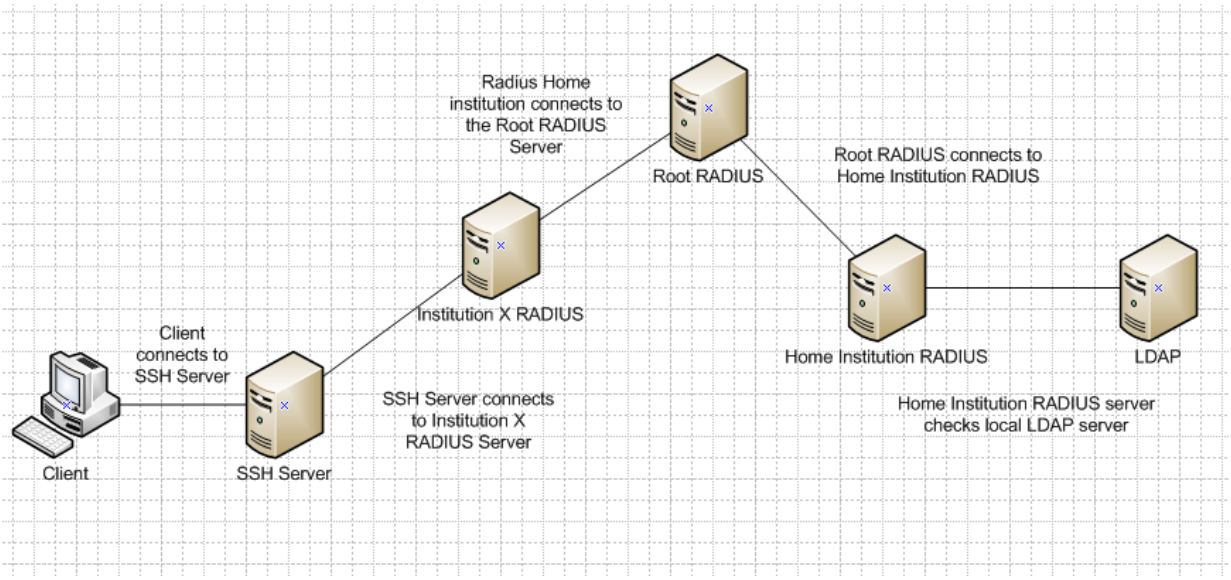


Figure 2 Network Topology

Port JaNET CentOS

The Moonshot environment of JaNET is a Debian (Wheezy) based environment. The standard Linux distribution for Nikhef is CentOS (6). Therefor the team wanted to install the Moonshot libraries and packages onto a CentOS system. JaNET did make some RPMs available, but were not updated for almost two years. The team needed to compile the source code of the Moonshot libraries and packages and built it from scratch.

Initially the team thought that this would not be a big issue, but after a while the team discovered several problems with the code in combination with CentOS 6. Compiling the code took much longer than expected. The team reported major bugs to the developers of JaNET.

Port Results

The source code of the different Moonshot libraries and packages was made available on the Moonshot repository. The source code contains the following libraries and packages:

- Shibboleth XMLtooling
- Shibboleth OpenSAML2
- Shibboleth Service Provider
- Shibboleth Resolver
- Libradsec library
- Jansson
- Moonshot User Interface
- Moonshot library (including the Moonshot-gss-eap mechanism)
- Trust Router
- Moonshot OpenSSH

The source code is available at a git repository, located at: http://www.project-moonshot.org/gitweb/

To be able to compile the code, you need a couple of dependencies. In order to compile code, you need the automake, autoconf, g++ and gcc-c++ package. Those are the basic packages. You also need some less basic packages like *boost-devel, log4cpp-devel, xerces-c-devel, libcurl-devel,* and *libtool*. Using only the CentOS repository is not enough. You need the EPEL repository too. This repository contains a lot of extra packages. Here is an overview of the required packages:

- gcc
- boost-devel
- log4cpp-devel
- xerces-c-devel
- automake
- autoconf
- libtool
- libcurl-devel
- libconfuse-devel
- gettext-devel
- gtk2-devel
- shared-mime-info
- desktop-file-utils

- libgee
- libgee-devel
- dbus-glib
- dbus-glib-devel
- openssl-devel
- sqlite-devel (for the trust-router)
- libtalloc-devel (for freeradius-server)

Some of the required packages needed to be compiled by hand. The libevent pacakage, which you can install using yum is not the right version. You need libevent2. The team used the version libevent-2.0.21-stable.
You also need the xml-security-c package by Apache Santuario. This package is, just like libevent, installable using yum, but the yum repostitory does not contain the right version. The team used the last version available, version xml-security-c-1.7.0.
After preparing the system, you can clone the source code of JaNET using the Moonshot GIT available at: http://www.project-moonshot.org/git/moonshot.git. You need to execute the following commands when you cloned the code onto your system:

cd moonshot
git submodule init
git submodule update

JaNET created a builder script, which automatically compiles the moonshot code in the right order.

Known Issues

Janet created the code for the libraries and packages especially for Debian Wheezy systems. Compiling the code on CentOS could be very problematic. A couple of important dependencies have a newer version on Debian, which are not available on CentOS. Compiling those packages will give a bunch of extra problems. Janet released a couple of new RPMs in early May 2013, but those packages will not work out of the box either.

One of the other package issues was *libevent*. Libevent is available at the yum repository, but not the required version. Moonshot code uses the one of the newest versions of *libevent*, libevent-2.0.21-stable. When you compile this by hand, it will also include the required SSL libraries.

A major issue of the moonshot user interface (on Debian), is that the user interface will only appear once, if you signed in successfully. The team reported this bug to the moonshot-ui developers, and they are working on this problem. Another major bug of the Moonshot UI on CentOS is that it will not run after installing the RPM or after compiling the code. This is caused by a GDBUS problem. This is probably a problem caused by the *glib* package, due to different versions. The team reported this issue to the developers of the Moonshot UI. When the team tried to compile everything on a new system this did not happen, and the Moonshot UI did appear. However, the SSH client did not work as it should. The Moonshot UI will not appear, like it should. The team did not understand why this happened.
https://www.jiscmail.ac.uk/cgi-bin/webadmin?A2=ind1304&L=MOONSHOT-COMMUNITY&P=R7491&1=MOONSHOT-COMMUNITY&9=A&J=on&d=No+Match%3BMatch%3BMatches&z=4
https://www.jiscmail.ac.uk/cgi-bin/webadmin?A2=ind1305&L=MOONSHOT-COMMUNITY&P=R4254&1=MOONSHOT-COMMUNITY&9=A&J=on&d=No+Match%3BMatch%3BMatches&z=4

A major difficulty for our team was the fact that JaNET did not document a lot about their code and packages. It was very difficult for the team to understand the different kinds of code and the way it works. Understanding the code took a very long time.

The SSH server and/or client in combination with the GSS-EAP mechanism does not work on CentOS. When you compile the code from the Moonshot repository, you will not be able to use the Moonshot SSH client out of the box. Somehow the team was not able to get it to work. When trying to access the Moonshot SSH server, this error will appear (in debug mode):

…
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug2: service_accept: ssh-userauth
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug2: key: /home/moonshot/.ssh/id_rsa ((nil))
debug2: key: /home/moonshot/.ssh/id_dsa ((nil))
debug1: Authentications that can continue: publickey,gssapi-keyex,gssapi-with-mic,password
debug1: Next authentication method: gssapi-keyex
debug1: No valid Key exchange context
debug2: we did not send a packet, disable method
debug1: Next authentication method: gssapi-with-mic
debug1: Unspecified GSS failure. Minor code may provide more information
Credentials cache file '/tmp/krb5cc_500' not found

debug1: Unspecified GSS failure. Minor code may provide more information
Credentials cache file '/tmp/krb5cc_500' not found

debug1: Unspecified GSS failure. Minor code may provide more information

debug1: Unspecified GSS failure. Minor code may provide more information
Credentials cache file '/tmp/krb5cc_500' not found

debug2: we did not send a packet, disable method
debug1: Next authentication method: publickey
debug1: Trying private key: /home/moonshot/.ssh/id_rsa
debug1: Trying private key: /home/moonshot/.ssh/id_dsa
debug2: we did not send a packet, disable method
debug1: Next authentication method: password
…

The team reported this issue to the mailing list of JaNET. More developers are struggling with this issue and JaNET is working on a solution.
https://bugs.launchpad.net/moonshot-ui/+bug/1180504

Because the JaNET code is made for Debian, the team tried to compile the code on Ubuntu. Ubuntu and Debian are much alike, but there were also problems. The team reported the issue to the mailing list, and moved on using another operating system, CentOS.
https://www.jiscmail.ac.uk/cgi-bin/webadmin?A2=ind1304&L=MOONSHOT-COMMUNITY&P=R7682&1=MOONSHOT-COMMUNITY&9=A&I=-3&J=on&d=No+Match%3BMatch%3BMatches&z=4

How does it work?

GSSAPI-EAP mechanism

Janet's Moonshot architecture is based on the ABFAB architecture. The Application Bridging for Federated Access Beyond Web architecture (ABFAB) describes an architecture for providing federated access management to applications using the Generic Security Service Application Programming Interface (GSS-API) and Simple Authentication and Security Layers (SASL).[1]

The mechanism name format is described at the EAP-GSS draft written by Hartman & Howlett. GSSAPI permits several types of generic names to be imported using GSS_Import_name(). The ***GSS_C_NT_USER_NAME*** form represents the name of an individual user. The ***GSS_C_NET_HOSTBASED_SERVER*** name form represents a service running on a host, e.g. "host@service". This name form is required by most of the SASL profiles and is used by many existing applications, e.g. Kerberos GSSAPI mechanism.

"RFC3748 discusses security issues surrounding EAP. RFC5247 discusses the security and requirements surrounding key management that leverages the AAA infrastructure. These documents are critical to the security analysis of the mechanism.
RFC2743 discusses generic security considerations for the GSSAPI. RFC4121 discusses security issues surrounding the specific permessage services used in this mechanism."[2]

This mechanism may introduce multiple layers of security negotiation into application protocols. "Multiple layer negotiations are vulnerable to a bid-down attack when a mechanism negotiated at the outer layer is preferred to some but not all mechanisms negotiated at the inner layer. One possible approach to mitigate this attack is to construct security policy such that the preference for all mechanisms negotiated in the inner-layer falls between preferences for two outer-layer mechanisms or falls at one end of the overall ranked preferences including both the inner and outer-layer. Another approach is to only use this mechanism when it has specifically been selected for a given service. The second approach is likely to be common in practice because one common deployment will involve an EAP supplicant interacting with a user to select a given identity. Only when an identity is successfully chosen by the user will this mechanism be attempted."

The security of this mechanism depends on the use and verification of EAP channel binding. Today EAP channel binding is in very limited deployment. If EAP binding is not used, then the system may be vulnerable to phishing attacks where a user is diverted from one service to another. These attacks are possible with EAP today although not typically with common GSS-API is definitely use with trusted-third-party mechanisms such as Kerberos. [2]

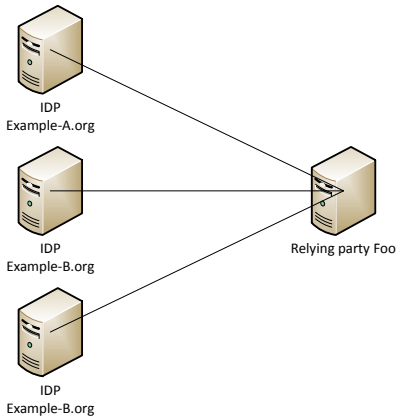
[1] http://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interface
http://en.wikipedia.org/wiki/Simple_Authentication_and_Security_Layer
[2] http://tools.ietf.org/html/draft-howlett-eap-gss-00

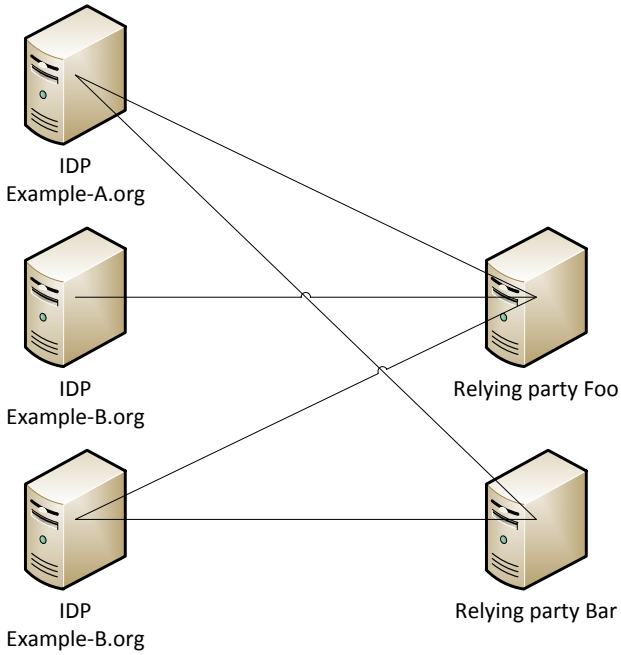
Trust Router

Project Moonshot needs explicit trust relations between organisations, normally these trust relations would be statically configured by hand. As the number of organisations in a federation increase, static trust configuration will probably not scale. As each organisation within the federation will need to configure individual trust relations with each other organisation in that same federation. The work involved in configuring these trust relations will take up way too much time. To counter this problem, the trust router protocol is being developed.
At the moment of writing the Trust Router is in its beta stage. We have not used nor tested the Trust Router. It's function is described in the following excerpt taken from https://launchpad.net/moonshot-tr

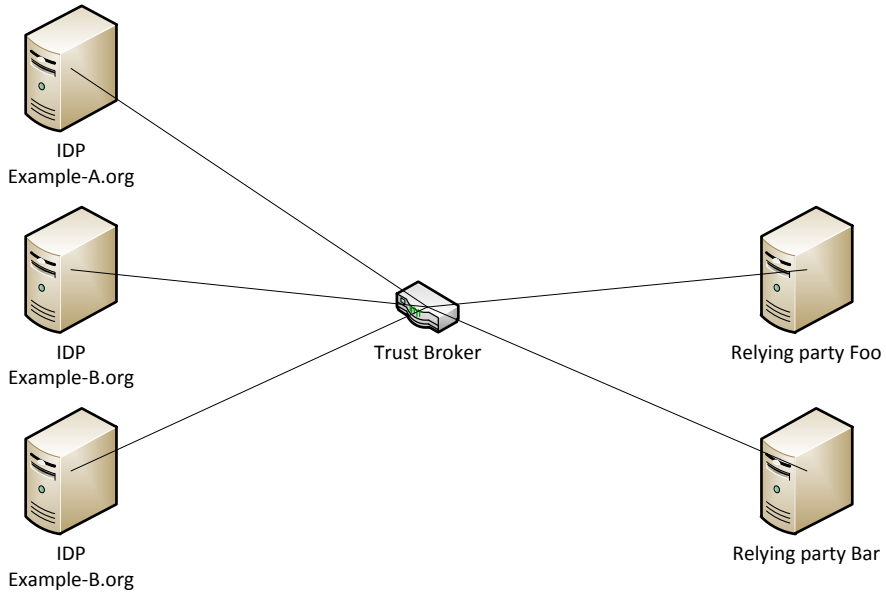
"*A Trust Router is an infrastructural element used to construct multihop Application Bridging for Federated Authentication Beyond the Web (ABFAB) federations. This sub-project of Project Moonshot is focused on the development of the Trust Router infrastructure and Trust Router Protocol. A trust router is a logical ABFAB entity that exchanges information about Trust Paths that Relying Parties can use to create transitive chains of trust across multihop ABFAB federations. The Trust Router Protocol is the mechanism used by two Trust Routers to exchange information about Trust Links and Trust Paths. The Trust Router Protocol, in conjunction with the Temporary Identity Protocol, can be used to enable multihop ABFAB federations without requiring a centralized Public Key Infrastructure (PKI).*"

Considering that this solution was developed with JaNET's mesh network in mind. One might ask the question as to whether or not this would also be needed in say a hub and spoke network like the one SURFnet operates. As opposed to JaNET's mesh network, a single governing trust body could be configured somewhere in the network.
The problem is described like this: Every IDP needs to maintain agreements to exchange credentials with every relying party its users are allowed to access. The likely outcome is shown in the figures. For a small federation this is no problem, but it simply does not scale.





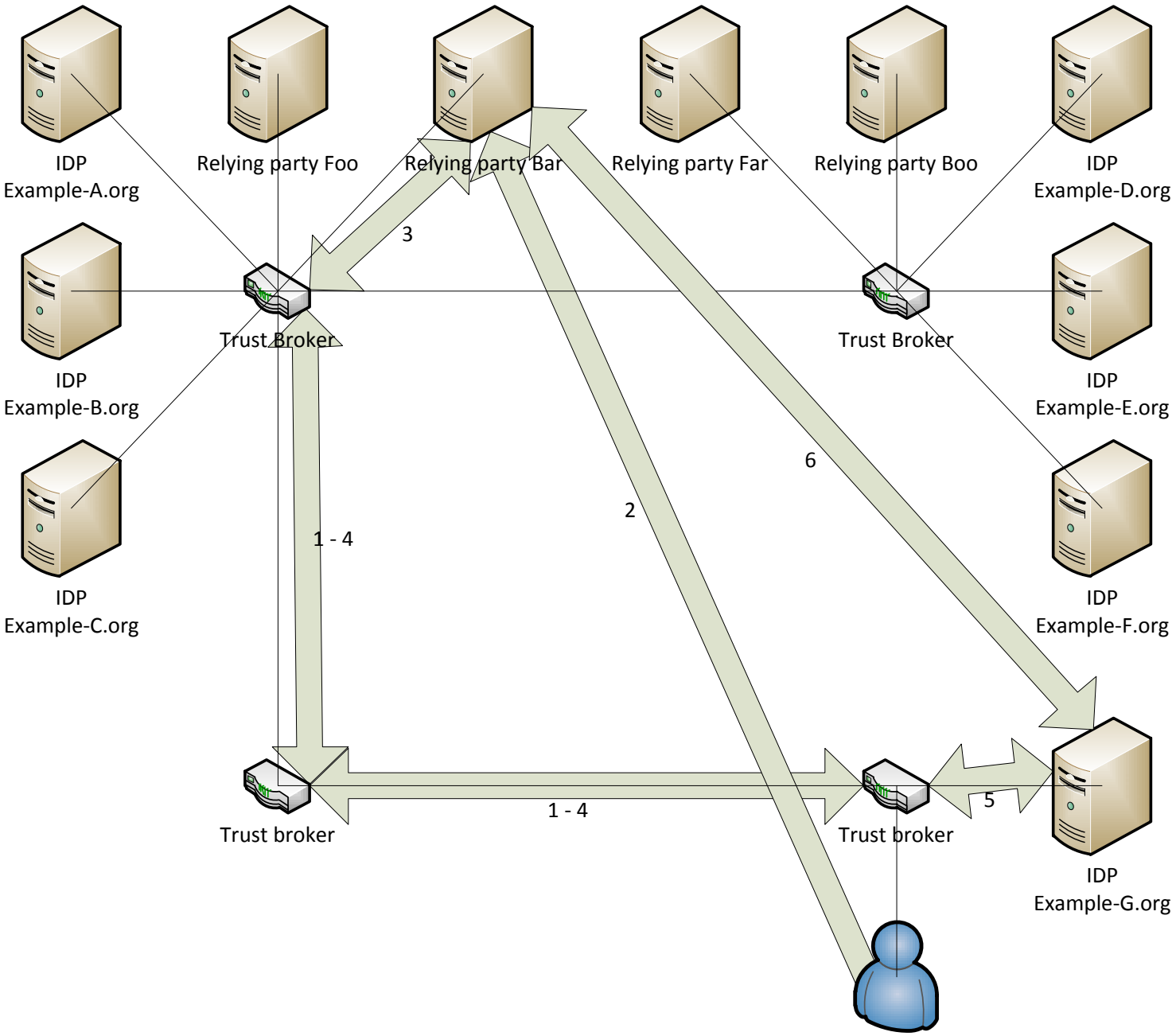
Janet's solution to counter this problem is to introduce a trust broker into the topology. Simplified it would look similar to this:



The solution as described in the IETF draft consists of a set of Trust Routers, running the trust-router protocol and forwarding temporary identity requests between relying parties and IdPs.

The trust Router protocol is a protocol used to exchange information between Trust Routers. The information exchanged will most likely be about available trust links. This information can be used to construct trust paths that can be used to reach for example RADIUS servers. When queried they will return the shortest path to the specified IDP.

En example of this exchange of information can be seen on the next page.

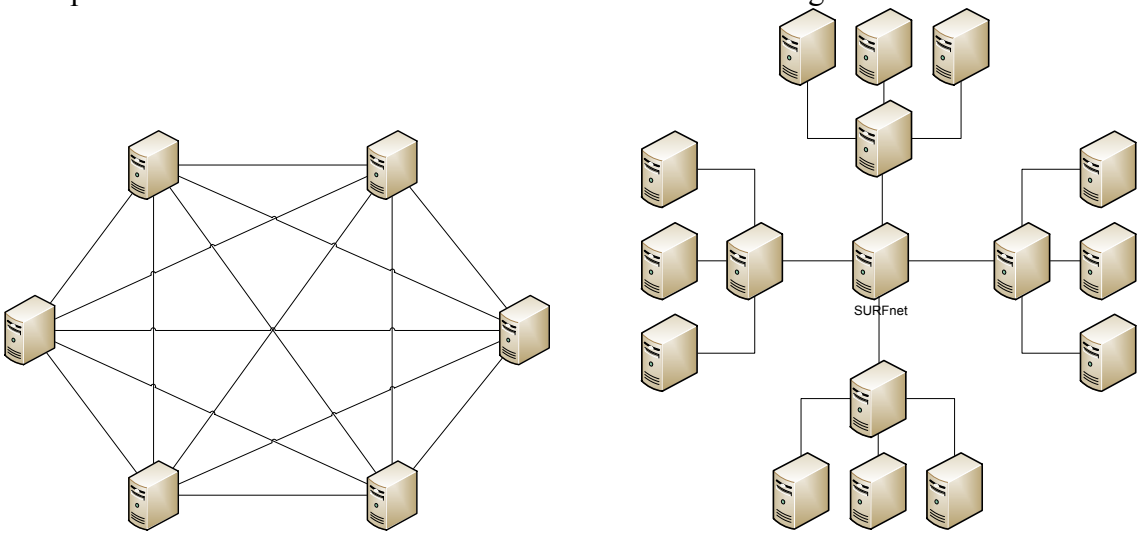


This example is based on the example in the IETF draft abfab-trust-router-02

1. We start with a single federation including four realms, each realm contains a single trust router. The trust routers are peered in such a way that their interconnects form a multi-hop federations.
2. A subject with an example-g.org identity attempts to access a service provided by relying party Bar.
3. The relying Party does not have direct access to the example-g.org server that is can use to authenticate the subject, so it asks its local trust router to forward a Temporary Identity request to example-g's local trust router.
4. The intermediary trust router(s) forward the request along the trust path to the intended trust router.
5. The Example-g.org server receives the temporary from the local trust router and configures a temporary identity for the intended relying party.
6. The relying party can now reach the Example-g.org server to perform the authentication.

Trustrouter in different networks

SURFnet en jaNET have different network topologys, when we look at those networks (simplified) jaNETs network resembles a mesh network. SURFnet has a hub en spoke network, a simplified view of both networks can be seen in these drawings:



The left drawing is a simplified example of jaNETs network. And on the right is a simplified version of SURFnets network. When you think of the trust router and the trust router protocol propagating trust path's trough the network, it would be a good solution for a network like jaNETs. For SURFnet however, this solution might not be ideal. SURFnet has a distinct core in the network, this might be an ideal location to place a single governing 'trust router'. More research can be done in this direction in order to determine the feasibility of such a solution.

Sources and additional information:

Trust router problem statement:
http://tools.ietf.org/html/draft-howlett-abfab-trust-router
http://tools.ietf.org/html/draft-mrw-abfab-trust-router

Application Bridging for Federated Access Beyond Web:
http://tools.ietf.org/html/draft-ietf-abfab-arch

Trust router specifications:
https://community.ja.net/system/files/288/Trust-Router-Func-Spec-3June2012.docx

Appendix I - JaNET Moonshot OpenSSH

Removed files	Changed files	Added files
/moonshot/openssh	**/moonshot/openssh**	**/moonshot/openssh**
gss-serv-krb5.c	auth2.c	changeLog.gssapi
sandbox-darwin.c	auth2-gss.c	kexgssc.c
sandbox-null.c	authfile.c	kexgsss.c
sandbox-rlimit.c	changelog	ssh_prng_cmds.in
sandbox-systrace.c	clientloop.c	ssh-rand-helper.0
ssh-sandbox.h	configure	ssh-rand-helper.8
	configure.ac	ssh-rand-helper.c
	entropy.c	WARNING.RNG
	gss-genr.c	
	gss-serv-krb5.c	
	kex.c	
	kex.h	
	makefile.in	
	monitor.h	
	mux.c	
	PROTOCOL.mux	
	servconf.c	
	servconf.h	
	sftp.1	
	ssh.c	
	ssh_config.0	
	ssh_prng_cmds.in	
	ssh-agent.c	
	sshconnect.c	
	sshconnect2.c	

	sshd.c	
	ssh-gss.h	
	ssh-keygen.0	
	ssh-keygen.1	
	ssh-keygen.c	
	/moonshot/openssh/contrib/cygwin	
	ssh-host-config	
	ssh-user-config	
	/moonshot/openssh/contrib./redhat	
	sshd.init	
	/moonshot/openssh/contrib./suse	
	openssh.spec	
	/moonshot/openssh/openbsd-compat	
	bsd-cygwin_util.c	
	/moonshot/openssh/regress	
	cfgmatch.sh	

Appendix II – Installing and Compiling packages

You need the EPEL repository in order to compile the different JaNET Moonshot packages. If the EPEL repository is installed, the dependencies can be installed.

yum install gtk2 gtk2-devel gcc gcc-c++ libtool boost boost-devel log4cpp-devel log4cpp log4cpp-devel xerces-c xerces-c-devel automake autoconf libconfuse-devel libconfuse gettext gettext-devel shared-mime-info desktop-file-utils libgee libgee-devel dbus-glib dbus-glib-devel openssl openssl-devel libcurl libcurl-devel libtalloc-devel git wget

wget https://github.com/download/libevent/libevent/libevent-2.0.21-stable.tar.gz
tar –xzvf libevent-2.0.21-stable.tar.gz
cd libevent-2.0.21-stable
./configure
make
make install

wget http://psg.mtu.edu/pub/apache/santuario/c-library/xml-security-c-1.7.0.tar.gz
tar –xzvf xml-security-c-1.7.0.tar.gz
cd xml-security-c-1.7.0
./configure
make
make install

You need the git package in order to pull the Moonshot GIT repository:

git clone http://www.project-moonshot.org/git/moonshot.git
cd moonshot
git submodule init
git submodule update

JaNET created a builder script for building the Moonshot code. Some of the packages need to be compiled by hand.

*./builder *
*-r '' *
-c –with-log4cpp=/usr/bin/log4cpp-config-x86_64

After compiling the code the OpenSSH code needs to be compiled. JaNET recommended to use the *debian-5.9p1* branch of the GIT repository for compiling the OpenSSH package:

(cd moonshot)
cd openssh
git checkout –b debian-5.9p1

*./configure --prefix=/usr/local/moonshot *

*LDFLAGS=-Wl,-L/usr/local/moonshot/lib, *
*-rpath=/usr/local/moonshot/lib \CPPFLAGS=-I /usr/local/moonshot/include *
*--with-system-libtool --with-system-libltdl --enable-tls *
*--with-gssapi=/usr/local/moonshot *
--with-xmltooling=/usr/local/moonshot --with-kerberos5
make
make install

Another way of installing the CentOS packages of JaNET Moonshot is to use the YUM repository. Unfortunately, while writing this document, not all of the packages are included in the JaNET repository, e.g. OpenSSH and FreeRADIUS. The YUM repo needs to configured like this:

/etc/repos.d/moonshot.repo
[moonshot]
name=Moonshot RPMs
baseurl=http://repository.project-moonshot.org/rpms/centos6/
#mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-6&arch=$basearch
failovermethod=priority
enabled=1
gpgcheck=1
gpgkey=http://repository.project-moonshot.org/rpms/centos6/moonshot.key

If you update the yum repo (*yum update –y*), it is possible to install the RPMs using yum install. The dependencies of, e.g. *moonshot-ui*, will automatically be installed when the *yum install moonshot-ui* command is issued.

The JaNET CentOS packages are still in development. Eventually there will be a working solution.