User Guide

OpenAKC

For version 1.0.x ⬢ Last Updated on 5 October 2020

OpenAKC © 2019-2020 ⬢ A. James Lewis ⬢ https://github.com/netlore/OpenAKC

Table of Contents

1 Introduction....4
1.1 Purpose....4
1.2 Scope....4
1.3 Versioning....4
1.4 System Components....5
1.5 Potential Application....5
1.5.1 A practical ‘Jump Host’....5
1.5.2 Beyond the ‘Jump Host’....5
1.5.3 Acting as a centralised trust management platform....5
1.5.4 Reducing the risk of ‘private keys’....6
2 Software Installation....6
2.1 Installing via OS repo....6
2.1.1 Ubuntu/Debian....6
2.1.2 Redhat/Centos/Fedora....7
2.1.2.1 Older systems using YUM....7
2.1.2.2 Newer systems using DNF....7
2.1.3 OpenSUSE/SUSE Enterprise....7
2.2 Initial Setup....7
2.2.1 Server Installation....7
2.2.1.1 Combined jump host/security server....7
2.2.1.2 Separated jump host/security server....8
2.2.2 Configuring an OpenAKC administrator....8
2.2.3 Clustering the security server....8
2.2.4 Client Installation....8
3 Configuring OpenAKC....9
3.1 Server Configuration....9
3.1.1 Keys....9
3.1.1.1 Tools keys....9
3.1.1.2 Administrator keys....9
3.1.2 Configuration File....10
3.2 Client Configuration....11
3.3 Tools only installation....12
3.3.1 Separate security server....12
3.3.1.1 Adding “sudoers.d” entry....12
3.3.1.2 Configuration....12
3.3.1.3 Associate with server....12
3.3.2 Remote administration workstation....13
3.3.2.1 Configuration....13
4 Administering Roles and Rules....13
4.1 Getting SSH keys into the system....13
4.1.1 Self Service SSH key submission....13
4.1.2 Manual SSH key submission....14
4.2 Updating or Deleting existing keys....15
4.3 Writing rules....15
4.3.1 RULE....16
4.3.2 DAY....16

4.3.3 TIM....17
4.3.4 SHELL....17
4.3.5 CMD....17
4.3.6 SCP....17
4.3.7 CAP....17
4.3.8 REC....17
4.3.9 FROM....18
5 Post-Install hardening....18
5.1 Passwords....18
5.1.1 Using passwords with SSH....18
5.1.2 Using passwords with legacy servers....18
5.1.3 Using passwords on the console....19
5.2 Access via SSH key pairs....19
6 Advanced Functions....19
6.1 Keystroke logging....19
6.2 Auditing updates to roles & rules....20
6.3 SSH Agent....20
7 Examples....21
7.1 Creating a user for demo....21
7.2 Editing a role definition....22
7.3 Different restrictions applied depending on rules....22
7.4 Session recordings....23
7.5 The power of rules....24
7.6 Non-interactive commands & SCP....24
8 Appendices....26
8.1 Command Reference & Examples....26
8.1.1 Command “openakc-plugin”....26
8.1.2 Command “openakc”....27
8.1.2.1 Users....27
8.1.2.2 Admins....27
8.2 Automated Test Environment....29
8.3 API Reference & Examples....31
8.3.1 Stage 1 Options....31
8.3.2 Stage 2 Options....31

Note: This is document is still a work in progress, it covers the functions of OpenAKC 1.0.x but we would certainly appreciate any feedback and/or corrections you may have. Feel free to contact, or leave an issue on the GitHub page.

At present the API reference is incomplete, but we are happy to raise the priority of this part of the documentation if needed.

Please see the GitHub link on the title page for updated versions of this document.

1 Introduction

1.1 Purpose

OpenAKC is in essence an SSH public key manager for Linux, implemented via a Client/Server API so that all public keys are held centrally. Additionally it adds advanced features such as session recording and powerful controls over how/when the keys are used, including the ability to restrict the root user after login.

Functionality including, but not limited to:-

* Centrally manage “static trust”, entirely replacing (and extending) the “authorized_keys” functionality.
* Allow users to register personal SSH key pairs, and then be granted access based on directory information such as AD, LDAP or even local Unix accounts, without requiring directory access on each host in the environment (Eg. using bastion/jump hosts).
* Do session recording without requiring a “proxy” or “man-in-the-middle” to access the session data.
* Extend the traditional restrictions which are applied to keys, permitting time/day/date based restrictions etc.
* Manipulate Linux capabilities to restrict functions available to the “root” even after they log in, or to restrict functions gained by using “sudo”.
* All security configuration performed in a central security server (or cluster of servers) via an API, no access to the managed hosts is required by OpenAKC admins after the client is installed.

1.2 Scope

This document describes the operation of OpenAKC including installation, configuration, operation and troubleshooting. It also touches on some details of the low level operation of the system, as well as recommended SSH configuration to make the most of OpenAKC features.

1.3 Versioning

The versioning of OpenAKC follows a number scheme, of *major* dot *minor* dot *patch level*, where odd numbered *minor* versions are development releases, and the *major* version is updated whenever a change is made which may result in existing configuration being broken or needing to be updated.

For example, 1.0.15, or 1.2.11 is a stable release, while 1.1.15 or 1.3.3 is a development release.

1.4 System Components

The system consists of 4 packages, containing the client, server and API tools components.

“openakc-shared” – this contains shared functions used across more than one component.

“openakc-server” – this is the back end security server which implements the API.

“openakc-tools” – this is the API tool, used by everyone to interface with the system.

“openakc” – this is the client package installed on the systems to be managed by OpenAKC.

1.5 Potential Application

Who is the target audience for OpenAKC, and what circumstances can it provide benefits?

1.5.1 A practical ‘Jump Host’

Having worked in many Linux environments, the author of the software has seen a number of ‘jump host’ solutions using dubious mechanisms such as “sudo” and others used to map users from AD or LDAP directories to SSH trust relationships using shared private keys etc. and these solutions are almost universally highly insecure. OpenAKC can implement a new solution, or act as an almost “drop in” upgrade for a legacy solution by simply migrating users to personal rather than shared keys with a “self service” key management mechanism. Requiring personal keys have pass phrases ensures increased security, and avoids ad-hoc automation from user accounts. The system provides rich control and monitoring features, while users can use familiar tools with little to no disruption to their workflow.

1.5.2 Beyond the ‘Jump Host’

Many businesses force users via a central choke point, simply to ensure that session recording is in place to meet various support and security requirements. OpenAKC performs the authorization process, and session recording/restrictions via an out of band connection from the target machine, to the security server…. This means that (depending on precise requirements), it may well be practical for users with ‘Putty’, ‘Xshell’ or whatever terminal they prefer, to access hosts directly from their workstations, without compromising the ability to apply restrictions, and session recording as required. You could also use OpenAKC to control access to the file transfer functionality in SSH, thereby limiting the potential for data loss.

1.5.3 Acting as a centralised trust management platform.

By allowing the “authorized_keys” mechanism on the hosts to be completely disabled, OpenAKC permits SSH trust across an entire estate to be managed (with rich control and monitoring features) centrally by “systems administration” or “information security” staff. This means that users, or application developers etc. cannot add or remove trust relationships, effectively enforcing any whitelist or approval process you might want to establish for the creation of trust relationships within an estate.

1.5.4 Reducing the risk of ‘private keys’

SSH private keys are everywhere in a modern Linux platform, and many solutions attempt to provide security by controlling access to private keys, the problem is that these solutions invariably have elaborate ways to gain access to private keys or completely alternate access methods and porting all your solutions to use those is an enormous task (if it’s even possible)! OpenAKC controls access to the public keys, and can apply many restrictions so that you can make it difficult or impossible for someone with access to a given private key to use it inappropriately.

Because OpenAKC manages public keys and not private keys, it may well prove to be compatible with, and able to work in parallel with other tools which manage access to private keys.

2 Software Installation

Depending on your environment, you may wish to install/update directly from the main repo, or download the packages for use either in a stand-alone environment, or in a private or corporate repository so that new versions can be tested.

2.1 Installing via OS repo.

The use of “`sudo`”, or the root account is assumed where necessary.

2.1.1 Ubuntu/Debian

Download and add the repository key:

```
wget -O - https://raw.githubusercontent.com/netlore/OpenAKC/master/resources/openakc.key | apt-key add -
```

Add the repository:

```
echo "deb https://netlore.github.io/OpenAKC/repos/ubuntu/20.04 ./" > /etc/apt/sources.list.d/openakc.list
```

Update packages:

```
apt update
```

Install packages, depending on the requirements, using “`apt install <packagename>`”

2.1.2 Redhat/Centos/Fedora

2.1.2.1 Older systems using YUM

Add the repository:-

```
wget -O - https://netlore.github.io/OpenAKC/repos/openakc-el7.repo > /etc/yum.repos.d/openakc.repo
```

Install packages, depending on the requirements, using "`yum install <packagename>`"

2.1.2.2 Newer systems using DNF

Add the repository (depending on OS, you may first have to install the config-manager):-

```
dnf install 'dnf-command(config-manager)'
dnf config-manager --add-repo https://netlore.github.io/OpenAKC/repos/openakc-el8.repo
```

Install packages, depending on the requirements, using "`dnf install <packagename>`"

2.1.3 OpenSUSE/SUSE Enterprise

A SUSE is currently planned, but not yet available, you can use the existing packages, but you must ensure that the package "libcap-progs" is installed, and you may have to ignore a dependency on "openssh-clients".

Packages tested with OpenSUSE 15+

```
wget -O - https://netlore.github.io/OpenAKC/repos/openakc-opensuse15.repo > /etc/zypp/repos.d/openakc.repo
```

2.2 Initial Setup

2.2.1 Server Installation

2.2.1.1 Combined jump host/security server

The package "openakc-server" provides the server itself, which handles authentication requests, registration requests from the client package, and permissions updates etc.

The package "openakc-tools" will also be installed, and contains the interface tool, which allows users to register their personal keys, and also allows administrators to perform administration tasks such as upload, replace, or delete static keys and configure rules etc.

The "openakc-tools" package can also be installed stand-alone on a separate system either as a tool for administration, or to facilitate a configuration where users register keys from different hosts where "openakc-server" is not installed.

Running on the same host as the “openakc-server” package, to verify that the server is working after initial installation, it is possible to type "openakc ping" after installing the server packages. This should force the server to write out a default configuration and reply with "OK: Pong!".

2.2.1.2 Separated jump host/security server

In the case where you wish to have a separate security server, then the “openakc-tools” package can be installed on a 3rd system. If normal users need to register personal keys from that 3rd system, then the key generated in “/var/lib/openakc/keys” by the “openakc-tools” package can be copied to the same folder on the security server. The only restriction is that the two systems must use the same directory, such that users and group memberships are identical on both.

If you only wish to manage trust relationships and not dynamic users then OpenAKC does not need any system which users have access to. Administration of trust relationships can be performed locally at the security or from another machine with the “openakc-tools”, even if that machine does not have directory access (or “sudo” rules, as discussed in the “Tools keys” section).

2.2.2 Configuring an OpenAKC administrator

After running the “openakc” tool, for the first time, a folder will be created in the users home folder, called “~/.openakc”, and an openssl key pair will be created. If the public key from this folder is copied to the “/var/lib/openakc/keys” folder on the server, then this will permit the user to perform security administration tasks.

If the “openakc-tools” package is installed on a machine other than the server, then a configuration file will be required to define the APIS variable, and this can be located centrally in “/etc/openakc/openakc.conf”, or in the users home folder, eg “~/.openakc/config”

2.2.3 Clustering the security server

Clustering multiple security servers can be achieved using some form of shared or replicated storage, perhaps a NAS solution using NFS, file replication (such as GlusterFS, details found here - https://docs.gluster.org/en/latest/Quick-Start-Guide/Quickstart/) or some form of clustered storage with a shared back end such as a SAN. Simply replicating the “/var/lib/openakc” folder across all the nodes in the cluster will allow any number of nodes to work together.

File replication is used instead of a database, as it is more flexible, and there are multiple asynchronous bi-directional file replication solutions (many of which are free software). There may be an option to use a database in future, but it involves extra complexities.

2.2.4 Client Installation

To set up your clients, install the package “openakc” on each host, and ensure that the client config (/etc/openakc/openakc.conf) lists the appropriate OpenAKC API server(s) in the APIS field (this can be either name or IP). The configuration is pre-populated with "openakc01 & openakc02", so you could simply ensure that these values are configured in DNS, (or in the /etc/hosts on the client for testing), or alter them as you prefer.

Once the client is installed, and configured, it should be possible to run “openakc-plugin ping”, to get an “OK: Pong!” response from the server, confirming successful communication. The plug-in has only very minimal command line functions, but if for any reason you rebuild the server and lose its openssl identity or a server which clients have already contacted has its identity changed (perhaps during setting up of clustering), you can remove the server identity from a client using “openakc-plugin resetkeys”. The client will then re-establish communication with the server.

3 Configuring OpenAKC

3.1 Server Configuration

3.1.1 Keys

Access to the server for both the “self service” process allowing users register their own ssh key pairs, and for administrators to add role configurations is via openssl key pairs. This may not be immediately obvious when it comes to the former, especially if you are installing a “combined” configuration where the security server and the jump host are one and the same because the tools package and the server package install keys in the same folder (/var/lib/openakc/keys) by design.

3.1.1.1 Tools keys

You may notice that the “openakc” tool, uses “sudo”, both to validate the users claim to a certain identity, and to gain access to the openssl key which gives it rights to register the users SSH key pair with the server. Users should not have direct access to this key.

If you want to create an installation where the users are on a different host to the security server, you must install the “openakc-tools package and copy the public key created by the tools package (in /var/lib/openakc/keys) folder into the same folder on the security server. By doing this, you are essentially declaring that these two hosts are using the same directory to obtain information about users. You can have more than one host where users are able to register keys, simply by copying the key from both.

3.1.1.2 Administrator keys

Once a user has registered their SSH key pair, with the system, a folder will be created in their homedir “~/.openakc”. This will contain their personal openssl key pair, as well as the openssl public key of each OpenAKC API server they have communicated with directly.

To signal that this account will be an OpenAKC administrator, simply take the user's public key (pubkey.pem) file, and place it into the /var/lib/openakc/keys folder on each server they will communicate with (Also see the section regarding replication if you will be creating a server cluster).

Once the user's public key is in place, they will be able to use many extra functions via the API, options such as "editrole", "getrole", "setrole", "submit", etc. These are detailed later in this document.

3.1.2 Configuration File

The server configuration file is not deployed by the installation package, but if it is not created manually before the first time the server is called, the server will write a default configuration, which will be perfectly adequate under almost all circumstances.

An example server configuration should look something like the following:-

```
APIS="localhost"
PORT="889"
CACHETIME="60"
DEBUG="no"
PREEXEC="/bin/true"
POSTEXEC="/bin/true"
```

APIS is a comma separated list of hostnames (or IP's) which will be used by any openakc tools on this system to communicate with the server, it is not used by the server itself.

PORT is the TCP port which will be used by any openakc tools on this system to communicate with the server, it is not used by the server itself as the server port is defined by the systemd unit configuration files.

CACHETIME is currently not used as of 1.0.0, it is for future use.

DEBUG determined whether high detail debugging output is generated in the logs.

PREEXEC points to a script which will be executed immediately *before* an authentication cycle takes place. The script will be called with the parameters in the format:-

```
$PREEXEC "role being called" "IP of target host" "fingerprint of calling user" "username of calling user"
```

Returning a non-zero exit code from the PREEXEC script, will cause the authentication to be rejected.

POSTEXEC points to a script which will be executed immediately *after* an authentication cycle takes place. The script will be called with the parameters in the format:-

```
$POSTEXEC "role being called" "IP of target host" "fingerprint of calling user" "username of calling user"
"[ACCEPT|DENY]" "Session ID"
```

Exit codes from this script are ignored. It is envisioned that it might be used, for example to force a flush of cached user data in a directory which may be out of sync with AD. The possibilities are endless.

3.2 Client Configuration

A default client configuration file is deployed by the installation package (openakc) to "/etc/openakc/openakc.conf", and will be perfectly adequate under most circumstances, but there are parameters which can be tuned. WARNING: the client installation marks configuration files as "immutable" if possible, so you may need to remove the immutable flag before editing these files.

An example server configuration should look something like the following:-

```
APIS="openakc01,openakc02"
ENABLED="yes"
PORT="889"
CACHETIME="60"
DEBUG="no"
PERMITROOT="yes"
AUDIT="yes"
QUIZ="yes"
HIDE="restrict"
FAKESUDO="yes"
```

APIS is a comma separated list of hostnames (or IP's) which will be used by any OpenAKC tools on the system (primarily the OpenAKC client) to communicate with the server. The APIS parameter is pre-populated in the deployment package to include the names "openakc01 & openakc02", this is intended to permit a "zero configuration" installation, since you could simply populate these names in your DNS to allow immediate authentication once the client is installed.

PORT is the TCP port which will be used by any openakc tools on this system (primarily the OpenAKC client) to communicate with the server.

CACHETIME is currently not used as of 1.0.0, it is for future use.

DEBUG determined whether high detail debugging output is generated in the logs.

PERMITROOT determines if OpenAKC is allowed to grant root access to this host.

AUDIT determines if OpenAKC is allowed to send the host configuration such as memory size, CPU configuration, Network configuration etc. back to the server for environment audit data.

QUIZ determines whether the OpenAKC session will ask the user to provide a summary of the reason for their connection before running their shell/command. The Quiz will only be presented if the command is user interactive.

HIDE determines how much OpenAKC will hide itself from the user. Normally there will be an "OpenAKC session banner", and any capability restrictions will be declared. If HIDE is set to "no", full format banner and capability restrictions will be displayed. If it is set to "restrict" then capability restrictions will be displayed. Note that if the QUIZ is enabled, then a minimal banner will be displayed even if HIDE is set to "yes".

FAKESUDO determines whether the client will generate "sudo" messages intended to feed information about privilege escalation to SIEM or other systems which cannot read OpenAKC logs.

3.3 Tools only installation

There are at least 2 reasons one might wish to install the “openakc-tools” package separate from the server, and to do so, you will need to perform some manual configuration. No configuration is deployed by the package, because depending on how you plan to use it, the required configuration is different.

3.3.1 Separate security server

In the case where the users are not able to log in to the security server (but still need access to the self service SSH Key registration process), as shown in the “Separate security server” diagram, you will need to install the “openakc-tools” package, and apply some manual configuration. Note that in this case, the tools package MUST be installed on a system using the same source for user information (AD, LDAP etc.) as the server.

3.3.1.1 Adding “sudoers.d” entry

In order to allow users to run the self service key registration process, simply copy the file “/etc/sudoers.d/openakc” to the machine with the tools package. Please take care with permissions for this file! Also note that you may remove this file from the server if no users are required to access self service key registration from there.

3.3.1.2 Configuration

Additionally in order to allow users to run the self service key registration process, the tools package will require a configuration file defining where the server or servers are located. Create the folder “/etc/openakc/”, and a file “/etc/openakc/openakc.conf” containing at least the “APIS=” line as shown above in the Client Configuration section above.

3.3.1.3 Associate with server.

So that the security server knows to accept key registration requests from this new system, you will need to copy the file “/var/lib/openakc/keys/openakc-system-client-XXXXXXXX-pubkey.pem” to the same location on the OpenAKC server(s), again taking care with file permissions. If this file is not present, attempt to register a key using “openakc register”, and a new file will be created.

3.3.2 Remote administration workstation.

It is also possible to have a user, or users on a separate workstation (which does not have to be part of the same directory as the server). Note that there is no “Associate with server” section here, since we are granting permission to a specific user, not an entire host.

3.3.2.1 Configuration

As in the configuration section above, the APIS parameter must be defined in the configuration, which can be “/etc/openakc/openakc.conf”, however if just one user requires access, the configuration can also be defined in the user’s “~/.openakc” folder. If this folder does not exist, you can create it manually, or run “openakc ping”, the ping will fail, but will create the folder. After defining the APIS field as described in the sections above, (in either location, IE add the line ‘APIS=”hostname01,hostname01”’ to either ~/.openakc/config or to /etc/openakc/openakc.conf) run “openakc ping” again, and you should see a response from the server. Now you will see some extra files in “~/.openakc”, and you can follow the instructions under the “Tools Keys” section of “Server Configuration” above, to copy the file “openakc-user-client-[username]-pubkey.pem” to the server, to grant admin permissions for this user.

4 Administering Roles and Rules

Firstly as a reminder, in the following sections, the term “role” refers to a set of “rules” which apply to a given remote account. For example, a “role” would be “root@myhost”. There is currently a single “special” hostname which will be discussed later. The term “rule”, refers to a set of parameters which determine what users or systems are permitted to access a “role”, at any given time, and with what (if any) restrictions.

4.1 Getting SSH keys into the system.

OpenAKC works by determining the identity of the user or system owning a given SSH key, so that it can determine whether that key should grant access at a given moment and what (if any) restrictions should be applied to that session. As a result, keys must be entered into the system before they can be granted access anywhere.

4.1.1 Self Service SSH key submission

In order to be granted access anywhere by the system, all users (including administrators) must first create an SSH key pair (which must include a pass phrase). The public key must then be “registered” with the system, so that it can be associated with their user account. The key configuration of the systems must be such that the security server and wherever the users can register keys from are using the same source of user information, be that AD, LDAP, NIS+ or some other system. Once a valid key pair has been created by the user (using SSH’s ssh-keygen command), the user simply calls the OpenAKC tool thus – ‘`openakc register`’ as shown.

```
demouser@openakc-combined:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/demouser/.ssh/id_rsa):
Created directory '/home/demouser/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/demouser/.ssh/id_rsa
Your public key has been saved in /home/demouser/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:rPRfmrXQTmOoqNxJkLOAhsYnNxcUnAlxrPqMWIwkFIY demouser@openakc-combined
The key's randomart image is:
+---[RSA 3072]----+
|.o.o=++          |
|E.  o=           |
|.   ..           |
|+o . ...         |
|=Bo++.. S        |
|+.*.o= o   o     |
|..+ . o . o B    |
|.. + o o o X o   |
|    o.+ . + o    |
+----[SHA256]-----+
demouser@openakc-combined:~$ openakc register
OpenAKC Copyright (C) 2019-2020  A. James Lewis.  Version is 1.0.0~alpha15~ubunt
u20.04-1.

This program comes with ABSOLUTELY NO WARRANTY; see "license" option.
This is free software, and you are welcome to redistribute it
under certain conditions; See LICENSE file for further details.

Passphrase is requested to ensure you own this key.

Enter passphrase:

Escalating to perform API call

Connected to OpenAKC server.  Sending key registration request
OK: Request processed

demouser@openakc-combined:~$
```

4.1.2 Manual SSH key submission

An "administrator" (as discussed in earlier in the "Administrator keys" section), is able to execute extra commands, including those to view, download, upload or edit a file, containing the "rules" associated with a given "role". Additionally however, they can submit a public key to the system which does not belong to them, this can then be referred to by "rules" they might create.

This is done by using the OpenAKC tool, thus – '`openakc submit filename.pub "Comment Field"`', again, as shown.

```
admin-user@openakc-combined:~$ openakc submit /tmp/3rdpart.pub "3rd party key"
OpenAKC Copyright (C) 2019-2020  A. James Lewis.  Version is 1.0.0~alpha15~ubunt
u20.04-1.

This program comes with ABSOLUTELY NO WARRANTY; see "license" option.
This is free software, and you are welcome to redistribute it
under certain conditions; See LICENSE file for further details.

Storing public key /tmp/3rdpart.pub with comment "3rd party key"
Connected to OpenAKC server.  Sending key submission request
OK: Stored public key - S/N = key-MjncPTcGc3

admin-user@openakc-combined:~$
```

As you can see, a serial number is returned when the key is stored, you should make a note of this, as it will be used to refer to the key when creating rules.

4.2 Updating or Deleting existing keys

Once you have submitted a key to the system it can be updated using it's serial number… this will change the key associated with the serial number, allowing you to rotate keys without updating any assocated rules. Keys can also be removed from the system if they are no-longer required.

To update a key, use:-
```
openakc update key-serial filename "Comment Text"
```

To remove a key from the server, use:-
```
openakc update {key-serial|username} DELETE
```

Any rules referring to these keys will be invalidated but not removed.

4.3 Writing rules

Each rule consists of a multi-line block, starting with "RULE=". The RULE line defines a period where the "rule" is active and who should be granted access, be that a "user" or "group" derived from the OS (via whatever directory you have configured in the OS), or a specific key as defined by the serial number. Other lines in the "rule block" determine other parameters such as "day of week", "time of day", "source IP", etc. as well as more advanced restrictions.

The rules associated with a specific role can be managed with the following commands:-

To view, use command:-
```
openakc showrole user@hostname
```

To download, use command:-
```
openakc getrole user@hostname filename
```

To upload, use command:-
```
openakc setrole user@hostname filename
```

To edit using either 'vi' or whichever editor is defined in the EDITOR variable, use command:-
```
openakc editrole user@hostname
```

Multiple rule blocks can be included for a single role, and an attempt will be made to merge similar rules, but this can be slightly unintuitive, since not all the parameters can be evaluated before the "accept/deny" decision is made. It is suggested for clarity that at least a blank line be included between the lines to separate the blocks, but you may also wish to add comments.

A simple rule of thumb is that as long as the "RULE", "DAY" and "TIME" parameters would result in the rule being accepted, and only one of the other parameters is changed, then the rules can be merged. If more than one other parameter is changed, then the rules cannot be merged, and the rule nearer the top of the role file will take precedence.

An example role file with 2 rules is shown below, as you can see the rule file will come pre-populated with comments detailing the format of each rule. An example rule will also be included in the comments, but it has been removed here for clarity.

```
# Edit this file to control permissions managed by OpenAKC.
# Rules for root@openakc-client.
#
# First matched rule will apply.
# Each rule begins with a RULE line "RULE=date,date,type,tag"
# "type" can be user, group or key(ssh key fingerprint)
# "tag" will depend on the type, either username, groupname, or key serial no.
# DAY= can be either "any" or a comma separated list of 3 letter day codes.
# TIM= is daily {Start Time},{End Time} 24 hour format, local server time.
# SHELL= users shell (you could make shell "rjoe/rvim file", or /bin/false).
# CMD= is a list of permitted commands, "any" or " ; separated list"
#   NB: 1) if CMD is not "any", it must include scp to allow SCP.
#       2) if CMD is not "any", sftp-server requires full path.
#       3) if CMD is exactly/only "internal-sftp", OpenAKC session will be
#          bypassed and "internal-sftp" will be forced.  Care! No capability
#          restrictions can be applied in this mode, and no session
#          recording will take place other than sftp's own logging.
#       4) Internal SFTP can only be chrooted in sshd_config, but capabilities
#          can be applied to external sftp Eg (/usr/lib/openssh/sftp-server).
#       5) if a users calls an absolute path to a binary then CMD must
#          match the absolute path, otherwise short names match path (Care!).
#   NB: Multiple CMD= lines are permitted and will be aggrigated.
#   EG: CMD=scp;ls;uname;dmidecode;/usr/lib/openssh/sftp-server
# SCP= "sed" expression applied to scp, to allow chroot or redirecting files.
# CAP= comma separated list of capabilities denied to role (man capabilities).
# REC= record session log, yes/no
# FROM= can be either "any" or a comma separated list of IPs or CIDR subnets
#
# Warning, TAGs must be CAPITAL, do not use quotes.
# Rules nearer the top take priority, but some attampt is made to merge fully
# compatible rules (See User Guide for details.)
#
RULE=2020/01/13 19:17,2020/01/13 20:17,user,jlewis
DAY=any
TIM=any
SHELL=/bin/bash
CMD=any
SCP=s,^/,/data/,g
CAP=cap_linux_immutable
REC=yes
FROM=any

RULE=2020/01/13 19:17,2020/01/13 20:17,key,key-MjncPTcGc3
DAY=Sat,Sun
TIM=any
SHELL=/bin/bash
CMD=any
SCP=s,^/,/data/,g
CAP=cap_linux_immutable
REC=yes
FROM=10.30.21.0/24
~
```

Rule entries are comma (,) separated (where appropriate), with the exception of CMD and SCP, which are semicolon (;) separated.

4.3.1 RULE

The RULE parameter consists of 4 fields, as follows:-

a) Start date/time field in the format “YYYY/MM/DD HH:MM”.

b) End date/time field in the format “YYYY/MM/DD HH:MM”.

c) Rule type can be one of “**user**, **group** or **key**”.

d) Value to match for the particular rule type, this may be an OS user, group or an OpenAKC key serial number.

4.3.2 DAY

Day is a comma separated list of 3 letter day codes, for example “DAY=Sat,Sun”. It is not case sensitive.

4.3.3 TIM

The TIM or TIME parameter consists of 2 fields, as follows:-

a) Start time only, in 24 hour format.

b) End time only, in 24 hour format.

4.3.4 SHELL

SHELL, indicates what shell to run when the user logs in. It would typically be "SHELL=/bin/bash", but could be used to run a restricted shell, some type of menu for more restricted users, or perhaps blocked entirely by setting "SHELL=/bin/false".

4.3.5 CMD

CMD (Command) is a semicolon (;) separated list of executables, if it is not "any" then the commands which can be executed remotely will be limited to those executables. It must include "scp" if files are to be transferred using that protocol. If commands are entered with their full path, then the request must include the full path, otherwise the system path will be honoured. Note that sftp can be fully supported only with the external sftp subsystem, and the path the sftp server must be included in the list to permit sftp to function.

Note that if the "internal-sftp" server is required, it will be the only command that can be executed, and no other restrictions will apply, and CMD must be set to exactly "CMD= internal-sftp" only.

Note that for clarity, there can be multiple CMD= parameters, and this will be interpreted no differently than if the values were in a semicolon (;) separated list.

4.3.6 SCP

SCP is a semicolon (;) separated list of "sed" commands which will be executed on any scp path (either put or get), called. The example shows a simple "chroot", where "/data" is added to the start of any paths called with scp.

Eg:- "`SCP=s,^/,/data,g`"

4.3.7 CAP

CAP is a comma separated list of Linux capabilities which should be withdrawn from logins using this rule (https://man7.org/linux/man-pages/man7/capabilities.7.html).

It will continue to operate regardless of "sudo" "su -" or setuid binaries executed after logging in with a restriction.

4.3.8 REC

REC is a simple "yes/no" parameter, it will determine if session recording is enabled for sessions opened with this rule. Note that file transfer and external commands executed will still be recorded, but no output or interactive session log will be recorded if this parameter is set to "no".

4.3.9 FROM

From is a comma separated list of either IP, or CIDR subnets, from where the connection will be permitted. Note that at the moment the rule is evaluated, the source of the connection is not known, so this parameter is passed directly to the remote ssh daemon in the restrictions applied to the public key. Because of this, it may appear that OpenAKC has permitted a connection but the remote ssh daemon may still block it and log a failure due to invalid source IP.

5 Post-Install hardening

Once you have an established OpenAKC system, you will want to remove alternate options for accessing OpenAKC clients.

5.1 Passwords

You will need to consider any way in which users can access the host using a password, and ensure you have understood the implications and taken the appropriate action, since OpenAKC is *only* controlling access via ssh key pairs.

5.1.1 Using passwords with SSH

The SSH daemon configuration should be such that passwords cannot be used. Eg:-

```
PasswordAuthentication no
ChallengeResponseAuthentication no
```

If there is a specific account which needs access via a password rather than using SSH keys which can be managed by OpenAKC (For example a dedicated SFTP server), then you should use the “Match User” directive. Eg:-

```
Match User sftpuser
  PasswordAuthentication yes
  ForceCommand internal-sftp
  ChrootDirectory /srv/sftproot
```

Obviously, it would be expected that no other directories would be configured, so one would assume you would also have other password authentication methods disabled. Eg:-

```
KerberosAuthentication no
GSSAPIAuthentication no
```

5.1.2 Using passwords with legacy servers

Services such as “telnet” and “rsh” should long ago have been disabled, as they are unencrypted, and have no place in a modern data-centre. Even thou “ftp” should probably also fall in that category, there may still be some cases where it is still required, so you should ensure that it only provides access to a “chrooted” server environment (not have access to the system folders), for a limited set of users and take all necessary precautions (not detailed here).

5.1.3 Using passwords on the console

Obviously, access to the system console via a password may be required in the event of a failure and since OpenAKC does not have administrative access to the client machines by design, it does not manage password rotation. As a result you will need to decide what the risk of physical access (or remote console access) in your environment is, and manage it appropriately.

It would probably be a good idea to disable passwords for most users, and OpenAKC could be used to permit access for a script to rotate root passwords, which should only be used for emergency access via the console.

In the future, OpenAKC may come with some template tools to assist with this, but it will not be a core component of OpenAKC, as it would defeat the design criteria of a client without administrative access.

5.2 Access via SSH key pairs.

OpenAKC does not require “authorized_keys” files to work, so these should certainly be disabled, once any pre-existing required access is migrated to OpenAKC. The proposed client configuration would include the following:-

```
AuthorizedKeysFile /dev/null
```

6 Advanced Functions

6.1 Keystroke logging

Keystroke logging can be a powerful tool for security, and for understanding what went wrong when an issue occurs.

Each session opened via OpenAKC will use the same API to authenticate with the security server and to log some details of the connection. Assuming session recording is not disabled, it will send a record of what was displayed on the terminal, including the users actions.

This session recording is very granular (you can break down recordings by host, or by user… even tie a recording to a specific session, or PTY), and can record sessions that do not come via a specific bastion/jump host, as they are recorded at the destination and streamed back to the OpenAKC server. However, be aware that the recordings are not 100% real time, as there is some local buffering for encryption and network reasons, otherwise the users session performance would be impacted. We therefore recommend that in high security environments, or where there may be a legal requirement to retain session recordings, that you always use “defence in depth”, consider channelling all ssh connections via a bastion/jump host, or other choke point where session recording can also take place. The tools will then complement each other.

If you are using RedHat, for example, you could use SSSD to provide access to AD/LDAP on the bastion/jump host and security server (which OpenAKC will need), and then use tlog to record users sessions on the baston/jump host [1]. These will be much less granular, ultimately more difficult to use, since a single session on the jump host may encompass many connections to different machines and indeed may not always record what you need if there is trust between hosts inside the environment, but it is possible you may catch something which OpenAKC does not. For example, if a user does something which causes an immediate kernel panic, then some part of the session log may never be flushed from the buffers. Equally, some failure on the bastion or jump host (or a user who is able to access without going via the choke point) might cause it to miss events, but in a "separated" configuration where OpenAKC is running on dedicated servers, it will continue recording.

[1] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/recording_sessions/index

6.2 Auditing updates to roles & rules.

It may be critical to know which administrator made edits to role configurations, and when. While OpenAKC 1.0 does not currently have an interface for displaying this information, it is recorded in the the internal data folder and can be read by any administrator familiar with a "unified diff" format in the "/var/lib/openakc/data/history". Be aware that the format of this information in the "internal use" data folder may be subject to change in future, but this will be to facilitate a future mechanism to recover this data via the OpenAKC API.

6.3 SSH Agent

Because users are identified by the key fingerprint associated with their incoming SSH connection, it is possible to use "Agent Forwarding" to retain access to an SSH "identity" even after connecting to a shared role account! As a result, if a user "joe" uses "ssh -A root@server1", and while logged in as "root", they then run "ssh (-A) root@server2"… OpenAKC can still identify that connection as being opened by "joe"… and, assuming "joe" is permitted to connect to "root@server2" and there are no IP restrictions blocking the connection then the it will go ahead, and will be recorded as initiated by joe (Session recording will, if enabled, also show the connection as being established by "joe", and record as normal).

An example application for this is where multiple users are accessing a shared Ansible account on a development server, and pushing changes from there. The user "barrie", for example might connect to a shared "ansible" account on a deployment server, using "Agent Forwarding", and then be able to push changes to a group of hosts which "barrie" has access to, without the shared "ansible" account needing to be trusted in any way whatsoever! Indeed multiple different users could be doing this concurrently, and each would inherit a different set of permissions, based on their own identity, not that of the shared "ansible" account.

7 Examples

7.1 Creating a user for demo

In this example an additional user is added to demonstrate restrictions which could be applied to the “root” account, this user then creates an SSH key pair, registers with the OpenAKC server and we return to the admin account to create a role configuration for the demo.

```
root@openakc-combined:/# useradd -c "Restricited User" -k /etc/skel -s /bin/bash -m restricted-user
root@openakc-combined:/# su - restricted-user
restricted-user@openakc-combined:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/restricted-user/.ssh/id_rsa):
Created directory '/home/restricted-user/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/restricted-user/.ssh/id_rsa.
Your public key has been saved in /home/restricted-user/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:RCuYezJs2j+TVR40NrKQq8uDqrBVKGcI1M39p+YEa84 restricted-user@openakc-combined
The key's randomart image is:
+---[RSA 3072]----+
| .. o ...        |
|.  . =oo..=      |
|.   o .o+= o     |
|.. o ..+..o.     |
|o + B.. Soo.     |
| + =.+ o.+.      |
|. oo..+o+        |
|.o. +.+E .       |
|=.   ..o         |
+----[SHA256]-----+
restricted-user@openakc-combined:~$ openakc register
OpenAKC Copyright (C) 2019-2020  A. James Lewis.  Version is 0.99~ubuntu19.10-4.

This program comes with ABSOLUTELY NO WARRANTY; see "license" option.
This is free software, and you are welcome to redistribute it
under certain conditions; See LICENSE file for further details.

Passphrase is requested to ensure you own this key.

Enter passphrase:

Escalating to perform API call

Connected to OpenAKC server.  Sending key registration request
OK: Request processed

restricted-user@openakc-combined:~$ exit
logout
root@openakc-combined:/# su - admin-user
admin-user@openakc-combined:~$ openakc editrole app-user@openakc-client
```

7.2 Editing a role definition

Creating the role configuration was done using “editrole” which spawns an editor (Based on the EDITOR environment variable), as shown, with the role definitions applied for the demo.

```
# Edit this file to control permissions managed by OpenAKC.
# Rules for root@openakc-client.
#
# First matched rule will apply.
# {TYPE} can be user, group or key(ssh key fingerprint)
# FROM= can be either "any" or a comma separated list of IPs or CIDR subnets
# DAY= can be either "any" or a comma separated list of 3 letter day codes.
# TIM= is daily {Start Time},{End Time} 24 hour format, local server time.
# SHELL= users shell (could make shell "rjoe/rvim filename", or /bin/false).
# CMD= is a list of permitted commands, "any" or "comma separated list"
#   NB: 1) if CMD is not "any", it must include scp to allow SCP.
#       2) if CMD is not "any", sftp-server requires full path.
#       3) if CMD is exactly "internal-sftp", OpenAKC session will be
#          bypassed and "internal-sftp" will be forced.  Care! No capability
#          restrictions can be applied in this mode, and no session
#          recording will take place other than sftp's own logging.
#       4) Internal SFTP can be chrooted in sshd_config, while capabilities
#          can be applied to external sftp Eg (/usr/lib/openssh/sftp-server).
#       5) if a users calls an absolute path to a binary then CMD must
#          match the absolute path, otherwise short names are OK.
#   EG: CMD=scp,ls,uname,dmidecode,/usr/lib/openssh/sftp-server
# SCP= "sed" expression implementing chroot for scp connections.
# CAP= list of linux capabilities denied to role (man capabilities).
# REC= record session log, yes/no
#
# Eg: (Warning, TAGs must be CAPITAL, do not use quotes)
#
#ROLE=2020/01/13 19:17,2020/01/13 20:17,{TYPE},jlewis
#DAY=any
#TIM=any
#SHELL=/bin/bash
#CMD=any
#SCP=s,^/,/data/,g
#CAP=cap_linux_immutable
#REC=yes
#FROM=any
#
ROLE=2020/01/13 19:17,2025/01/13 20:17,user,normal-user
DAY=any
TIM=any
SHELL=/bin/bash
CMD=any
SCP=s,^/,/tmp/,g
CAP=cap_linux_immutable
REC=yes
FROM=any

ROLE=2020/01/13 19:17,2025/01/13 20:17,user,restricted-user
DAY=any
TIM=any
SHELL=/bin/bash
CMD=any
SCP=s,^/,/tmp/,g
CAP=cap_linux_immutable,cap_net_raw+ep
REC=yes
~
~
~
~
```

Note: This screenshot shows an early development release, the token “ROLE” shown has been replaced with “RULE”.

The “CAP=” field lists capabilities removed from that role, and you will note that the “restricted user” has an extra capability denied.

7.3 Different restrictions applied depending on rules

Now we log in to each of “normal-user” and then “restricted-user”, using the roles thus configured to access the “root” account on the target, and show that this role definition blocks access to raw network, even for the “root” account.

```
root@openakc-combined:/# su - normal-user
normal-user@openakc-combined:~$ ssh root@openakc-client
Enter passphrase for key '/home/normal-user/.ssh/id_rsa':
OpenAKC Interactive Session Initialized

Please enter the reason for your connection
Title / Change Number: 1234
Description (blank line to end)
Test restrictions

Restrictions Applied:-
Captbility cap_linux_immutable withdrawn by role config

root@openakc-client:~# tcpdump -ni eth0 port 25
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@openakc-client:~# exit
exit
Connection to openakc-client closed.
normal-user@openakc-combined:~$ exit
logout
root@openakc-combined:/#
root@openakc-combined:/#
root@openakc-combined:/#
root@openakc-combined:/# su - restricted-user
restricted-user@openakc-combined:~$ ssh root@openakc-client
Enter passphrase for key '/home/restricted-user/.ssh/id_rsa':
OpenAKC Interactive Session Initialized

Please enter the reason for your connection
Title / Change Number: 1234
Description (blank line to end)
Test restrictions

Restrictions Applied:-
Captbility cap_linux_immutable withdrawn by role config
Captbility cap_net_raw+ep withdrawn by role config

root@openakc-client:~# tcpdump -ni eth0 port 25
tcpdump: eth0: You don't have permission to capture on that device
(socket: Operation not permitted)
root@openakc-client:~# exit
exit
Connection to openakc-client closed.
restricted-user@openakc-combined:~$ exit
logout
root@openakc-combined:/#
```

7.4 Session recordings

Session logs for the restricted user record the event:-

```
root@openakc-combined:/# cd /var/lib/openakc/keylogs/2020/07/03/
root@openakc-combined:/var/lib/openakc/keylogs/2020/07/03# ls -lastr
total 60
 4 drwxr-xr-x 3 root root  4096 Jul  3 07:45 ..
 8 -rw-r--r-- 1 root root  4929 Jul  3 07:45 07:45:13-root@openakc-client.restricted-user.1348-000a0803.log
 4 -rw-r--r-- 1 root root   128 Jul  3 07:46 07:46:07-root@openakc-client.restricted-user.1794-000a0803.log
 4 -rw-r--r-- 1 root root   871 Jul  3 07:51 07:51:19-root@openakc-client.normal-user.2170-000a0803.log
 4 -rw-r--r-- 1 root root   128 Jul  3 07:53 07:52:59-root@openakc-client.normal-user.2543-000a0803.log
 4 -rw-r--r-- 1 root root   128 Jul  3 07:57 07:57:36-root@openakc-client.normal-user.2916-000a0803.log
 4 -rw-r--r-- 1 root root   128 Jul  3 08:00 08:00:13-root@openakc-client.restricted-user.3288-000a0803.log
20 -rw-r--r-- 1 root root 17618 Jul  3 08:32 08:32:16-root@openakc-client.restricted-user.3676-000a0803.log
 4 -rw-r--r-- 1 root root   710 Jul  3 08:37 08:34:51-root@openakc-client.restricted-user.4051-000a0803.log
 4 drwxr-xr-x 2 root root  4096 Jul  3 08:37 .
root@openakc-combined:/var/lib/openakc/keylogs/2020/07/03# cat 08:34:51-root@openakc-client.restricted-user.4051-000a0803.log
Title: 1234
Summary: Testing Restrictions
Summary:

Network Source: 10.0.3.217 55518 22
Key Used: SHA256:Tar9D8z4ok7BwYwTq5t4mtLTnEcuoecK0Dn04kqVcd0
TTY: /dev/pts/4
Term Type: xterm-256color
Term Language: en_US.UTF-8
Shell: /bin/bash
Restricted Capabilties: cap_linux_immutable,cap_net_raw

Script started on 2020-07-03 08:34:51+00:00 [TERM="xterm-256color" TTY="/dev/pts/4" COLUMNS="254" LINES="68"]
root@openakc-client:~# tcpdump -ni eth0 port 25
tcpdump: eth0: You don't have permission to capture on that device
(socket: Operation not permitted)
root@openakc-client:~# exit
exit

Script done on 2020-07-03 08:35:13+00:00 [COMMAND_EXIT_CODE="1"]
root@openakc-combined:/var/lib/openakc/keylogs/2020/07/03#
```

7.5 The power of rules

Another example of useful restrictions which could be applied using OpenAKC's capability function would be protecting sensitive user files from the "root" account. You may want to have a 1st line support group who do not have full access to the system, and in this example, you can see the power of the tool, as it blocks the "root" account from accessing a user's files, by blocking its ability to override file permissions… as well as its ability to change UID/GID via tools such as "su".

```
normal-user@openakc-combined:~$ ssh root@openakc-client
Enter passphrase for key '/home/normal-user/.ssh/id_rsa':
OpenAKC Interactive Session Initialized

Please enter the reason for your connection
Title / Change Number: 1234

Description (blank line to end)
Test Restrictions

Restrictions Applied:-
Captbility cap_linux_immutable withdrawn by role config
Captbility cap_setgid withdrawn by role config
Captbility cap_setuid withdrawn by role config
Captbility cap_chown withdrawn by role config
Captbility cap_dac_override withdrawn by role config
Captbility cap_dac_read_search withdrawn by role config
Captbility cap_fowner withdrawn by role config

root@openakc-client:~# cd /home/app-user/.ssh/
bash: cd: /home/app-user/.ssh/: Permission denied
root@openakc-client:~#
root@openakc-client:~# su - app-user
su: cannot set groups: Operation not permitted
root@openakc-client:~#
root@openakc-client:~# chown root:root /home/app-user/.ssh
chown: changing ownership of '/home/app-user/.ssh': Operation not permitted
root@openakc-client:~#
root@openakc-client:~# ls -asl /home/app-user/.bashrc
4 -rw-r--r-- 1 app-user app-user 3771 May  5  2019 /home/app-user/.bashrc
root@openakc-client:~# echo "Modified" >> /home/app-user/.bashrc
bash: /home/app-user/.bashrc: Permission denied
root@openakc-client:~#
root@openakc-client:~#
root@openakc-client:~#
root@openakc-client:~# echo "Modified" >> /etc/hosts
root@openakc-client:~#
root@openakc-client:~# reboot
Connection to openakc-client closed by remote host.
Connection to openakc-client closed.
normal-user@openakc-combined:~$
```

Of course, you may want to further protect the user data, by blocking access to raw disk IO, but remember, these restrictions will eventually impact normal operation. For example, a "root" account blocked from accessing user files can create entries in /etc/passwd, /etc/shadow etc…. So could create a new user, but could not then create that user's home folder. Always test new restrictions to understand where they may not be suitable.

7.6 Non-interactive commands & SCP

In some other examples, session logs are able to show both non-interactive commands, and scp file transfers, as in this example, where we can see that the user executed a remote (non-interactive) command "uname -a", and how this is recorded in the session log.

```
noping-user@openakc-combined:~$ ssh app-user@openakc-client 'uname -a'
Enter passphrase for key '/home/noping-user/.ssh/id_rsa':
Linux openakc-client 5.4.0-8-generic #11-Ubuntu SMP Fri Dec 6 22:43:53 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
noping-user@openakc-combined:~$ exit
logout
root@openakc-combined:/# cat /var/lib/openakc/keylogs/2020/07/03/03\:29\:07-app-user\@openakc-client.noping-user.2533-000a2603.log
"uname -a" called via ssh

Script started on 2020-07-03 03:29:07+00:00 [<not executed on terminal>]
Linux openakc-client 5.4.0-8-generic #11-Ubuntu SMP Fri Dec 6 22:43:53 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux

root@openakc-combined:/#
```

As well as SCP transfers. Note how the target is changed from /myfile to /tmp/myfile by the search/replace “sed” script in the role configuration.

```
normal-user@openakc-combined:~$ scp /etc/hosts app-user@openakc-client:/myfile
Enter passphrase for key '/home/normal-user/.ssh/id_rsa':
hosts                                                 100%  182   664.7KB/s   00:00
normal-user@openakc-combined:~$ exit
logout
root@openakc-combined:/# cat /var/lib/openakc/keylogs/2020/07/03/03\:
03:29:07-app-user@openakc-client.noping-user.2533-000a2603.log  03:38:12-root@openakc-client.normal-user.3257-000a2603.log
03:37:47-root@openakc-client.normal-user.2884-000a2603.log      03:39:22-app-user@openakc-client.normal-user.3629-000a2603.log
root@openakc-combined:/# cat /var/lib/openakc/keylogs/2020/07/03/03\:39\:22-app-user\@openakc-client.normal-user.3629-000a2603.log
"scp -t /myfile" called via ssh

SCP PUT /tmp/myfile executed, exit code 0
```

And into the local log on the client machine.

```
root@openakc-client:~# grep scp /var/log/syslog
Jul  3 03:39:22 openakc-client openakc-scp: PUT /tmp/myfile ; Succeeded for user app-user
root@openakc-client:~# exit
exit
Connection to openakc-client closed.
normal-user@openakc-combined:~$ exit
```

8 Appendices

8.1 Command Reference & Examples

8.1.1 Command “openakc-plugin”

The tool “openakc-plugin” is primarily the interface between the ssh daemon, on the client machine and the OpenAKC back end server via the API. It does however have a couple of additional functions which are are useful for debugging issues.

1. When used as a plug-in, it is called in the “sshd_config” as follows (note that the installation process will update the ssh configuration so you should not need to do this manually):-

```
AuthorizedKeysCommand /usr/sbin/openakc-plugin %u %h %f
AuthorizedKeysCommandUser openakc
```

The parameters passed to the plug-in include the local user and home folder of the local user for which the incoming connection wishes to be authenticated, as well as the key fingerprint of the key being used as a token. Also, notice that the client runs as an unprivileged user.

2. When running as a normal user (not root), the only other possible use of this tool is to test connectivity to the server:-

```
$ openakc-plugin ping
```

The server response “OK: Pong!” will be seen if connection is working OK.

3. When running as root there is one additional option available, this is to reset the server public keys stored on the client. This might be required if the servers identity changes. In this scenario, the client will no longer communicate with the server until the keys are reset. This is similar to the way SSH itself will warn before connecting to a server who’s host keys have changed:-

```
$ openakc-plugin resetkeys
```

8.1.2 Command “openakc”

The tool “openakc” is the tool used by both users, and admins to interact with the API from the command line.

8.1.2.1 Users

OpenAKC is largely intended to be transparent to the user at the point of use, so with the exception of restrictions imposed by the rules associated with each role (or the login reason “quiz”), the only real reason for a normal “user” to interact with AKC is to register their public key.

```
$ openakc register
```

This verifies the ownership of the public key and identity of the user, then records this information with the back end server via the API. Once recorded in this way, the user’s public key will be made available on client machines wherever they wish to log in, so long as the role rules governing access to that system permit them to log in.

Eg. If a new web developer “fred” joins, assuming his user account in AD (or LDAP etc.) is created as a member of the group “webadmins”, then provided there is a rule permitting members of the “webadmins” group to login in via ssh to “apache@webserver”, then “fred” only needs to create and register a suitable ssh key pair (pass phrases are mandatory), and he will instantly be able to log in to that account. There is no delay while keys are distributed, since keys are not distributed but are accessed on demand by the client as part of the authentication process.

8.1.2.2 Admins

Security administrators have two functions they can perform via the “openakc” tool.

1. Defining role rules for access to given clients, this can be done either via an editor (defined by the EDITOR environment variable).

```
$ openakc editrole username@hostname
```

Alternatively, they can download the existing roles, or upload a pre-defined set of rules.

```
$ openakc getrole username@hostname filename
```
or

```
$ openakc setrole username@hostname filename
```

The “getrole” command will overwrite file defined by “filename” with the current rules for the specified account, while the “setrole” command will upload the content of “filename” to replace the existing rules.

2. Managing static trust (equivalent to authorized_keys).

Public keys can be uploaded to the system using the “submit” command (pass phases are not enforced for this mode, but obviously recommended where possible). The openakc tool will provide a serial number when a key is submitted.

```
$ openakc submit filename “Comment Text”
```

```
admin-user@openakc-combined:~$ openakc submit /tmp/id_demo "Demo Key"
OpenAKC Copyright (C) 2019-2020  A. James Lewis.  Version is 0.99~ubuntu20.04-7.

This program comes with ABSOLUTELY NO WARRANTY; see "license" option.
This is free software, and you are welcome to redistribute it
under certain conditions; See LICENSE file for further details.

Storing static key /tmp/id_demo with comment "Demo Key"
Connected to OpenAKC server.  Sending key submission request
OK: Stored static key - S/N = key-kIbjBGBvnO
```

Existing public keys can be updated without changing the serial number using the “update” command. The openakc tool will overwrite the key with the specified serial number, which is useful because rules can refer to keys by serial number, thus allowing keys to be replaced/rotated without updating all the rules which might refer to them.

```
$ openakc update key-serial filename “Comment Text”
```

As a subset of the update feature, a key can be removed from the server as follows.

```
$ openakc update {key-serial|username} DELETE
```

Note that the delete feature can also be used for user keys, perhaps when a member of staff leaves and their account is removed. Obviously deleting a user’s key would be a housekeeping function (perhaps part of a JML process), as their key would not be useful after their account was deleted in whatever directory was used, but for static trust relationships it would definitely be easier to delete a key than search all the rules to see if it is referenced.

8.2 Automated Test Environment

There is a script located in the GIT repo, which will set up a demo/test environment on Debian/Ubuntu, it will automate configuration of 1 user and one admin for testing.

The script located in OpenAKC/contrib/debian-lxc-build+demo.sh will configure a basic environment for testing and/or demonstration purposes.

Eg:-

```
root@trinity:/home/james/GIT/OpenAKC/contrib# ./debian-lxc-build+demo.sh
Setting up OpenAKC demo environment, using standard LXC containers.

Checking for installed package, lxc - Found!
Checking for installed package, uidmap - Found!
Checking for installed package, bridge-utils - Found!
Checking for installed package, debootstrap - Found!
Checking for installed package, dnsmasq-base - Found!
Checking for installed package, gnupg - Found!
Checking for installed package, iproute2 - Found!
Checking for installed package, iptables - Found!
Checking for installed package, lxc-templates - Found!
Checking for installed package, lxcfs - Found!
Checking for installed package, openssl - Found!
Checking for installed package, rsync - Found!
Checking for installed package, acl - Found!

All Components found, continueing.

Containers will be "openakc-combined" & "openakc-client".

"openakc-combined" will contain the OpenAKC server, and user
access host (and be used to build the packages)

"openakc-client" will be the demo client system for OpenAKC
to grant access

The next step will destroy the test containers and rebuild!

Press ENTER to rebuild test containers, or ^C to abort


Destroying old containers...

Installing with options "-d ubuntu -r eoan -a amd64", please wait...

Almost done!...

NAME             STATE   AUTOSTART GROUPS IPV4 IPV6 UNPRIVILEGED
RH6-build        STOPPED 0         -      -    -    false
RH6-tmp          STOPPED 0         -      -    -    false
centos8          STOPPED 0         -      -    -    false
openakc-client   RUNNING 0         -      -    -    false
openakc-combined RUNNING 0         -      -    -    false
syslog-test      STOPPED 0         -      -    -    false

Waiting for new containers to settle

Setting up containers
```

A lot of output is generated at this point as the containers are set up, and packages compiled, so it is not included here.

After the containers are built, packages compiled and installed, the user is asked to enter pre-defined pass phrases into the OpenAKC registration tool, before the example role definition from the contrib folder is applied to a target application user on the demo client machine.

```
Using openakc setrole (as admin-user) to upload the example role configuration
openakc setrole app-user@openakc-client /tmp/examplerole
NB: use "openakc editrole app-user@openakc-client" for interactive configuation

OpenAKC Copyright (C) 2019-2020  A. James Lewis.  Version is 0.99~ubuntu19.10-4.

This program comes with ABSOLUTELY NO WARRANTY; see "license" option.
This is free software, and you are welcome to redistribute it
under certain conditions; See LICENSE file for further details.

Connected to OpenAKC server.  Sending role update request
OK: Request processed, role data updated

Done!

You should now connect to the "openakc-combined" container,
Then verify that the "normal-user" account can successfully connect
using "ssh app-user@openakc-client"

You should now have a working demo/sample install in the containers!

To access the container, type "lxc-attach -n openakc-combined"
then, "su - normal-user"
then, "ssh app-user@openakc-client"

If everything above worked, you should be able to connect using the "normal-user" key (Use passphrase: "userkey")

root@trinity:/home/james/GIT/OpenAKC/contrib# lxc-attach -n openakc-combined
```

8.3 API Reference & Examples

The API has 2 stages. Stage one is unencrypted, and allows the client to establish the connection, and negotiate the encryption, while stage 2 is where the business of exchanging credentials and performing authentication happens.

8.3.1 Stage 1 Options

help

ping

getproto

setproto

quit

8.3.2 Stage 2 Options

Note that for debugging purposes, stage 2 commands can be sent unencrypted, or sent with encryption using the “message” command.

help

pubkey

sessioncode

challenge

handshake

setrole

getrole

listrole

registerkey

submit

update

sessiondata

message

audit

auth

ping

logturn

quit