System Security

Security

The system defaults to a self-signed web certificate.

- A unique web certificate can be copied onto the host using **scp** or **system download**.
- The web certificate is installed using **web cert add <certificate file>**.

SSH keys are used for sftp, passwordless ssh and scp.

- Keys can be created using **keys createkey**.
- The public key copied to a remote host using **keys sendkey <user@host>**.
- A host can be authorized for incoming connections using **keys add <host>**.

The system uses an internal repository to check whether security package updates are available. Additional repositories can be added with:

security repos add <repo-name> <url> <distro> <section> <categories>

For example, **security repos mymirror add http://archive.ubuntu.com/ubuntu/ precise-updates main universe multiverse**

In order to check whether there are security updates available, use:

security check

The system can be updated using:

security update

Security Patches and Updates

During installation the system will automatically install the application named "security" which is a collection of all the latest security patches available for the various pieces of software in the platform at the time the system was built. Updates to this application are released to customers regularly. The security application provides these updates but does not automatically install them - allowing customers with concerns to verify them on lab machines first for example. Some security updates may also require scheduled downtime to complete and for this reason the final installation of updates is a manually triggered process.

The health command will inform the user if any security updates are currently available but not installed. Users can install security updates at any time by running the command:

security update

Those who would prefer to automate this can create a scheduled command to do so on a regular basis. The security update will install all operating system updates to both the main system and the application jails, but it will not generally contain updates to the core applications themselves - these are shipped separately as new application install versions as they require additional QA to ensure compatibility.

Using Your Own Repository Mirrors

Some users may prefer to obtain security updates from the upstream distribution more frequently than they are repackaged by Cisco Unified Communications Domain Manager 10.6(1) . This is fully supported by the security application.

From the command line, use the command:

security

to see the list of available command parameters.

Users can use the command:

security repos

to list current repositories from which updates are available.

Use the command:

security repos add <repo>

to add repositories from which updates are available. These repositories are in the same standard format as used by the underlying operating system.

For example on Ubuntu platforms they will resemble the following line:

security repos add http://security.ubuntu.com/ubuntu precise-security main restricted

You can point the system directly at the upstream provider or set up an internal mirror on your network which hosts only the updates you have approved, as long as the repository is network reachable by the platform it will function.

After adding your repositories you can check for any new available updates using the command:

security check

This command is also periodically run by the system itself and will report if any new updates are available. The command security update will install updates from your own repositories just as it would for the updates shipped with the platform.

To remove repositories, use the command:

security repos del <repo>

Configuration Encrypted

In order to help protect customer data and service stability the system configuration files are frequently recreated by the platform. This means that even malicious tampering to the platform will generally be undone by a simple restart. The data used to do manage this is stored in the platform's internal configuration files. These files are encrypted using a strong AES encryption layer to make them tamper-proof. They are never decrypted on disk, instead the applications which manage them will decrypt them in memory, read and make modifications as needed and then re-encrypt the data before writing them back to disk.

In this way the risk of tampering or data theft through the configuration system is greatly minimized.

Backup Encrypted

System backups include copies of the full system configuration as well as the full contents of the database. Thus theft of a backup would effectively constitute theft of all customer data stored on the platform. To mitigate this risk backups are encrypted using a strong 2048-bit in-line GPG encryption.

The encryption key for this is auto-generated by the platform based on a unique machine UUID. While it's possible for support to recover backups from a different machine this process is deliberately hard and only available to official technical support representatives. Backups on shared locations are separated on a per-source-machine basis to prevent conflicts.

Application Install Files Encrypted

In order to protect the trustworthiness of applications shipped for the platform, all application installers are encrypted files. The strong 2048-bit key needed to decrypt these are shipped with the platform and is different from the per-machine unique keys used for other encryption tasks. This key will only decrypt applications encrypted specifically with the unique key owned by Cisco Unified Communications Domain Manager 10.6(1). The system will refuse to install any application that is not encrypted or encrypted with a different key.

This ensures that only valid, untampered copies of genuine Cisco Unified Communications Domain Manager 10.6(1)-released applications can be installed on the system.

Protected Application Environments (Jails)

Cisco Unified Communications Domain Manager 10.6(1) runs the service providing applications in secured jail environments. This has significant value for the security and reliability of the system. It prevents applications from cross-interfering which makes the system more stable and reliable. In terms of security it effectively confines all services to dedicated and separate mini file systems with predictable content. In the event that an attacker were to gain access to the system through a vulnerability in a service he would therefore not gain access to the platform but only to the small confined jail in which the service was running. In that environment only the jail itself is vulnerable and this can be very easily restored if damaged. The underlying system cannot be accessed from the jailed environment.

Restricted User Shell

The platform attempts to reduce the risk of unintentional harm to the operation of the software by restricting the actions users can take. This is done using a specially configured setup of the well-known and actively maintained rbash shell.

The shell actively prevents the following:

- Users cannot set environment variables or alter their command path.
- Users cannot change the current directory.
- Users cannot specify a path to a command to run.

The commands users thus are able to run is only what is allowed by the platform setup. The vast majority of these commands use a common execution interface designed to allow only enough privileges to perform the system administration tasks they are created for. The exact list of commands a user can run is determined by his specific privileges and the specific setup of the machine on which he is working (different applications can add their own additional commands). This list is displayed on login and can be redisplayed with the **help** command.

Creating Additional Users

During installation a user called 'platform' is created which has full access to all allowed commands within the restricted environment. This user (and others with the appropriate rights) can then create additional users who are further restricted to only be able to run certain commands. For example a user could be created who can only run diagnostic and logging commands - able to monitor the health of a system but required to escalate any actions.

Users are created, managed and deleted through the user command. To create a new user use:

user add <username>

The system will create a Unix user with the name specified and set up to use a restricted shell identical to the platform user. Initially this user's password is set to match the username but it must be changed on first login. New users start out with no rights and can only run the very basic system commands provided to all users (such as **ls**).

Granting and revoking user rights

Once a user is added the user needs to be granted access to run commands. The user's command menu will only display those commands to which access have been granted.

To grant access to a command use the 'user grant' command as follows:

user grant <username> <command> [options]

Only one command can be granted at a time, however these can be complex. The more detailed the command, the more fine-grained the privilege becomes. This is best explained by example.

Running the following command:

user grant peter app

Will allow the user peter to execute any command within the 'app' series of commands. However it could be restricted further by instead running a command like:

user grant peter app list

With this version peter will see the **app** command on his menu, but its help will only display 'list' as a sub-command - peter can thus see the list of apps but cannot perform more potentially risky tasks such as installing or restarting applications.

This can be expanded to other subsets by simply running additional grants:

user grant peter app start

Would now allow peter to both see the list of applications or restart applications that failed, however he will not be able to do other app related tasks such as installations. The **grant** command effectively verifies that the start of a command by a user matches one of the privileges granted to that user - so peter will be able to add options to any command he is granted access to.

In order to restrict commands - be sure to determine whether any options should be allowed and if not, only grant access to the specific parameters you wish peter to be able to execute. For example if peter is your database administrator for example you may wish to use:

user grant peter app start mongodb

Instead of giving access to all **app start** commands.

Should you wish to revoke a command privilege from a user you can do this using the following command:

user revoke <username> <full command>

The command being revoked must match exactly one of the commands previously granted to a user. To review the current privileges of a user use:

user list <username>

Which will display the user's entire list of granted commands in full. You can also just run

user list

Without an option to list all users created on your system and their privileges.

Password Strength Rules

The platform user and any users created are held to strong password rules to help reduce the risk of system penetration. These rules are enforced whenever passwords are changed or set. In order to meet system password strength rules a user's password must:

- Be at least 16 characters long.
- Contain at least one capital and one non-capital letter.
- Contain at least one number.
- Contain at least one special character.

SSH key management

SSH authentication requires maintaining the system SSH keys. This can be done as follows:

- **keys create** creates a local SSH keyset
- **keys send <user>@<host>** will send the public key from the local SSH keyset to the remote server, thereby enable remote SSH authentication.
- **keys add <host>** adds the remote host to the known hosts list allowing outgoing connections

For example, if you wish to perform a backup to a remote host, first create a local key if necessary with **keys create**. Allow communication with the host using **keys add <host>**. Send the key to the remote host with **keys send <user>@<host>**.

The certificates are independent of web servers/proxies.

For more details on SSH key-based authentication, refer to OpenSSH documentation.