SSH Server with User Key Example

1. Introduction

This example demonstrates how to use the NetBurner SSH library to create a SSH server that can utilize the SSH library key, user application compiled-in key and upload a user supplied key.

This application requires the optional NetBurner SSH/SSL Encryption software.

Application Features:

- SSH Server.
- Ability to upload a user provided SSH key through a web page using multipart forms.
- SSH Library Key and Application compiled-in key to provide security if a user key has not been uploaded to the device.
- Embedded Flash File System (EFFS-STD) for on-chip storage of user key.
- Web page interface for viewing and uploading keys.
- Storage of user parameters in the User Flash storage area.
- Optional NTP client to set system date and time. Note that you must have Internet access to communication with the NTP server.
- Serial port debug/command interface to display system status, file system status, and format of file system.

2. Running the Example

- Run IPSetup to determine the IP settings of your NetBurner device.
- Run the NetBurner MTTTY serial terminal program to view status messages from the application. You can use either USB or a RS-232 DB9 connection with the NetBurner development board.
- Enter the IP address into the URL of a web browser to view the main web page for viewing and maintenance of the SSH keys.
- Use the Putty SSH Windows Client program to connect to the NetBurner device using a secure SSH connection.
- You may create your own SSH RSA or DSA key using a program such as Putty and upload it to the NetBurner device through the web interface.
- Any username and password will allow a login as long as the two are not the same value.

3. Description of Application Files

Files	Description
main.cpp	Contains the UserMain() thread which starts the application and provides initialization of subsystems such as the EFFS-STD file system, FTP, NTP and Multipart forms. It also contains the command parser for the serial command interface.
EffsStdFlashDrv.c FileSystemUtils.cpp FileSystemUtils.h fs_main.c fs_main.h	EFFS-STD file system configuration, initialization and function interface.
AM29LV160B.h AT49BV163D.h SST39VF040.h	EFFS-STD header files for the type of physical memory on your NetBurner device. **WARNING: You must use the proper header file for you NetBurner device or the flash memory may be corrupted.** If you are new to the EFFS-STD flash file system, please refer to the NetBurner EFFS Programmers Guide.
effs_time.cpp effs_time.h	File system timing functions including file stamps and NTP.
permanentkeydss.h permanentkeyrsa.h	The permanent SSH keys that will be compiled into the application code. If you wish to have your own default keys you can create your own certificate and key, and replace the data inside these files.
sshuser.cpp sshuser.h	SSH functions.
web.cpp	Web page interface functions
index.htm	Main web page
nbfactory.h	Constant definitions for things like the file names of the keys stored in the flash file system, application name, platform name and web page key names.
key.cpp	This is the key of last resort for SSL. It is used if both the permanenetkeyxxx.h files and any uploaded user certs are invalid. This will enable the uploading of a user certificate that is valid.

4. Application Code Details

4.1. Major Points and Data Storage

- The objective of the example is to demonstrate how a SSH key can be uploaded to a NetBurner device and used by the SSH Server to accept incoming SSH connections for a client such as Putty.

- There are many ways to put a SSH key in your NetBurner device. We chose the web server in this example to have better control over the user interface. We purposely did not use the secure SSL web server option to avoid the complexity of managing an additional certificate and key for SSL. The objective of the example is to focus on SSH with the minimal amount of complexity.

- The RSA key uses RSA encryption. The DSA key uses DSS encryption. You will see references to both DSA and DSS in the code, but both acronyms refer to the DSA key.

- There are 3 keys in the application: the NetBurner SSH/SSL library keys, the keys stored in the application header files permanentkeyrsa.h and permanentkeydss.h, and the keys that can be uploaded through the web page. The keys are used in that order. If the user key is invalid or does not exist, the application key is used, if the application key is invalid (you can supply this one at compile time), then the NetBurner library key is used. The objective is to make sure there is at least one good key available so the device can always be accessed.

- The EFFS-STD on-chip file system is used to store the RSA and DSA keys. These single RSA and DSA key files always contain the active SSH key, whether the data was copied from the NetBurner SSH/SSL Library, application permanent key, or uploaded by the user via the web page. The default behavior is to copy the keys from the application's permanent key header files, then overwrite these files when a user uploads their own key.

- User of the EFFS-STD file system requires that you specify compcode settings in your project and know exactly what flash chip is used on your NetBurner device. If these settings are incorrect it is very likely the flash memory of your NetBurner device will be corrupted and the boot monitor will not be able to resolve the problem. Please refer to the EFFS-STD section of the EFFS Programmers Guide for additional information.

- The User Parameter Flash Storage area sector of flash is used to store the application settings. You could use the flash file system for this purpose instead.

- The web page contains links to display the public RSA and DSA keys so you can copy the key and install it into your SSH client program to ensure a secure connection. If you do not do this, the default behavior of a SSH client is to put up a dialog box asking the user to accept the connection anyway.

- The active SSH keys, once read from the file system, are stored in global arrays allocated in sshuser.cpp:

```
char* gSshRsaKeyPemEncoded[ ( SSH_KEY_SIZE_MAX_PEM + 1 ) ];
char* gSshDsaKeyPemEncoded[ ( SSH_KEY_SIZE_MAX_PEM + 1 ) ];
```

4.2. Initialization

The application initialization takes place in UserMain() of main.cpp:

- The TCP/IP Stack is initialized, the AutoUpdate utility is enabled, and the web server is started.
- The web server option of handling Multipart Forms is enabled, and a buffer is specified to handle the largest key file size we expect to handle.
- The EFFS-STD file system is initialized.
- The NV_Settings user parameter structure is initialized. If this is the first time the application has run, the structure is filled with default values and the functions that create the default SSH keys are executed to create the RSA and DSA key files in the Flash file system. Otherwise the saved values are copied into a runtime structure variable, and the existing SSH key files are used as the active keys in the application.
- The callback function used by the SSH library to authenticate user name and password login is assigned by: SshSetUserAuthenticate(SshUserAuthenticate).
- The callback function used by the SSH library to obtain the active SSH key is assigned by: SshSetUserGetKey(SshUserGetKey)

After the initialization process the UserMain() task processes all I/O from the debug serial port.

4.3. Web Page Display of Active Keys

The functions that process the web page interaction are located in web.cpp. When a user clicks on a link to display a public key, the functions DisplayDsaPublicKey() and DisplayRsaPublicKey() are called to create a web page and display the key data.

4.4. Uploading a RSA or DSA Key

This is where the most action occurs. The process to upload a key is:

- A user clicks on the Browse button, selects a SSH key file, and clicks on the Install Key submit button to initiate a multiform file post to the NetBurner web server.

- The initialization in main.cpp included assigning the MyDoPost() callback function to the web server to process form posts. Once called, it checks to see if the Install Key or Reset Keys button was pressed. If Install Key, it calls the SshKeysPost() function.
- The SshKeysPost() function reads the key file, calls SshUserValidateKey() to veify the key is good, determines if it is an RSA or DSA key, saves the new key in the flash file system, and updates the NV_Settings structure.

4.5. Resetting Keys to Factory Defaults

When a user clicks on the Reset Keys button on the web page, the CheckNVSettings(TRUE) function is called, where the parameter TRUE forces a re-initialization of the SSH keys and NV_Settings structure to default values.