DATUM SYSTEMS

SnIP Remote Execution Guide

SnIP875 Technical Note

Revision History

Rev 0.1	8-15-2009	Initial Release.
Rev 0.2	6-14-2010	Add new lexec script information.

1.0 SnIP Remote Execution Overview

The Satellite network Interface Processor or “SnIP” is a fairly complete Linux computer on a small card which can be installed into a PSM-500 series modem. It can serve as both a selectable data interface and an IP based monitor and control element for the modem independently.

Why Remote Execution?

The SnIP has multiple capabilities for controlling both the SnIP configuration and operation and that of the modem itself. In most communications systems there is a need to control the system components from various distributed or central control points. Linux is capable of multiple methods which allow control from those diverse control points by either manual or automated procedures. Almost all desired connection methods are via Ethernet using TCP/IP and Linux provides multiple IP connection types or "sockets".

Manual Methods:

- ❑ Both semi-secure Telnet and very secure SSH type session connections.
- ❑ Standard HTTP and secure HTTPS web browser access via the SnIPs web server.

Automated Methods:

- ❑ SSH RSA key secure authentication and execution.
- ❑ SNMP semi-secure formatted authentication and execution.

The purpose of this document is to explain the procedures necessary to accomplish manual and automated control using SSH RSA key secure authentication and execution.

1.1 Equipment Setup

To allow any type of socket connection and execution via IP requires that all SnIPs be accessible via the Ethernet or WAN (modem) connection. This in turn requires proper IP addresses and masks are assigned to each of the SnIPs. It also requires a control computer where an operator or automated program functions to access the SnIPs for control.

Any communications system and especially a satellite network or link encounters potential problems where remote devices that are part of the communications link itself can be set so that contact is lost and cannot be easily recovered. Care must be exercised to avoid this possibility.

1.2 SSH and Secure Key Introduction

SSH implementations are common on networks and the Internet. They are a built in part of most Unix/Linux systems and much less common on Windows platforms. The SnIP and virtually all Linux PC systems have SSH or OpenSSH built in. In Windows the open source program named “***PuTTY***” also implements a compatible form of SSH which can also be used.

The default use of SSH which includes also SCP and SFTP is similar to Telnet in that a remote user logs into a user account by correctly providing the user’s password. When you log into a SnIP via SSH or send a file via SCP there is more authentication than just the user name and password, there are also key authentication procedures which occur between the originating and destination computers. The procedures involve an exchange about private and public keys which are normally kept in the “Known Hosts” file in each user's “.ssh” hidden directory. Telnet has the security disadvantage that the password is sent in the clear which could be captured externally.

All of the SSH implementations also have provisions to generate and distribute additional secure keys which can be used for automated authentication. Secure Keys are generated in "pairs", one public and one private. The private key is kept on the client computer and the public key is given to servers that the client wishes to communicate with in a secure manner. The desired key type in all cases is "RSA" which is used in the more modern SSH2 implementation and by default we use a 1024 bit code.

What makes secure SSH key based authentication desirable is that by the use of keys a client computer can essentially automatically login to a server and execute commands. The SnIP is a server in this application. For example, you could manually log into a SnIP using an SSH session via Ethernet and run a command to look at the current directory listing by going through a procedure like:

```
ssh root@192.168.22.33
enter password for root
#> ls
--- directory listing shown ---
#> exit
```

After generating keys and distributing them the same client can now simply say:

```
ssh <key name> root@192.168.22.33 "ls"
```

which will produce the directory listing on the computer (or SnIP) with that IP address and return control to the client computer. That is a procedure which can be easily used within a program or script on the client computer.

Our purpose is not to just look at directory listings, but to use more constructive SnIP commands, for example, the SnIP program named "**m500ctl**" which can be used to monitor and control the over 100 PSM-500 series modem parameters using fairly simple mnemonics. Using m500ctl to set the transmit frequency on the modem whose SnIP IP address is 192.168.22.33 we would enter at a controller session:

```
ssh root@192.168.22.33 "m500ctl 0 mif 82465000"
```

There are only 3 variable elements in this command:

1. The ID of the modem/SnIP (IP address of 192.168.22.33),
2. the parameter to be read or set - "mif" which is the mnemonic for Modulator IF Frequency,
3. and the value of the parameter, in this case 82.465 MHz.

Note: Many implementations also require a 4th element which is the name of the key to be used as shown in the original ssh command.

Note:

“m500ctl” Modem Control Basics

The “m500ctl” program is built into the SnIP and is described in more detail in the SnIP Quick Start Guide and briefly in the usage statement obtained from a SnIP session by just typing “**m500ctl**”. For more detailed information see the ***SnIP Modem Control Guide***.

Use consists of following the “m500ctl” program invocation with optional switches, the desired modem address (0 for the host modem) plus at least one command and parameter pair. Commands are mnemonics which can be viewed by typing “**m500ctl –c**”. Parameters are read by using a “**?**” or set by using a value.

In the example modem commands above the method for reading a parameter is to replace the value with a "?". The value could also have been entered as "82.465" and the program would have properly read and set it, but the response will always be returned in the full format to Hz resolution, so keeping all values in their least increment representation avoids potential problems.

These elements can easily be set as variables in all common scripting languages such as bash, Perl, PHP, Python or Ruby. With a slight bit more effort they could be used in compiled languages such as C, C++ or Java.

2.0 Setup for Remote SSH Execution

To enable remote SSH Execution you must generate an RSA secure key pair and distribute the public key as needed. Each control computer (client) need only generate a single key pair and the public key is sent to all SnIPs which it will be allowed to control. The SnIP can accept multiple public keys in its "**~/.ssh/authorized_keys**" file. The procedures below append the public key in this file. Note "~" or tilde is a real character and valid argument in a Linux file system meaning the current user's home directory.

The specific method for generating and distributing keys and remote execution varies on each client platform type and are shown in the formats below.

2.1 Format 1 - From a Linux PC to one or more SnIPs

On most modern Linux PC distributions such as Ubuntu or Fedora or Red Hat use the following command to generate an RSA key pair. There is one private and one public key

```
ssh-keygen -t rsa
```

This will produce the keys in the default ~/.ssh/id_rsa and id_rsa.pub. The public key (.pub) is the one that is sent to the server (SnIP in this case) where you want to login without a password or execute remotely. Now copy the public key to each Linux computer or SnIP that remote control of is desired. For example to the SnIP at the IP Address of 192.168.15.33 you would use the command:

```
ssh-copy-id -i .ssh/id_rsa.pub root@192.168.15.33
```

The public key is automatically appended to the SnIP's ~/.ssh/authorized_keys file using this command. Since the keys are not set up yet you will be required to enter the password for the SnIP root user (default "datum").

After the public key is installed you should now be able to execute a command on the remote SnIP (example "ls") by using the following command on the Linux PC in a terminal session:

```
ssh root@192.168.15.33 ls -l or
ssh root@192.168.15.33 "ls -l"
```

You can also now log in using just **'ssh root@192.168.15.33'** and no password is requested. Both the SnIP and the Linux PC remember the keys and allow this interaction even after reboot. If the SnIP has new software installed or loses its .ssh/authorized_keys file then the “ssh-copy-id” command given above can be used again to restore it.

Exchange of the private key is not limited to a single SnIP. One possible use for remote execution is to allow a single Linux PC to control multiple SnIP/Modems. As an example consider a hub station with multiple modems where the same public key is sent to each of the SnIPs. Then commands can be directed to the correct unit by using its IP Address. Or a custom Network Management System could distribute one public key to all of the SnIPs used for control anywhere reachable within the system.

2.2 Format 2 - From a SnIP to one or more SnIPs

The capability to remotely execute from one SnIP to another is central to implementing “Link Command and Control” which involves control of both modems in a link as if it was a single entity. For more information about Link Control and Link Partners refer to a companion SnIP Guide named “***SnIP Modem Control Guide***”.

The SnIP uses an embedded version of the OpenSSH suite called "**dropbear**" which is very similar in functionality to that in a larger PC based Linux system. Before beginning make sure that the /root/.ssh directory exists, and create it if not. Note: The “ssh-setup-lp” script discussed in Section 2.2.1 below makes this check and directory creation if necessary plus creation and transfer of the keys.

To create a key pair manually - On the originating SnIP from the /root directory use the following command to generate an RSA key pair. This command creates one private and one companion public key.

```
dropbearkey -t rsa -f ~/.ssh/id_rsa  | grep ssh-rsa > .ssh/id_rsa.pub
```

Note this is a slightly more complex Linux command and uses the “pipe” directive “|” and has a “.” preceding the directory name ssh since it is normally hidden.This will produce the keys in the default ~/.ssh/id_rsa and id_rsa.pub files. The public key (.pub) is the one that is sent to the server (SnIP in this case) where you want to login without a password or execute remotely. Now copy the public key to each SnIP you want to work with:

```
cat ~/.ssh/id_rsa.pub | dbclient root@192.168.15.35 "cat >> ~/.ssh/authorized_keys"
```

The public key is automatically appended to the ~/.ssh/authorized_keys file by the above command. This command is even more complex and it may be possible to use cut and paste to copy the command directly into a SnIP session console. Remember that in Linux and the SnIP the paste command is the keys “Shift-Ctrl-v” not the common “Ctrl-v” used in graphical environments.

In filesystem versions 0.6.11 and above scripts were created to simulate the full ssh suite ssh-keygen and ssh-copy-id commands. In these versions you can use commands similar to those from a full Linux ssh implementation.

```
ssh-keygen id_rsa          <== differs from normal, specifying the filename only – rsa assumed
ssh-copy-id -i ~/.ssh/id_rsa.pub root@192.168.15.33
```

Depending on previous commands on the local and remote SnIP, there may be no /root/.ssh directory, in which case one needs to be created by logging in and executing “mkdir /root/.ssh”.

To execute a command on the remote SnIP it should now be possible to use the following command sequence:

```
dbclient -i ~/.ssh/id_rsa root@192.168.15.35 ls -l or
dbclient -i ~/.ssh/id_rsa root@192.168.15.35 "ls -l"
```

You can now also log in using just:

```
dbclient -i ~/.ssh/id_rsa root@192.168.15.33
```

and no password is required. You will note here that the commands are more complex looking that the equivalent ssh commands used from full Linux with SSH installed. Version 0.6.11 and above filesystems can normally use the alias ssh instead of dbclient, but the -i key_filename parameter is still required.

2.2.1 Scripts to Make Remote Execution Setup Easier

The above SnIP command line requirements for setup and remote execution are classic Linux – that is too complex for the casual user. To avoid problems we created two newer scripts which attempt to not only make entry less prone to error, but also usable by web or other user interface programs. Those are named “**ssh-setup-lp**” and “**lexec**”. The first is a single script to allow fairly complete setup of the required ssh keys and copying them to the desired SnIP. The command is normally executed from a command line in one SnIP as:

```
ssh-setup-lp  192.168.15.35
```

and will check that the named SnIP actually exists first and then create and transfer keys as necessary. It does require 2 destination SnIP password entries since the keys are not located there yet.

The second command script is the actual execution script named “**lexec**” short for Link Partner Execution. A desired command to be executed on the current Link Partner is written as lexec followed by the command with options in quotes following. For example to set the modulator data rate to 512 kbps on the link partner the command would be:

```
lexec  “m500ctl 0 mdr 512000”
```

and if you wanted to execute the same command on any other SnIP that did already have the proper keys previously sent to it, in this example 192.168.15.201, the lexec command would have the “**-p**” option flag and desired IP Address added as in”

```
lexec  -p 192.168.15.201 “m500ctl 0 mdr 512000”
```

The lexec command is not limited to modem control. It can use any Linux command just as if you were currently logged into a terminal session on the remote or Link Partner SnIP.

These commands can be added to existing SnIPs by updating to the latest software version greater than 0.6.11 or adding just those commands using rsync or WinSCP.

2.3 Format 3 - From a Windows PC to one or more SnIPs

On the Windows PC you must first download and install the program "**PuTTY**". This program is common and open source without any cost or fees. You should read the help which comes with the program which should answer most questions you may have. You will need to download the full set of files either as a zip file or the installer. Installation is not required as the zip files can simply be uncompressed into a suitable folder and then run by double clicking on them. The instructions be low assume the installer, but the same can be achieved by running the programs individually.

To generate an RSA key pair run the PuTTYgen program from Windows Start>>Programs>>PuTTY menu. Then use the “Generate” button. The new key pair can be saved and distributed according to the instruction in the PuTTY manual. The default key type of SSH-2 RSA is fine and should be used with 1024 bits. Press the “Generate” button to create the keys and then save both the public and private keys according to the instructions.

3.0 Removing SSH Keys

To remove a specific or all public keys from a SnIP you can simply edit the public keys list in its "~/.ssh/authorized_keys" file. Removal of the client private key or the private key file will require generating nd distributing a new set.

MAB
End of Document.