SATELLITE GROUND STATION SECURITY USING SSH TUNNELING

Kendall Mauldin
New Mexico State University

Dr. Stephen Horan
Faculty Advisor

ABSTRACT

As more satellite ground station systems use the Internet as a means of connectivity, the security of the ground stations and data transferred between stations becomes a growing concern. Possible solutions include software-level password authentication, link encryption, IP filtering, and several others. Many of these methods are being implemented in many different applications. SSH (Secure Shell) tunneling is one specific method that ensures a highly encrypted data link between computers on the Internet. It is used every day by individuals and organizations that want to ensure the security of the data they are transferring over the Internet. This paper describes the security requirements of a specific example of a ground station network, how SSH can be implemented into the existing system, software configuration, and operational testing of the revised ground network.

KEYWORDS

Space Telemetry, Satellite Networking, Scalable Networking, Ground Station, Nanosatellite, Security, Encryption, Secure Link

INTRODUCTION

The Internet has been used for many different purposes. One of these many uses is to connect satellite ground communications stations together. When multiple ground stations are used to communicate with a satellite or constellation of satellites, it is usually a requirement that the data from these stations can be collected to a central location. These communications are usually needed to be done in as close to real-time as possible. The most practical way to do this is to use the largest computer network in the world; the internet.

With the use of the Internet comes risk. The very nature of the internet is to connect as many computers to each other as possible. Human nature causes some computer users to be dishonest and destructive. Therefore it is very important to protect important computer systems that are connected to the internet, as well as the data they transfer between each other. A common way to do so is to use
encryption technology. Several methods of encryption exist, but due to its popularity and simplicity, the type of encryption that was chosen for this particular application is Secure Shell (SSH)™. Small satellites known as “nanosats” are more commonly being designed and built by small organizations and research institutions. With these nanosat projects come various designs of ground station networks to accompany them. It would be helpful to these organizations if the security of the ground networks is included as part of the ground station software package. To demonstrate one way this could be done, a specific nanosat project called 3 Corner Satellite will be used as an example.

The 3 Corner Satellite (3CS) project is part of the AFOSR/DARPA University Nanosatellite program. It is a joint effort among Arizona State University, University of Colorado at Boulder, and New Mexico State University. The project is building a constellation of three satellites that will demonstrate new types of nanosat technology such as stereo imaging of cloud and land formations, formation flying, and new types of command and control scheduling. [1] Because of the nature of the constellation and communication system configurations, a custom design for the ground station network is needed, and its security is critical.

This paper will discuss a means of constructing and ensuring the security of such a ground network using the LabVIEW™ version 6 software suite developed by National Instruments™, version 3.2.3 of the SSH network client, and version 3.2.3 of the SSH network server by SSH Communications Security™. The original components of the network are two different Multi-platform Virtual Instruments (VIs). [2] These VIs were designed to be compatible with many computing platforms and operating systems. In this paper, we will examine the design goals for this secure ground network, the server/client configuration, the encrypted data interaction between the server and client, selection of a secure data protocol for implementation, and the virtual instruments that were created to complete the design.

SECURE GROUND NETWORK DESIGN GOALS

There are many existing ground station network designs; however none met the specific design requirements for the 3CS project. Therefore, keeping in mind the security requirements placed upon this network, the following design goals will be followed during development and testing:

1) Data passed between stations shall be encrypted with no less than 128-bit encryption schemes.
2) The encryption key should not be public. Only private keys shall be used.
3) Initiation of an encryption connection shall be password protected. The password shall not pass through the network as unencrypted data.
4) Ground station software must be able to run without encryption layer in case of temporary encryption failure.
5) Encrypted link must be able to transfer data at a rate greater than 10 KB/sec.

---

1 SSH Secure Shell is a trademark of SSH Communications Security.
This project builds upon the baseline discussed in [2]; therefore these requirements are only for the security aspect of the networked ground stations. Other more general requirements apply, but are assumed to be part of an existing ground network software package.

**GENERAL IMPLEMENTATION**

The current ground station software that is being used for the 3CS project is arranged in a client/server configuration. The client resides at each remote ground station. The server resides at a central location, such as the mission operations center. The server communicates with each client simultaneously over unencrypted TCP/IP socket connections. Unencrypted TCP/IP connections can be easily intercepted, modified, or destroyed by someone with wrongful intentions on the internet. Therefore, the TCP/IP socket communications will be “tunneled” through an SSH encrypted data link.

SSH tunneling is a feature built into most SSH client and server software packages. Tunneling allows a local TCP/IP networking port on a user’s computer to be mapped to a port on a remote computer through an encrypted SSH link. An SSH session must first be started between the user’s computer and the SSH server on the remote computer. Once this session is established, any data that is sent to/from the user computer tunnel port will be sent to a specific port on the remote computer. This is commonly used for applications such as an e-mail client program obtaining a user’s e-mail from a POP e-mail server. If tunneling is supported by the POP mail server, a user can have their e-mail software try to connect to a local tunneling port instead of directly to the POP server. This provides a level of security to the transfer of the user’s e-mail. Figure 1 shows the overall layout of the 3CS implementation of this concept:

![Figure 1: Secure ground network organization](image-url)
TUNNELING CONFIGURATION

For use of this tunneling technique, the appropriate software must first be installed on both the client and server computers. The server computer must have a SSH compatible server package installed that supports SSH tunneling. The client computer must have a SSH client software package installed that also supports tunneling. The file transfer capability of many SSH server packages is not needed in this application, since the SSH connection itself provides the necessary tunneling feature.

Most SSH servers use the user names and passwords that already exist on the operating system of that computer. It is important to note that any user who has an account on that computer will have access to the SSH server as well.

The client software must then be configured to map a specific local port to a port on the server computer. This is usually done in the main configuration of the software. An example of how to do this in SSH is shown in Figure 2.

Figure 2: Tunneling setup in SSH software
This figure shows an example configuration where the SSH client listens to port 1002 on the local user’s computer. Any attempted connections to that port will be redirected to the server computer’s port 1000. If the server computer has some software that is listening to port 1000, a connection will be made between that software and the local user over an SSH link. Multiple tunnels can generally be set up by adding more to the outgoing tunnel list. Each tunnel only supports the redirection of one local port to one remote server port.

There are several different encryption schemes that most SSH clients and servers support. For this particular application the “Twofish”² 256 bit encryption algorithm was selected because it is inherently more secure than 128 bit encryption schemes. The server must also support this algorithm for successful communications. Some older SSH servers do not support this level of encryption.

SOFTWARE COLLABORATION

Once the tunneling is set up and functioning properly, the software running on the server computer must be configured to connect to the local SSH tunneling port instead of a normal TCP/IP connection to the destination remote port. In our specific example, the LabVIEW VI programs were modified to support the following connection scenarios:

- Direct TCP/IP socket connection to remote server
  a. This is necessary in the case that the SSH system is having problems or is not useable for some reason. It will allow temporary insecure connectivity to the remote server.
- Local connection to a SSH tunnel-mapped port.
  a. This mode is used during normal operations.

The client VI was modified to include and accept these two different modes of connectivity. These modes are selectable with a toggle button on the front panel, as shown in Figure 3. Other features were also added to monitor the status of the TCP/IP links.

---
² Twofish was developed by Counterpane Labs™.
The server VI (shown in Figure 4) was also modified to be able to initiate a TCP/IP connection to the client using both of these modes. In the SSH mode, the server connects to a specific local port that is mapped to a particular remote client computer through the SSH software configuration. In non-SSH mode, the server connects directly to the remote port on the client VI computer. Because of this, the remote address of each client computer must be entered into the server VI software, as well as the SSH software. The address in these two parallel software packages must agree for each client. This brings up a unique separation of connection responsibility. In the SSH mode, the original server VI is not responsible for making the actual remote connection. This means that the connection can be maintained out of the main processing loop. Therefore, a certain amount of processing load is effectively removed from the main loop.
TESTING AND VALIDATION

The ground station VI’s have been extensively tested in their original insecure configuration in the first three phases of the 3CS team effort. Since then, the project has passed through two additional phases to attempt to increase the security of the system. The next phase explored various means of increasing the security of the ground network system. The concept of designing a custom security protocol was explored. This idea was dropped because of the complexity that it added to the server and client VIs. The search was then directed towards a third-party software package that could provide the necessary security. Several different packages were explored, but most proved to be very cumbersome to set up and utilize. Finally, SSH was selected due to its reliability and easy configuration. SSH also seemed to support many encryption algorithms that other packages did not.
In the last phase, the cooperation between the existing LabVIEW VIs and the SSH software packages was ironed out. A list of modifications to the VIs that were needed was created, and these changes were performed. Finally, the entire collaborative combination of software was tested in a communications lab at New Mexico State University. A basic connectivity test was performed on the local LAN in the lab. After this successful test, a more long range test was performed. In this test, a computer in the lab acted as a client with actual radio hardware attached to it. Sample digital data was being collected from a local amateur radio packet network. This data was transferred to an off-site location in Las Cruces, NM to a computer that was running the server VI software.

During these phases, a few challenges were encountered. First, if there is a network firewall in place, some negotiation with a network administrator may be necessary to allow a direct connection to a machine behind this firewall. More testing will be necessary to assess the difficulty of dealing with the various different kinds of firewalls. Another aspect that will need additional study is the case where packets are lost across the network. If there is any packet loss, it is unknown how well the encryption algorithms will recover, and how quickly the recovery occurs.

Future phases for this project will also include testing the ruggedness of the encryption algorithms by sending the data over wireless networks, heavily loaded networks, and over long distances (such as Las Cruces to the east coast).

CONCLUSION

In a world where the internet is increasingly used as a form of connecting systems together, security becomes a serious matter. Millions of people have access to the internet. There are a few safeguards that help ensure the data that a particular user is sending across the net is not being monitored or modified. However, there are very few that come close to guaranteeing that your data will travel from point A to point B without modification or compromise. Satellite ground networks definitely fall into the category of systems that should not be compromised by an unauthorized user. This could reek havoc on the satellites they communicate with in orbit. For this reason, the 3CS project has implemented the SSH encrypted link protocol to attempt to increase the security of its ground network significantly. In this specific project, an off-the-shelf solution was used to reduce the man hours and funding required. The 3CS team hopes that this example of an inexpensive increase in information security will help other nanosat or similar programs that are in search of such a solution.

ACKNOWLEDGEMENTS

This work was supported under NASA Grant NAG 5-9323 to New Mexico State University from Goddard Space Flight Center and AF Grant F49620-99-01-0194 from the Air Force Office of Sponsored Research.
REFERENCES
