Global Information Assurance Certification Paper

GIAC Certification

Track 1- GIAC Security Essentials (GSEC)

Practical Assignment

Version 1.4b

Option 2

Secure File Transfer with SSH2

By Renato Lozano

June 26, 2003

Introduction to Case Study

I work for a financial institution which core business is payment transaction processing. They have launched a product that acts as an Internet Payment Gateway where merchants on the Internet will be able to process transactions in real time from their e-commerce websites. Alternatively, Internet merchants also have the capability of gathering transaction information and sending it as a batch file in XML format to our Internet Payment Gateway at the end of the day.

The topic of this case study will focus on how to transfer files that contain sensitive information securely over an untrusted network such as the Internet. The protocol that will be discussed on this case study to address the transfer of files securely is SSH2. I will be discussing various implementations of the SSH protocol that were evaluated and the challenges that I encountered. I will also be discussing the configuration of the SSH2 server to meet the needs of the security requirements that the project had At the end of this case study, the sftp functionality in the SSH2 protocol proved to be a safe method for secure file transfer for our Internet Payment Gateway.

Before Section of Case Study

Traditional ways of file transfers

FTP (File transfer protocol) is probably the most popular protocol for transferring files on the Internet. Internet websites usually provide software downloads for their products through an FTP server.

The problem with the FTP protocol

One of the problems with the FTP protocol is that clients connecting to the server send their authentication credentials in clear text. Also, the data that is being transferred through the wire is not encrypted. Clearly, this method of transferring files to our Internet Payment Gateway did not meet the security requirements of the project. The information being sent to us is very sensitive and the authentication was done in clear text.

Anybody sniffing the network could find out what the username and password of the merchant account is, and logon to their account. Once inside, they could gain access to confidential financial information and even change the contents of the financial information therefore violating the integrity of the data.

Security requirements for file transfer

The nature of the data that needed to get transferred from our merchants to our gateway was very sensitive and confidential. Information such as credit card numbers, total amounts of purchases and merchant information needed to get transferred to our payment gateway in a secure way. Here are the requirements from a security standpoint for the transfer of this information from our merchant to us:

- Authentication is required to be encrypted.
- Data being transferred over the Internet must be encrypted.
- The integrity of the data could not be compromised. Data could not be modified in transfer to our Internet Payment Gateway.

When trying to come up with a solution for a way to transfer files securely, I had to keep the above requirements in mind. I come from a background of System Administration. Traditionally, in Unix/Linux systems, the way to transfer files securely between host is by using the SSH protocol. More specifically, the scp command (Secure Copy) is used to transfer files. I thought that scp could be the solution for transferring files based on the above requirements.

During Section of Case Study

Environment

Our Internet Payment Gateway was fully developed using open source tools running on Red Hat Linux.

Architecture

The solution that needed to be implemented for the file transfer to our Internet Payment Gateway was based on the client / server model. Our merchants would require an SSH client and they would connect to our SSH server on our end.

Testing Phase

Based on the architecture described above, a need existed to set up an SSH server to receive incoming files from our merchants. Red Hat Linux comes with a SSH server installed by default. OpenSSH (http://www.openssh.org) is the free implementation of the SSH protocol and is included with Red Hat and other Linux distributions.

Over the course of time, new vulnerabilities are disclosed and software needs to get patched to address these vulnerabilities. It is good practice to keep up to date with all security patches. Before opening connections to the SSH server I had to make sure that I was running the latest version of OpenSSH.

Red Hat Linux uses rpm as the software packaging tool. Here are some of the commands used to determine the current version of OpenSSH and how to upgrade to a higher version

- To determine the current version of OpenSSH running on the system, use rpm –q openssh –l
- To upgrade to a higher version of OpenSSH, use rpm –Fvh

Once I had upgraded the OpenSSH rpm to the latest version available at http://updates.redhat.com, I was ready to start testing transferring files with the scp command. It was then where I encountered my first challenge.

First Challenge

I overlooked the fact that most of the clients connecting to our Internet Payment Gateway were running Windows Operating Systems. Linux and Unix users would be able to transfer files using scp from their hosts. The command scp

does not exist in Windows OS and therefore, I encountered my first stumbling block.

Hoping to solve this problem, I started doing some research on the Internet and soon found a solution to my problem. I found an scp client for Windows that will allow merchants to connect to the SSH server and transfer files. The name of the client is WinSCP (http://winscp.vse.cz/eng/). This client would allow merchants to transfer files from their Windows hosts to our SSH server.

I downloaded and installed WinSCP to test its functionality and I was able to connect to my SSH server. I tried transferring a couple of files and it worked fine. I thought to myself that I had a solution for the secure transfer of files since I had met all the security requirements mentioned in the requirements section above.

After further testing, I realized that I was able to browse the file system of the SSH server and this is where I encountered my second challenge.

Second Challenge

I soon figured out that the reason why I was able to browse the file system in the SSH server was because the account that I was using to connect to the SSH server had a normal shell associated with it. This meant that all the clients that would connect to the SSH server would have shell access to the SSH server opening up potential problems on the SSH server. Merchants could log on to the SSH server and execute scripts. In the worst-case scenario, a merchant could install a root kit in their home directory, eventually gaining control of the whole SSH server. Also, clients could see the home directories of other merchants disclosing their account names.

Clearly, the merchants had more access to the server than just to transfer files to our Internet Payment Gateway.

I needed a way to enhance the security of the SSH server by only providing access to transfer files. This brought me to the third challenge that I encountered during the testing phase of this project.

Third Challenge

The needs for this project was only to allow file transfers from the client to the server and not provide additional shell access through the SSH server to all merchants doing business with us. The ideal scenario would be for a merchant to only have access to his/her own home directory and only allow access to perform file transfer commands. To accomplish the ladder, a user would have to be restricted (chrooted) to their own home directory and only be able to perform file transfer commands.

After conducting more research on the Internet, I stumbled across a commercial implementation of the SSH protocol with chrooting capabilities and restrictive sftp shell. The sftp command is part of the SSH2 protocol. This command is also available from the free implementation of the SSH protocol, OpenSSH but they do not support the chrooting of accounts with restricted access just to transfer files. As part of the research that I conducted for coming up with a product that met the security requirements of the project, I found a site that had a patch for allowing OpenSSH to chroot user accounts (http://chrootssh.sourceforge.net/). Although this could have been very useful, it did not restrict users only to transfer files.

The Solution

F-secure SSH Unix server is a commercial implementation of the SSH protocol. After further research on their web site (http://www.f-secure.com), I downloaded a trial version for testing purposes to see if the product met the requirements for my project.

It is worth mentioning that in order to have support for the chrooting capabilities that this product offer, the installation has to be made from source since the sftp chrooting feature is not available in binary form as an rpm. In the next section of this case study, I will be discussing the set up and configuration of F-secure SSH Unix server to support the restrictive sftp shell and chrooting clients to their own home directory.

Set up of F-secure SSH Unix Server

To start the installation of the F-secure SSH server I downloaded a trial version of the product from http://www.f-secure.com. It is worth mentioning that I had to contact the sales department so they can grant me a 30day evaluation license of the product. Other SSH products for Windows were available as a download. Here are the steps needed to install the f-secure ssh server to meet the requirements of a restricted shell and chrooting users

- Cd to /usr/local and uncompress the source code by issuing gzip –d f-secure-ssh-x.y.z.tar.gz | tar –xvf - . This command will uncompress and untar the source code inside /usr/local.

- After the file got uncompressed and untar, cd to f-secure-ssh-x.y.z and type ./configure - - enable-static - - with-libwrap The - - enable static option will allow you to chroot users on their own home directory so they won't be able to browse outside their assigned space in the file system. The - - with-libwrap[=PATH] option enables the F-secure SSH server to

have support for tcp wrappers. This allows the F-secure SSH server to allow or deny connections based on IP Address. The files /etc/host.allow and /etc/host.deny can then be edited to add an extra layer of security for the F-secure SSH server.

- After configuring the installation with the above parameters, type ./make and then ./make install . This will install the software on the local system.

In order to start and stop the F-secure SSH server, it needs to be added to the start-up scripts inside /etc/init.d . The source distribution of the SSH server comes with a sample start-up script inside the /usr/local/ f-secure-ssh-x.y.z/startup . Inside this directory, start-up scripts exist for AIX, HPUX, Linux, Solaris and NETBSD. Go inside the Linux directory and then change directory to RedHat. Simply copy the sshd2 script from that location to the /etc/init.d directory by typing cp sshd2 /etc/init.d

Once the start-up script is copied to the right location, it needs to be added to the different run-level directories. The script that is provided with the software supports the chkconfig command. This command updates and queries run level information for system services. In order to add the sshd2 start up script to the run levels issue the following command:

- chkconfig - -add sshd2 .
- To make sure it has been added to the startup configuration issue the command chkconfig - -list sshd2. This will display the different run levels where the SSH server is either on or off.

Setting up merchants for sftp file transfers

In this section I will be discussing how to set up user accounts with a restricted shell only to allow sftp commands and restricting users to their own home directory (chrooting).

The first step is to add the user to the local system specifying a restricted shell that F-secure SSH server provides with their product. Follow the above commands to achieve this:

- As root, type the following command to create the user on the local system: useradd –s /bin/ssh-dummy-shell testuser This command will create the user “testuser” . The –s flag assigns /bin/ssh-dummy-shell to the testuser. This is a very restrictive shell that only allows the executing of sftp account for this account. If the –s flag was not specified in the account creation, then the testuser would have a valid shell like /bin/bash and could have more access to the server than just sftp files to the

account. This is very important since the requirement of the project was to only allow file transfer capabilities in a secure way.

The second step is to make the restrictive shell available for the testuser in his/her home directory. F-secure SSH server comes with a helper script to help you copy the right files to the right locations. The name of the script is ssh-chrootmgr. Type the following command to create the files need to create the chroot:

- As root, type ssh-chrootmgr testuser. This command will create a bin directory inside the testuser home directory with the necessary files to only be able only to sftp. This reduces the possibility of a user browsing the local file system and being able to see configuration files or other information in the local system.

The third step is to change the configuration of the F-secure SSH server to add the user to the list of accounts that will be chrooted. The configuration file for the F-secure SSH server can be found in the /etc/ssh2 directory. The file name that needs to be edited is sshd2_config.

- As root, open /etc/ssh2/sshd2_config with your favourite text editor and add testuser to the ChRootUsers Directive.
- After editing the file, start the SSH server by issuing the following command: /etc/init.d/sshd2 start

Testing the restricted set-up with different clients

Merchants connecting to our Internet Payment Gateway require a client with sftp support so they can connect to our SSH server. After some research on the Internet I have tested the following ssh clients successfully with the set-up of the F-secure SSH server:

Commercial SSH clients:

- F-secure SSH client for Windows and Unix (http://www.f-secure.com/products/ssh/index.shtml)
- SSH Communications Security, SSH Secure Shell for workstations (http://www.ssh.com/products/security/secureshellwks/) Windows only.
- Vandyke Software SecureFx for Windows. (http://www.vandyke.com/products/securefx/index.html)

Free SSH clients:

- PSFTP (Putty SFTP) (http://www.chiark.greenend.org.uk/~sgtatham/putty/)
- FileZilla (http://filezilla.sourceforge.net/)
- OpenSSH sftp client (http://www.openbsd.org/cgi-bin/man.cgi?query=sftp&sektion=1)

Adding more security to the authentication process for sftp

One of the primary authentication methods that the SSH protocol support is username and password. Traditional username and password authentication is not as secure as public/private key authentication which is also supported by the SSH protocol. In comparison with the username and password authentication, public/private key authentication requires two secret components, which are the private key and the passphrase of the private key. This is a two-factor authentication and is considered to be more secure. In a username and password scenario, the user usually generates a password that is easy to remember and weak in characteristics. These weak passwords are easily crackable by a dictionary attack in most cases.

In order to enhance the authentication process for accounts hosted on our Internet Payment Gateway, public/private key authentication was chosen as being the preferred method of authentication for sftp transfers from merchants to us. All commercial SSH clients that were tested support public/private key authentication for sftp file transfers. From the free SSH clients that were tested, only Filezilla did not support public/private key authentication for sftp transfers.

In the next section of this case study I will be discussing how to set up private/public key authentication as means of authentication for sftp transfers from merchants to our Internet Payment Gateway.

Public/Private Key authentication

In order to implement public/private key authentication to our Internet Gateway, our merchants would have to create a key pair on their end (client) and provide the public key to us. There are numerous ways of creating a key pair depending on the SSH client. Usually Windows commercial SSH clients come with wizards that guide users on how to achieve the key pair creation. Note that users should always passphrase protect their private keys. A private key that is not passphrase protected could fall on the wrong hands and could then be used for the authentication process. For Unix/Linux SSH clients, there is a utility called ssh-keygen that will generate the key pair with the following command:

- Ssh-keygen –t rsa/dsa This command will generate the key pair for public/private authentication. Depending on the arguments after –t , the program will generate either an RSA key pair or DSA key pair.

Once the merchant has generated the key pair, they would provide us with their public key. The public key is then place in a directory called .ssh2 on their home directory of the SSH server. A file by the name of authorization also needs to be created inside the .ssh2 directory and should list the public key that will be used for private/public key authentication. The contents of the authorization file should look like the following:

- Key publickeyname.pub

Once the public key and the authorization file exist in the .ssh2, the F-secure SSH server needs to have the following directive in its configuration file to allow public/private key authentication:

- AllowedAuthentications publickey

When merchants will try to connect using public/private key authentication, they will be prompted to enter their private key passphrase in order to connect to the F-secure SSH server.

In some scenarios, some of our merchants would like to automate the authentication method using private/public key authentication. One highly insecure way of achieving this kind of automation is with a private key that does not contain a password. This is highly not recommended since private keys should always be passphrase protected. A more secure method of automating public/private key authentication would be using a program that is part of the SSH2 protocol called ssh-agent and ssh-add. Users can load their private keys to the ssh-agent with the ssh-add command and the ssh-agent will provide the passphrase of the private keys at time of authentication against the SSH server. This is the preferred method of automating the authentication process using private/public key authentication.

After Section of Case Study

Using the sftp command, which is part of the SSH protocol, prove to be a good solution since it addressed the following areas to provide a secure method of communication for file transfers:

- The data that is being transferred from our merchants to the Internet Payment Gateway is encrypted addressing the issue of privacy. Since all data transfers are encrypted when using the SSH protocol, the threat of sniffing the network is eliminated since the data is encrypted and will be of no use for a possible attacker.

- Data that is being transferred when doing the upload to our Internet Payment Gateway can not be modified since the SSH protocol checks for the integrity of the data. This is very important since the data being sent contains financial information and needs to be accurate.

- SSH provides strong methods of authentication using public/private keys.

In conclusion, file transfers with sftp using the SSH protocol is a way of safely transfers files over insecure networks like the Internet. The case study discussed above was implemented into a production environment and merchants are currently transferring files securely to our Internet Payment Gateway. The SSH protocol has other capabilities like being able to tunnel TCP/IP applications in order to make them more secure. This capability is not in the scope of this case study but it's worth mentioning since it is a very easy way to secure any TCP/IP application.

REFERENCES

Bibliography

SSH, The Secure Shell: The Definitive Guide, by Daniel J. Barret and Richard and Richard Silverman, 1st Edition January 2001, ISBN: 0-596-00011-1

Securing Linux Step-by-Step, by The SANS Institute, Version 1.0, ISBN: 0-9672992-0-9

Online Resources

- OpenSSH http://www.openssh.org

- RedHat Updates http://updates.redhat.com

- WinSCP http://winscp.vse.cz/eng/

- ChrootSSH http://chrootssh.sourceforge.net/).

- F-secure http://www.f-secure.com

- SSH Communications http://www.ssh.com

- Vandyke Software http://www.vandyke.com

- PSFTP (Putty SFTP) http://www.chiark.greenend.org.uk/~sgtatham/putty/

- FileZilla http://filezilla.sourceforge.net/

Last Updated: May 23rd, 2017

Upcoming Training

SANS Stockholm 2017	Stockholm, Sweden	May 29, 2017 - Jun 03, 2017	Live Event
Security Operations Center Summit & Training	Washington, DC	Jun 05, 2017 - Jun 12, 2017	Live Event
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
Community SANS Ottawa SEC401	Ottawa, ON	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Rocky Mountain 2017 - SEC401: Security Essentials Bootcamp Style	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
Community SANS Portland SEC401	Portland, OR	Jun 12, 2017 - Jun 17, 2017	Community SANS
SANS Secure Europe 2017	Amsterdam, Netherlands	Jun 12, 2017 - Jun 20, 2017	Live Event
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
SANS Munich Summer 2017	Munich, Germany	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Phoenix SEC401	Phoenix, AZ	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Minneapolis SEC401	Minneapolis, MN	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
Mentor Session - SEC401	Macon, GA	Jul 12, 2017 - Aug 23, 2017	Mentor
Mentor Session - SEC401	Ventura, CA	Jul 12, 2017 - Sep 13, 2017	Mentor
Community SANS Atlanta SEC401	Atlanta, GA	Jul 17, 2017 - Jul 22, 2017	Community SANS
Community SANS Colorado Springs SEC401	Colorado Springs, CO	Jul 17, 2017 - Jul 22, 2017	Community SANS
SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
Community SANS Charleston SEC401	Charleston, SC	Jul 24, 2017 - Jul 29, 2017	Community SANS
SANSFIRE 2017 - SEC401: Security Essentials Bootcamp Style	Washington, DC	Jul 24, 2017 - Jul 29, 2017	vLive
Community SANS Fort Lauderdale SEC401	Fort Lauderdale, FL	Jul 31, 2017 - Aug 05, 2017	Community SANS
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event