SoftNAS Application Guide:

In-Flight Encryption

12/7/2015
SOFTNAS LLC

SoftNAS Application Guide:

In-Flight Encryption

Contents

Introduction to In-Flight Data Encryption

In-Flight Data encryption involves encrypting the data stream at one point and decrypting it at another point. For example, if you replicate data across two data centers and want to ensure confidentiality of this exchange, you would use In-Flight Data encryption to encrypt the data stream as it leaves the primary data center then decrypt it at the other end of the cable at the secondary data center. Since the data exchange is very brief, the keys used to encrypt the frames or packets are no longer needed after the data is decrypted at the other end so they are discarded - no need to manage these keys.

This is in contrast to data-at-rest encryption solutions, where the data might sit a long time before retrieval. For data-at rest, the keys used to encrypt the data need to be managed in order to allow later retrieval of the data in question. The key management system usually comes in the form of an appliance but it could also be a software application running on a server.

SoftNAS supports two methods of in-flight data encryption:

- CIFS Encryption In-Flight
- Tunneling NFS through SSH

CIFS Encryption In-Flight

SoftNAS has built in support for Windows 2012, including the inherent In-flight Data Encryption improvements offered by the now standard SMB 3 protocol in Windows Server 2012. Any CIFS enabled SoftNAS share can benefit from these data-in-flight encryption improvements.

Windows 2012 SMB 3 Improvements

SMB 3 in Windows Server 2012 adds the capability to make data transfers secure by encrypting data in-flight, to protect against tampering and eavesdropping attacks. The biggest benefit of using SMB Encryption over more general solutions (such as IPSec) is that there are no deployment requirements or costs beyond changing the SMB Server settings.

SMB 3 uses the AES-CCM encryption algorithm, which also provides data integrity validation (signing). SMB 3 uses a newer algorithm for signing – AES-CMAC instead of the HMAC-SHA256 used by SMB 2. Both AES-CCM and AES-CMAC can be dramatically accelerated on most modern CPUs with AES instruction support.

SMB 3 also includes a new capability to detect “man in the middle” attempts to downgrade the SMB 2/3 protocol “dialect” or capabilities that the client and server negotiate. When either client or server detects such manipulation, the connection will be disconnected and Event ID 1005 will be logged in the Microsoft-Windows-SmbServer/Operational event log. Secure Negotiate cannot detect/prevent downgrades from SMB 2 / 3 to SMB 1, and for this reason we strongly encourage users to disable the SMB 1 server if possible.

Windows SMB 3 Deployment Considerations

If deploying an SMB 3 enabled share or server, only SMB 3 clients will be allowed to access the affected shares. The reason for this restriction is to ensure that the administrator's intent of safeguarding the data is maintained for all accesses. However there might be situations (for example, a transition period where mixed client OS versions will be in use) where an admin may want to allow unencrypted access for clients not supporting SMB 3. To enable that scenario, run the following powershell command:

```
Set-SmbServerConfiguration –RejectUnencryptedAccess $false
```

The Secure Negotiate capability described in Windows 2012 SMB 3 Improvements does prevent a "man in the middle" from downgrading a connection from SMB 3 to SMB 2 (which would use unencrypted access); however it does not prevent downgrades to SMB 1 that would also result in unencrypted access.

For this reason, in order to guarantee that SMB 3 capable clients will always use encryption to access encrypted shares, the SMB 1 server must be disabled.

If the `–RejectUnencryptedAccess` setting is left at its default setting of `$true` then there is no concern, because only encryption capable SMB 3 clients will be allowed to access the shares (SMB1 clients will also be rejected).

Making SoftNAS CIFS shares SMB 3 Compatible

In order for the SoftNAS instance to support in-flight data encryption, the following considerations must be met:

1) The SoftNAS VM must be added in Active Directory.

2) The file **/etc/samba/smb.conf** must be edited as follows:

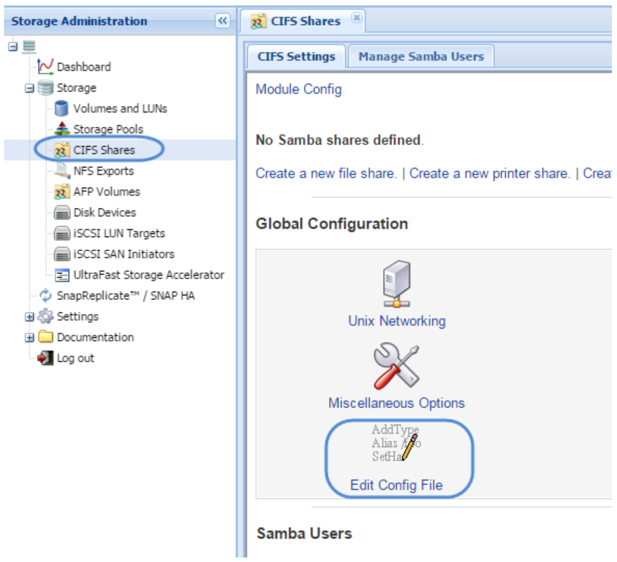
Under global, add the following:

smb encrypt = mandatory

server signing = mandatory

Service smb restart

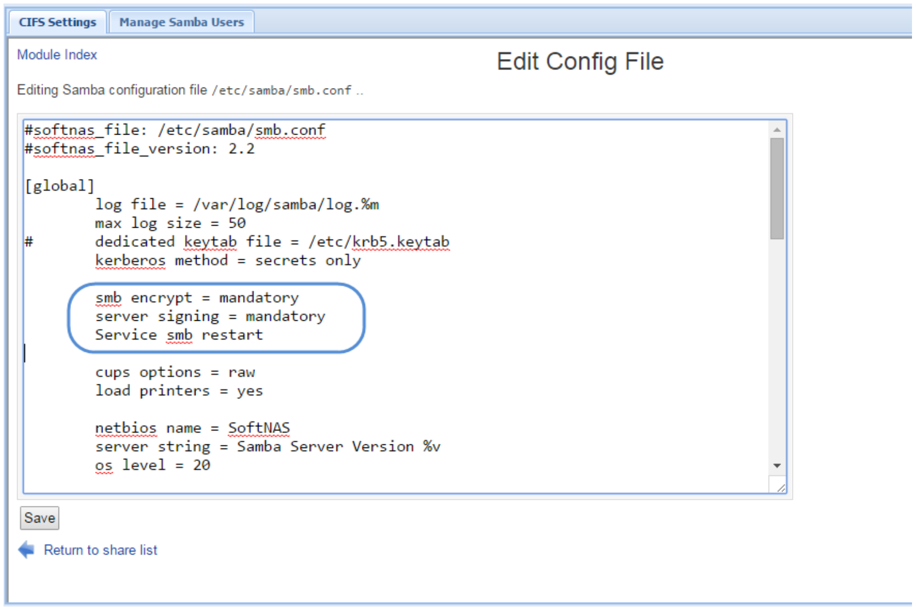
To make the above change is simple in SoftNAS. Go to **CIFS Shares** in the **Storage Administration** panel, and select **Edit Config File**.



No Samba shares defined.

Create a new file share. | Create a new printer share. | Crea

Copy and paste the information presented above into the **Config File**.



Editing Samba configuration file /etc/samba/smb.conf ..

```
#softnas_file: /etc/samba/smb.conf
#softnas_file_version: 2.2

[global]
        log file = /var/log/samba/log.%m
        max log size = 50
#       dedicated keytab file = /etc/krb5.keytab
        kerberos method = secrets only

        smb encrypt = mandatory
        server signing = mandatory
        Service smb restart

        cups options = raw
        load printers = yes

        netbios name = SoftNAS
        server string = Samba Server Version %v
        os level = 20
```

Windows 2012 clients can now connect with the SMB 3 encrypted CIFS share.

Tunneling NFS Through SSH

NFS is not secure on its own. It is a storage solution, plain and simple. If your data is on a closed system, (no network access) in which you trust all the internal machines, and all the users with access to the system, this is not an issue. This is not possible in a cloud solution, where remote access is a part of the definition, and data is in constant motion between two or more locations.

With NFS alone, there are only two steps required to access the data:

1. Mount Access - Mount access is achieved by the client machine attempting to attach to the server. The security for this is provided by the */etc/exports* file. This file lists the names or IP addresses for machines that are allowed to access a share point. If the client's ip address matches one of the entries in the access list then it will be allowed to mount. If this access list is in any way compromised, access to your data is simple.
2. File Access – Once mount access is granted, normal file system access controls are used to govern access. If the intruder has knowledge of user credentials, he or she will have immediate access to any files said user has access to, whether by individual permissions or group. Moreover, if someone gains super user status on the client machine, they can –su username, and become any user. NFS will not know the difference.

As you can see, it is important to prevent unauthorized access to the data at all points of access. Access to host locations are typically fairly regulated by IP restrictions, closed networks with only VPN or jumpbox access, etc. In addition, there are hosts of data-at-rest 3rd party encryption solutions. However, in-flight data may still be a risk for your environment. When you back up your data between your office and cloud storage, or if accessing remote files from another office location, your data may travel vast distances without encryption, providing a would-be adversary ample opportunity for access. The answer, of course, is In-Flight Data encryption.

NFS does not natively protect (encrypt) data in transit. However, tunneling NFS through SSH will provide this protection for our NFS clients.

Configuration of Tunneling NFS through SSH

In order to tunnel NFS through ssh, you will need to use ssh to forward ports. For example, **ssh** can tell the server to forward to any port on any machine from a port on the client.

Let’s assume that we have created a server with the IP *20.20.0.20*, and that we have pinned mountd to port 32767 using the argument **-p** 32767. Then, on the client, we'll type:

```
# ssh root@20.20.0.20 -L 250:localhost:2049  -f sleep 60m

# ssh root@20.20.0.20 -L 251:localhost:32767 -f sleep 60m
```

With the first command in play, the client will relay any request directed at the client's port 250, first through **sshd** on the server, and then on to port 2049 on the server. The second command does a similar relay, from port 251 on the client, through **sshd** on the server, to port 32767 on the server.

Localhost, of course, refers to the server. In other words, forwarding is to the server itself. The *localhost* is relative to the server; that is, the forwarding will be done to the server itself. The port could otherwise have been made to forward to any other machine, and the requests would look to the outside world as if they were coming from the server. NFSD on the server will treat these requests as if they are coming from the server itself.

Note: If you wish to bind to a port below 1024 on the client, the above command must be run as root on the client. You would need to bind to a lower port if exporting the file system with the default **secure** option.

The last part of the command, **`-f sleep 60m`**, prevents a shell from opening on the remote machine. Typically, when ssh is used, even with the –L option, a shell will open. In this case, however, we want the port forwarding to occur in the background, so we get the shell on the client back. The –f sleep 60m command will run in the background on the server, telling it to sleep for 60 minutes. This causes the port to be forwarded for 60 minutes until it gets a connection. The port will continue to be forwarded either until the hour is up, or until the connection dies, whichever happens last. This command can be placed in your startup scripts, after your network is started.

Once the above is complete, we have to mount the filesystem on the client. However, we need to specify a different port than the usual 2049 (because it is in use already). In order to do this, we enter the following in /etc/fstab telling the client to mount a file system on the localhost, specifying the different port:

```
localhost:/home /mnt/home nfs rw,hard,intr,port=250,mountport=251 0 0
```

Security Considerations

Remember, tunneling through **ssh** is an external protection measure only. Ordinary users who are able to log in locally are still a risk. Internal users with the right privileges can perform the exact same commands, and use ssh to forward a privileged port on their own machine (where they are legitimately root) to the same ports, 2049 and 32767 on the server. This means an ordinary user can mount the file system with the same rights as root on our client.

Keeping the above in mind, if you wish to use this method, there are two additional caveats:

- The connection travels from the client to the server via **sshd**; therefore you will have to leave port 22 (where **sshd** listens) open to your client on the firewall. However you do not need to leave the other ports, such as 2049 and 32767, open anymore.
- File locking will no longer work. It is not possible to ask **statd** or the locking manager to make requests to a particular port for a particular mount; therefore, any locking requests will cause **statd** to connect to **statd** on localhost, i.e., itself, and it will fail with an error. Any attempt to correct this would require a major rewrite of NFS.

Summary

In-Flight Encryption is important to securing any cloud environment, where data travels constantly back and forth between hosted environments and your local servers. There are numerous paid solutions that can bolster security by encrypting data both at-rest, and in-flight, but they can be costly. The above solutions offer free protection for your environment that are simple to implement, and provide more than adequate protection from outside threats.