2019

Poindexter Papers

How to Do File Transfer the Right Way

GO ANYWHERE®
A HelpSystems Solution

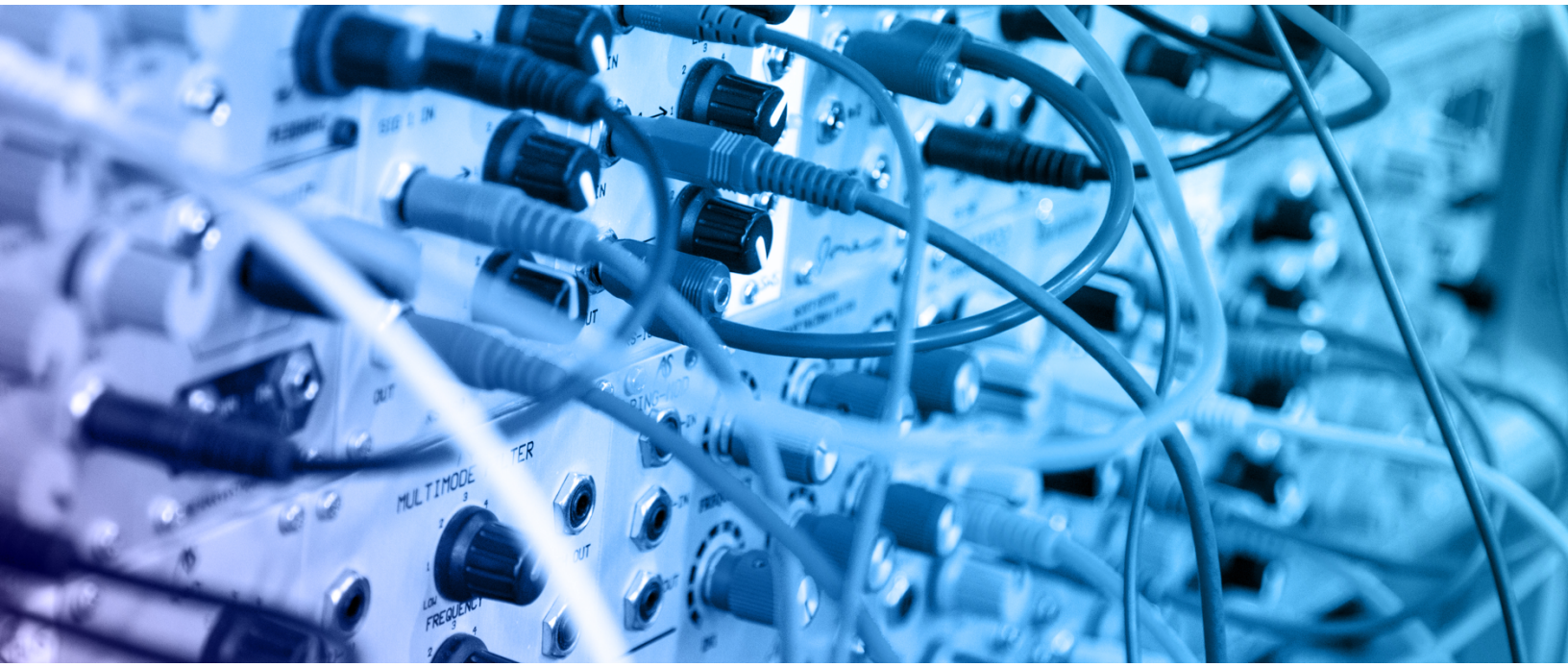
BY MIKE DIEHL

Table of Contents

Introduction

Being able to move information from one place to another is an important aspect of almost any organization. This information might be a list of products that were sold or offered for sale on-line. Or it could be information that requires tighter security, such as an employee or patient list with social security numbers, sensitive legal documents, or surveillance footage to be sent to law enforcement. It's hard to imagine an organization that doesn't produce or require information that needs to be shared with external organizations. One of the many challenges that a system administrator faces is how to move this information from where it is produced to where it is consumed. This information flow needs to be convenient, efficient, secure and reliable.

We're going to discuss some of the methods, in order of increasing sophistication, that organizations use to move data from one place to another.



Old-School and Un-Recommended

The first method that people came up with was to simply move files around manually using portable storage media. The storage media could then be mailed or hand-couriered to its destination. This method is obviously a low-tech, but intuitive, method, and the easiest to orchestrate. But it's not very efficient, even if the destination is in the office next door. This method is also fraught with error. People make mistakes. For example, a user might copy the wrong data, or get the media mixed up and send the right data to the wrong destination. Certainly, this method isn't going to comply with any rational data privacy regulations such as PCI, HIPAA, or the GDPR. Except in those rare cases where an air-gap exists between two networks, such as in a Classified Environment, this is almost certainly the wrong way to move files around.

The traditional method of moving files from one machine to another was to use FTP. While FTP does offer user authentication, it doesn't provide any end-to-end encryption, so it's not very secure. User names and passwords are sent from the client to the server in plain text and doesn't provide any error alerting capability. Basically, FTP meets our requirements for efficiency and convenience, but falls flat with respect to security and accountability.

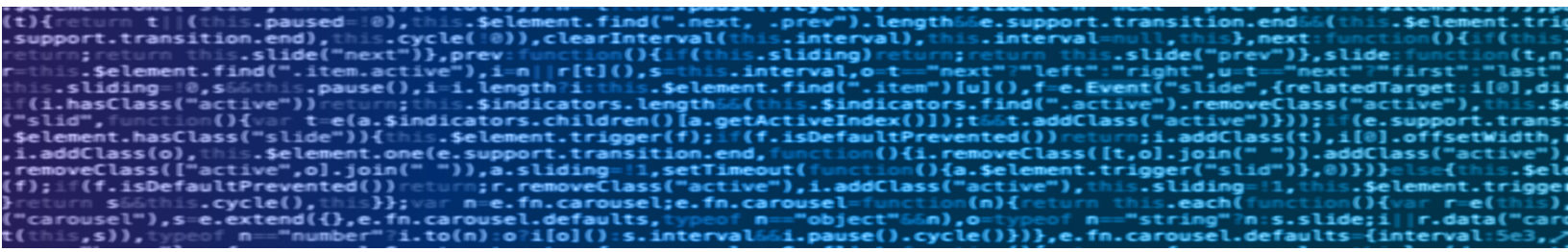
A new trend in file transfer is to use third-party cloud services such as Dropbox or Google Drive as a means of getting files from one place to another.

Since email is ubiquitous these days, one might be tempted to simply email files where they need to go. In some cases, such as sending an occasional report to a manager, this method might make sense. Email is efficient and extremely convenient. But we've all faced the difficulty of sending an attachment to someone and having it blocked by the destination server because the file was either too big, or had an extension that triggered a mail filter rule.

While some email systems do provide encryption features, most of them are cumbersome at best. So, the tendency is to simply email potentially sensitive documents in plain-text. We can certainly do better than this, but we first need to discuss a relatively new player in the file transfer arena.

A new trend in file transfer is to use third-party cloud services such as Dropbox or Google Drive as a means of getting files from one place to another. For individuals and small businesses that have absolutely no security or accountability requirements, this might make sense. But consider this: once the data has left your network, you have very little control over who accesses (or grants additional access to) your data. Even worse is that you don't have any direct influence over your cloud provider's security policy. At best, you have to be careful about what you put in the cloud. Remember, even a simple Profit and Loss report represents Intellectual Property and could even yield useful information to your competitors if this cloud service was ever breached.

So far, we've only discussed file transfer methods that, frankly, aren't recommended. The only reason we discussed them at all is that they are still in common use, and we'd like to discourage it. As we'll soon see, it doesn't take too much effort to configure and maintain a secure, efficient, and accountable file transfer infrastructure. We're going to discuss three other file transfer mechanisms: how they work, how to configure them, and later, how to automate them. Finally, we'll discuss how a Managed File Transfer (MFT) solution can enhance this infrastructure.



Newer, Better, Faster

In a nutshell, FTPS is to FTP what HTTPS is to HTTP. FTPS is simply FTP that uses a TLS encryption layer to secure the underlying communication. Once a connection is made, the same commands that you'd use with FTP work with FTPS. You can authenticate with user name and password, or you can use certificate-based authentication, which we'll discuss soon. However, the FTPS protocol does suffer from a couple difficulties. Since FTPS is simply FTP with TLS added, it also requires two network connections, one for control, and one for data transfers. Of course, some firewalls block this type of connection. Also, some FTP servers don't support the encrypted variant, and those that do may require additional configuration. If you are already using FTP and your server software supports FTPS, changing to FTPS may be the proverbial "no brainer."

The next two transfer protocols, SFTP (not to be confused with FTPS, though they often are) and SCP are both related to the SSH protocol. Even though they are both derived from the SSH protocol, they are used differently. The SFTP client works much like the standard FTP and FTPS client; this similarity in usage and the obvious name resemblance is why SFTP and FTPS are so often confused. The SFTP client does support a batch mode, where a user simply supplies a file that contains the commands needed to perform a given file transfer. This allows for convenient and potentially complex batch processing.

> Although scripting is free in most computing environments, and most network administrators know how to write scripts to automate business processes, scripts do have some drawbacks to be aware of...

The SCP protocol, on the other hand, is used much like the Unix cp (copy) command. The SCP command does support automatically preserving file ownership and permissions.

SCP also supports recursive file transfers. One interesting feature of SCP is that it allows for the transfer of files between two remote servers.

You can use either user name and password or a public key to authenticate with SCP and SFTP. The first step to creating and configuring a public key is to use the ssh-keygen command to create a public and private key pair:

```
ssh-keygen -b 2048 -t rsa
```

The output of this command resembles:

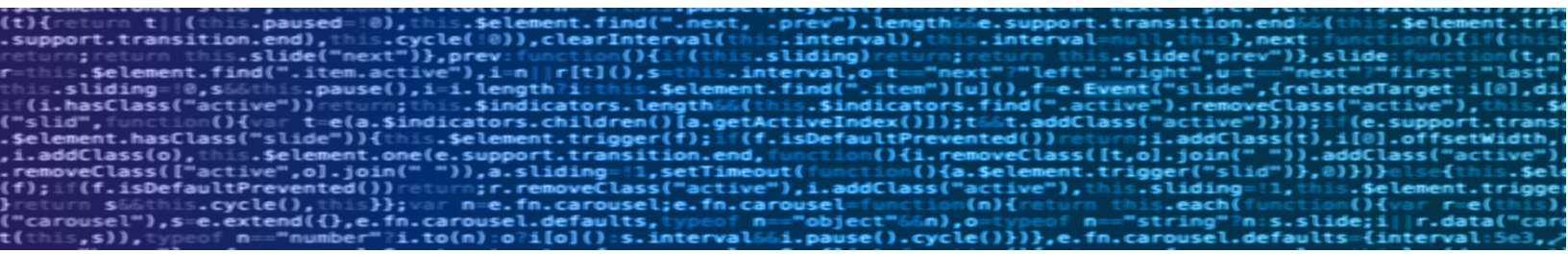
```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:VKdoHKLaLFujNrpeanDErL0O3a7m44UMX4SUYtQ2FWY user@example.com
The key's randomart image is:
+---[RSA 2048]----+
|..o..Eo . . .    |
|.o.=o. o + o     |
|.+o +   = .      |
|  +=   o         |
|.+o *   S        |
|o=oO .           |
|o.Xoo            |
| =*+             |
|=O=o.            |
+----[SHA256]-----+
```

There are a few things to notice here. First, we were asked to provide a name for the keys to be generated. If this is the first key pair that we are generating, it's usually recommended to simply take the default. If you don't provide a file name, ssh-keygen will create two files in the ~/.ssh/ directory, id_rsa and id_rsa.pub. The id_rsa file is your private key and should be protected from disclosure. The id_rsa.pub file is your public key. This is the key that you copy to remote destinations in order to connect to them using your private key as a credential, as we'll see shortly. On the other hand, if you had provided a filename, such as "mailserver," ssh-keygen would have created mailserver and mailserver.pub in the current directory.

The content of the public key file will look something like:

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDDgRFmuYp+7Tj/NZ5WQtJQRCJcx0Iz835PL3p
BU5XIj04Adt4E1UVBwqz4LRNyCmo6w0NaebTcaGq3wPl/9Uf9LF3SOG0M+HqPjASRcDRf
nCUDUFIhsCTseUT5o+82ySte7KsfBlt3/3Z+RYfviwnunNkXp8Kyl/oMvyBJ6zIaMBNEVXFYc
+6QN9idhUNtvV4jh2BXnOlsuRa5ivs8KNwcHwsed1veCcCbSTYnz3KRokKG6iifNHx3+zW
weNyrk1XBBl1ZjG6Ad7vmmgPCP11JA03XsvIhYieNxzqqRlt0XLZK516DLkytYPKrawhdXp5
29Aex1kjTpS/MAG54yEHn user@example.com

As you can see, it's mostly just a jumble of random characters.



Once the private and public keys have been generated, we simply add the client's public key to the end of the server's ~/.ssh/authorized_keys file. The easiest way to do this is to log into both accounts via ssh. Once logged into both servers, it's easy to print out the contents of the public key file on the client, copy it to your clipboard, edit the authorized_keys file on the server, and paste the clipboard to the end of the file. Since you are connected directly to the server via a secure connection, there aren't any security concerns to worry about. However, it is perfectly safe to email a public key to the system administrator of the remote systems so that they can install it on the appropriate servers. After all, the public key's sole purpose is to establish a secure session without having to present login credentials in plane-text. In order to protect against potential man-in-the-middle attacks, you should verify the key's fingerprint using a different method of communication, such as over the phone.

To display a key's fingerprint, simply have both parties issue this command at the shell prompt:

ssh-keygen -lf /home/user/.ssh/id_rsa

Both parties should use the path to wherever they stored the public key if it is different than /home/user/.ssh/id_rsa.

This command will display the key's fingerprint, which may look like:

2048 SHA256:VKdoHKLaLFujNrpeanDErL0O3a7m44UMX4SUYtQ2FWY
user@example.com (RSA)

If the public keys at both the client and the server have the same fingerprint, the keys were transmitted securely and the key is safe to use for secure communication.

Next, you will notice that we were prompted for an optional password. If we had provided one, that password would be used to encrypt the private key. In this case, that

password would have to be used to open the private key and thus, connect to the remote host. This is a nice feature to use with accounts that reside on laptops or other devices that aren't completely under the control of the system administrator. However, using a password won't allow you to automate connections; the password would have to be entered by hand or stored somewhere and that would defeat the purpose of having a password! Just press the enter key when prompted for a password.

It's important to remember that the ssh daemon has to be configured to allow for public key authentication. This configuration is easy, though. Simply edit the /etc/ssh/sshd_config file and make 2 changes. First, you need to find the PubkeyAuthentication entry and change its value to "yes." Then, it's also a good idea to turn password authentication off so that the only way to gain access to the server is to have the appropriate private key. So, the changes you would make would look like:

PasswordAuthentication no
PubkeyAuthentication yes

Once the configuration is complete, the ssh daemon needs to be restarted.
Sometimes, a connection to a particular host requires special configuration options. This happens when the server's ssh daemon isn't listening on port 22, for example. Openssh, and all of the file transfer programs that we've discussed that depend upon it, allow for connection-specific configuration. This configuration is accomplished by adding an entry to the /home/user/.ssh/config file. Such an entry might resemble this:

```
Host example
        Hostname 10.0.1.1
        User admin
        Port 2022
        IdentityFile /home/user/.ssh/example
```

Here we see an entry for a computer named “example,” The IP address of 10.0.1.1 is specified for the server. In this case, we also see that the server’s sshd process is listening on port 2022 instead of the standard port 22. Finally, we have declared that a connection to this server will use a specific key file, as indicated by the IdentityFile entry. This configuration needs to be performed for each client and server that needs to communicate with each other. Once this configuration is complete, transferring files from one computer to the other is easy!

For example, to use scp to copy a file (report.txt in this case) from the local machine to the machine named “example,” from above, you would issue a command that resembles:

```
scp ./report.txt example:/home/admin/
```

If you were transferring the file to a machine that did not have an entry in the ~/.ssh/config file but did have a DNS entry, you would issue a command that resembles:
scp ./report.txt admin@example.com:/home/admin

Here, we’ve specified the host name to connect to and the user name to use to authenticate as. In both cases, the destination file path is specified after the “:” character.

As we said earlier, using sftp is very similar to using the old-style ftp program. Such an interaction might look like this:

```
sftp admin@example
Connected to 10.0.1.1.
sftp> cd /home/admin
sftp> put report.txt
Uploading report.txt to /home/admin/report.txt
100%  399    32.7KB/s   00:00
sftp> quit
```

All of this interaction can be automated by putting the (s)ftp commands in a batch file and using the -b flag to tell the sftp client to execute the commands in the batch file: sftp -b batchfile.txt admin@example

Managed File Transfer (MFT) solutions solve the problems that we've outlined here, while adding unique benefits as well.

In order to use the FTPS protocol to transfer files, you need to install the lftp client. The lftp program is pretty flexible and can transfer files using any of FTP, FTPS, HTTP, HTTPS, HFTP, FISH, SFTP and BitTorrent protocols! Using lftp is much like using sftp, so we won't go into too much detail here.

Now that we've discussed how to configure and use each of these technologies in simple cases, it's important to understand that almost no one does it that way. Typically, a system administrator would combine these commands, with others, into a script that performs a particular business operation. Such a script would then be scheduled using a system scheduler such as cron. It's easy to imagine a script that transfers purchase requests to a vendor for processing, or another one that downloads a subscribed users list from one server and distributes that list to several other servers in order to create an account for each user on one server, subscribe them to a mailing list on another server, and potentially set up billing on yet another server. All of this would be accomplished (ideally) in a single script. By scripting these transactions, and then scheduling each script, we allow each business operation to run securely and conveniently.

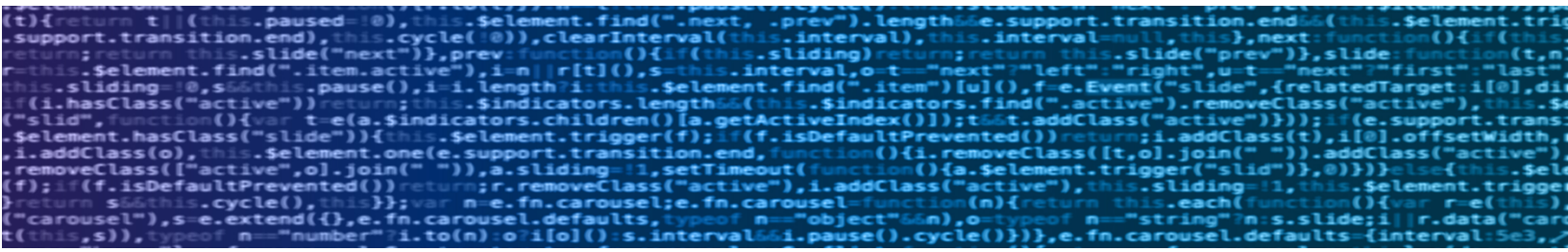
Although scripting is free in most computing environments, and most network administrators know how to write scripts to automate business processes, **scripts do have some drawbacks** to be aware of. First and foremost, scripts are simply programs,

and programming may be a skill that only a few staff members have. You also have to think about managing complexity. Are your business operations scripts becoming so large and complicated that they are becoming difficult to maintain? And at some point, change management becomes an issue. That is, what if someone makes a single-character change to a script that breaks the script? Is there a testing regimen in place to catch mistakes like this? Or does anyone actually look at the cron logs to verify that the scripts ran correctly? These are tedious tasks that typically get ignored.

Unfortunately, change management and error checking are only part of the problem with relying solely on scripting to perform critical business operations. If your scripts use user name and passwords to authenticate, you have the problem of managing those credentials. Typically, credentials are stored in plain-text in files for use by the various scripts. This is an obvious security problem. Public/private keys can also be also compromised but they are much easier to secure.

When writing file transfer scripts, very few system administrators take the time to build auditing and exception reporting into their scripts. Let's face it, this is a lot of work for something that we hope we never have to refer back to! But when you do need to find out when a particular script started to fail, detailed logs can be invaluable.

Fortunately, there are solutions.



Managed File Transfer (MFT)

Managed File Transfer (MFT) solutions solve the problems that we've outlined here, while adding unique benefits as well. To make things easy, most MFT solutions support the file transfer protocols that you are probably already using, such as the ones we've discussed here. Most MFT solutions use GPG, PGP, or AES to encrypt files in transit and at rest. This feature alone could be a huge benefit in some of the more regulated industries.

There are many reasons to migrate an existing file transfer regimen to an MFT solution.

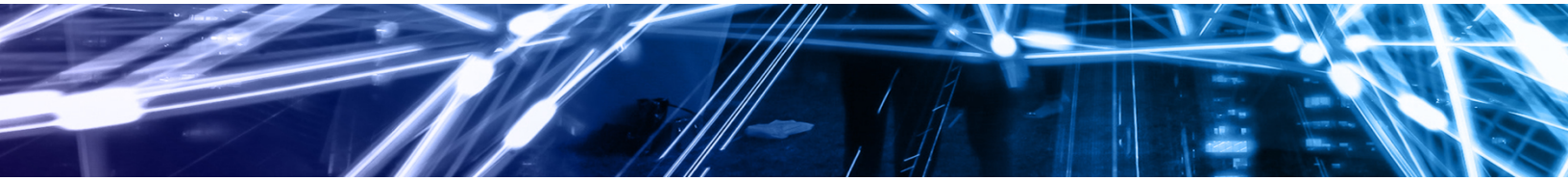
> An enterprise-grade MFT solution should provide enterprise features, including revision history, scalability, and support.

An MFT solution allows a system administrator to centralize business process scheduling, monitoring, auditing and alerting. It's quite common to have business processes scattered across various servers, each in one or more scripts that are scheduled via cron. This lack of centralization is a natural consequence of day-to-day operations in an ever-changing IT environment. But it can also make the system administrator's job more difficult. For example, if a particular business process begins to fail, it could take a significant amount of time to detect the failure without proper (and uniform) monitoring and alerting. Once the failure has been noticed, the offending scripts have to be located on whichever server they happen to run from and then the scripts have to be debugged to find out why they failed. Sure, most system administrators organize and even document how such processes operate, but even this is an unnecessary burden if there are convenient alternatives. It should be noted that notification and alerting could help comply with data security standards in certain regulated industries as well.

An enterprise-grade MFT solution should provide enterprise features, including revision history, scalability, and support. While it's possible to impose a change management system upon the scripts that drive an organization's business processes, this simply amounts to additional administrative burden on an already overburdened IT department. The MFT system should be able to track changes so that if something does break, the change that caused the problem can be quickly identified and backed out.

The last major consideration for an MFT candidate is installation flexibility. It might make sense to be able to deploy the system in the cloud, or on-premises. Being able to support Windows and Linux might be important. Keep in mind that these needs may change over time. An organization that manages its resources on-site today may migrate to the cloud later. On the other hand, an organization that is mostly cloud-based today may decide that it wants to own the hardware resources it relies upon. Any MFT worth considering should be flexible enough to adapt to such changes.

GoAnywhere is an enterprise-ready Managed File Transfer solution from HelpSystems. GoAnywhere will enable an organization to simplify, secure, and automate critical file transfers in a centralized, manageable manner. GoAnywhere is easy to install, deploys on-site or in the cloud, and supports most operating systems. We invite you to learn more and **try it yourself free for 30 days**.



About the Author:

Mike Diehl has been a Linux Administrator since 1997 and has a broad technical background including Networking, Programming, and Database Administration. Much of his career was spent in High Security environments. Mike lives in Columbia, SC. with his wife and 4 sons.