The Magic Proxy

Ali Tariq, Edward Suh, and Todd Mills
The Massachusetts Institute of Technology
{alitariq, suh, mills}@mit.edu

Abstract

The rapid growth of the Internet has created the need for widespread file sharing between Unix and Windows machines. Doing this securely has always been a problem. This paper presents a proxy for securely accessing files over the Internet, specifically files stored in Unix machines from Windows machines. The proxy talks to the Windows machines using the CIFS protocol and Unix machines using the SFS file system. The main strength of the system is that it can be used easily and without a lot of modification to existing computer systems.

1 Introduction

The rapid growth of the Internet over the past years has resulted in a wide use of Windows based PCs and huge distributed storage systems, which are usually based on Unix. It has brought Windows and Unix closer together than was previously thought possible. Most people have web-based email accounts which accesses file servers though the HTTP protocol. In companies and research labs, people have Windows boxes on their desks and at the same time have Unix file servers. Windows and Unix systems have been invading each other's area, and therefore it became crucial for Windows and Unix to share files and services.

Also, the Internet and distributed storages have created the issue of security. As data is transmitted over huge networks, which are open to anyone, there is danger for an attacker to intercept or change important information. Therefore, it is essential to have a secure way to access files over the Internet.

People have done intensive studies on the file accesses between different platforms. FTP has been a good solution for one-way file transfers, which was the characteristic of the Internet and the World Wide Web. However, as the Internet becomes more interactive, read/write interfaces are becoming increasingly necessary. The Common Internet File System (CIFS) protocol [1], which is also referred to as the Server Message Block protocol, LanManager or NetBIOS protocol, was developed to support more interactive file accesses such as shared files, byte-range locking, coherent caching, change notification, replicated storage, etc. Currently, the Samba suite [2] implements the CIFS protocol for Unix systems, and therefore enables Windows machines to access Unix machines' file system.

Although no existing file system has grown to span the Internet, there are several secure network file systems proposed to provide secure file sharing over distrusted networks. The most recent approach is the Self-Certifying File System (SFS) [3]. SFS has separated key management from file system security and provides security, versatility and a global name space.

Unfortunately, no single solution has solved the problem of enabling secure communication over distrusted networks between different platforms. The CIFS protocol does support interactive interfaces between Windows systems and Unix systems, but does not provide security. Moreover, the Samba program suite should be installed and run on all the Unix files servers to share Unix files on Windows machines, which makes file sharing difficult. Secure file systems such as SFS do provide security but are unable to communicate with Windows machines.

We propose a CIFS to SFS proxy to enable secure file access between Windows clients and Unix servers. We call this the Magic Proxy. The proxy runs on a Unix machine. It communicates with Windows machines using the CIFS protocol, and communicates with Unix file servers using the SFS protocol. The proxy server authenticates Windows users using a user name and a password. It can service multiple Windows users and multiple SFS file servers at the same time. Assuming secure channels between the proxy and Windows machines, the proxy server provides secure file sharing between Windows clients and Unix file servers. Also, if Unix servers support the SFS protocol, our approach does not require a new program such as the Samba to be installed on the file servers. Therefore, the proxy makes it easy to share files between Windows and Unix.

The paper is organized as follows. Section 2 explains the CIFS protocol, Samba, and SFS as a background. In Section 3, we discuss our design goals and high-level design decisions. Section 4 presents detailed implementation of our system. Then In Section 5, we evaluate our design, and Section 6 concludes the paper.

2 Background

There are several different components that went into the design of Magic Proxy. This section will give a brief overview of those components including CIFS, Samba, and SFS.

The Common Internet File System (CIFS) protocol [1] is most widely used by Windows to share files over the Internet. CIFS contains the ability to support multiple users reading and writing files, file locking, and file sharing. It runs over TCP/IP and has a global namespace. CIFS security is lacking because it only lets servers authenticate clients. The communication channel is not encrypted and clients cannot authenticate the servers they are talking to.

Samba [2] is a program that implements CIFS for Unix systems. It allows Windows clients to access files on Unix systems just as if they were another Windows system. The services it provides include file and print services, authentication and authorization, name resolution, and browsing. The Samba source code is freely available under the GNU Public License and so can be easily modified.

The Self-certifying File System (SFS) [3] is a file system that is designed for security, a global name space, and versatility. It separates key management from file system security. By using self-certifying path names clients can authenticate servers. Also SFS allows clients to authenticate themselves to the server using an agent run by that user.

3 Design

Magic Proxy's main functionality is to let Windows users access file systems located anywhere on the Internet. We designed Magic Proxy with several goals to help us provide this functionality. We then used these goals to provide us with the basis for the architecture of our design.

3.1 Goals

We designed Magic Proxy with three goals. First it needs to be secure since there are many rogue parties on the Internet. Second, Magic Proxy must be easily usable. Users will only use Magic Proxy if it is convenient and not difficult to use. Finally, Magic Proxy needs to be cleanly implemented in the sense that we do not want to make a lot of hacks and modifications for it to work.

3.1.1 Security

Since we want Windows users to access file systems located anywhere on the Internet we need to make sure there is adequate security between the Windows client and the server. Security can be provided by the client being able to verify the identity of a server, the server verifying the client has access permissions, and by encrypting the communication channel between the client and the server. The CIFS protocol Windows uses for remote file systems unfortunately does not provide much security at all. It only provides a way for a server to authenticate a client. So, to fulfill the goal of security we need a better way of communicating with file servers. Magic Proxy does this by using SFS. SFS provides us with security, a global namespace, and decentralized control. Having a global namespace and decentralized control are also two very good properties that make Magic Proxy even better.

3.1.2 Usability

Magic Proxy needs to be easily usable so that all users can benefit from it without a steep learning curve. Magic Proxy accomplishes this goal by using the standard Windows interface for mapping network drives and connecting to other Windows file systems. This allows the users to access the SFS servers just as if they were connecting to another Windows computer. Users do not need to learn a new program but only need to learn about functionality already built into Windows which they may already be familiar with.

We also want Magic Proxy to be easily maintained. Users do not want to constantly maintain programs just to keep them working. We want Magic Proxy to just work with no user intervention. We accomplish this by having each user configure Magic Proxy just once and then after that point it will work without having to do anything more. The only time a user is even aware of Magic Proxy is if it were to crash but even in this case when Magic Proxy starts running again it's as if it never crashed. All of the configuration data is retained and the user can resume accessing SFS servers like nothing happened.

3.1.3 Clean Implementation

The final goal for Magic Proxy is that we wanted a clean implementation. The reason we did this is because we did not want to complicate the pre-existing code and make it more difficult to maintain and upgrade. As much as possible we tried not to make too many changes to the SFS and Samba code.

3.2 Architecture

Since we decided to use SFS because of its ability to secure the communication between the server and client we had to figure out how Windows would use it. The SFS code is written for use with a Unix operating system so getting it working on Windows would involve a complex porting process. Doing this would also defeat our goal of having a clean implementation since we would need to modify SFS with all sorts of hacks to get the code working on Windows. In addition if we had gotten SFS ported to Windows we would still need to write a CIFS to NFS protocol converter because the SFS servers currently use NFS to communicate with the local file system. For these reasons we decided that we should have Windows access SFS using a different method.

The method that worked best was to have a proxy running on a separate computer between the Windows computer and the SFS servers. We call the proxy the Magic Proxy. The proxy converts from the CIFS protocol to the SFS protocol and also the proxy is running Unix and so can easily run SFS. The only other major component the proxy needs is something to communicate with the Windows client using the CIFS protocol. This can be done using the Samba software. We do not need to modify Windows at all to make this work because it just keeps on communicating using the CIFS protocol.

The only problem with using a proxy is that the communication between the proxy and the Windows client is using the CIFS protocol. This protocol implements very little security so we still need some way to provide security between the Windows machine and the proxy. There are a couple of ways we can deal with this. First, we can assume the connection to the proxy is trusted and that the proxy is located on the client's local area network. Most likely we will not trust the connection to the proxy so the other alternative we can do is to encrypt the communication between the client and the proxy. To do this we can use a protocol like IPSec [4] to secure this channel. Since the client only needs to communicate with one other machine using IPSec it will only require configuring the client once to securely talk with the proxy. The client does not need to be configured for each SFS server it talks to.

4 Implementation

As in all proxies Magic Proxy is invisible from the point of view of the Windows user. It runs on a separate machine – not the Windows client or a file server. This proxy machine can run either Linux or the Unix operating system.

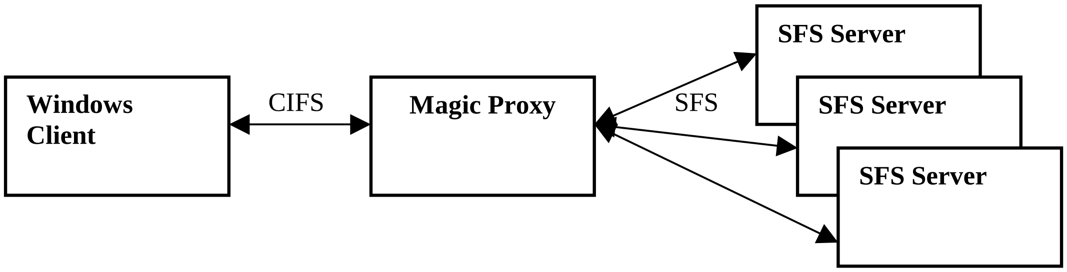

Figure 1: General system overview

On top of the operating system runs the proxy, which consists of Samba and the client side of SFS running as one. Samba provides the interface between Windows CIFS commands and the Linux/Unix file system. SFS-client takes over from there and talks to the SFS-server, which runs on the file server. Figure 1 illustrates this setup.

As Figure 1 shows, the proxy simply consists of Samba and SFS running side by side. Ideally, this should have been a straightforward setup. However, this was not the case. Both Samba and SFS, and even Windows require modifications and tweaking to start talking to each other properly.

4.1 Windows

To be able to use the file servers securely, the user will have to map the directory being served as a network drive in Windows. He will be required to enter a username and password. Upon entering those, he will be able to view the served directory. Any subsequent actions taken by the user on the directory are converted on the fly by the proxy into the appropriate format and passed on to the SFS server. Similarly, responses from the server are converted back and passed to Windows. As far as the user is concerned, the served directory will appear as any other directory in Windows. As mentioned earlier, the proxy remains invisible to the user.

Prior to using files from an SFS server, the user needs to register a private key with that server. Or, he needs to import a key registered with some other server. The Magic Proxy does not support creation/registration of new keys. Instead it enforces that the keys should be registered in advance on the proxy machine.

A user is created on the proxy machine for every person wishing to use the proxy to access SFS servers. Each user also has an SFS key registered with the proxy machine. The key needs to be decrypted by a ‘pass phrase’ before it can be used. The login password for users on the proxy has to be the same as the ‘pass phrase’ needed to decrypt the keys. This makes logging into SFS servers and reading files very easy.

The username and password entered by the user in Windows upon mapping the served directory is the same username and password that logs him into the proxy machine, and gets his keys decrypted. He needs to only enter the password once from Windows.

This implementation required a slight change to Windows. In the normal setup, Windows encrypts all passwords before sending them out over the network. In this case, non-encrypted passwords are required by Samba and SFS to be able to decode the keys properly. Hence, a small change to the Windows registry has to be made before using the Magic Proxy. This change effectively ensures that Windows sent out plain-text passwords.

4.2 Samba

Our original aim was to have Samba running on the proxy machine as a user process. In that scenario, whenever a user would want to use our file system, he would run a Samba process, which would then enable him to talk to SFS. This would have been useful, as no administrator intervention would be needed. Unfortunately, due to the way Samba works, this scenario did not seem very plausible.

Basically, Samba enables the Windows CIFS to talk to the Unix file system. In other words it converts CIFS commands to Unix file system commands. However, along with all this it does a lot of necessary house keeping type functions. When a new user wishes to use the Unix file system, he enters his username and password into Windows. This is grabbed by Samba, which then proceeds to log the user into the Unix system. Once that is done, Samba checks the permissions on the file(s) the user wishes to access. If he has adequate permissions Samba enables him to access the file. These actions can be taken by Samba only if it is run as a super-user process.

In addition, the Windows CIFS is programmed such that it only talks to a certain port (port 137 or 139) on the server machine. A simple user-level process cannot have access to this port on the server machine. Hence, a super-user process was needed.

4.3 SFS

One of the first and biggest issues with SFS was the code. SFS code is extremely difficult to understand, especially because comments are very rare and the code isn't self-documenting. The extensive usage of C++ templates and the asynchronous library makes the code even more unreadable. The unavailability of any documentation is also distressing. Hence, all efforts were made to reduce any modifications to the SFS code.

Just like Samba, the original aim had been to run SFS-client on the proxy machine as a user-level process. In that scenario, SFS client would not run continuously in the background. Instead it would be started up whenever someone would need to use one of the SFS servers. Again, administrator intervention would have been minimal. However, getting SFS-client to run as a user process isn't exactly straightforward.

SFS-client uses a program called 'nfsmounter', which runs in the background and waits for the user of the SFS to 'cd' into an SFS directory, indicated by the '/sfs/host-id:key' syntax. It then automatically mounts that directory onto the local machine. This mounting requires super-user access. A solution could have been to remove the auto-mounting done by nfsmounter.

However, the auto-mounting is useful for a very important reason. It makes sure SFS receives the right input. Samba sends out Unix file system commands. SFS receives NFS commands only. Due to the auto-mounting, the underlying Unix or Linux OS recognizes that the directory being accessed is not on the local machine and automatically converts the Unix calls to NFS calls. Hence the output of Samba is transparently and automatically converted to a format that SFS can understand. If the auto-mounting is removed, this conversion will have to be done by the Magic Proxy itself.

A lot of time was spent in achieving this specific goal. Due to the complexity of NFS, that is not a simple task. As the decision to run Samba as a super-user process had already been made, it was decided that this OS to NFS conversion would mostly be redundant and unnecessary. It was hence decided that SFS too would be run as a super-user process, thereby making the implementation of the Magic Proxy much simpler and maintainable.

Also, this ensured that the SFS client is run only once, which is more than enough. A separate SFS client for each user would have been overkill.

The SFS client code also had to be modified slightly. The self-certifying pathnames for SFS directories contain the ':' character. Windows does not accept the ':' character in pathnames. However, it does allow the '+' character. A small change was hence made to SFS-client to allow Windows to use the '+' sign instead of the ':' sign in the self-certifying pathnames.

4.4 SFS-Samba Communication

The communication between Samba and SFS client was hence made very simple. SFS client and Samba both run all the time, in the background. Whenever a new user logs in and wishes to use an SFS server, Samba authenticates him and upon authentication, runs a copy of sfsagent, which actually spins off a new process for the user. Samba then sends the user password to this new process. Sfsagent uses

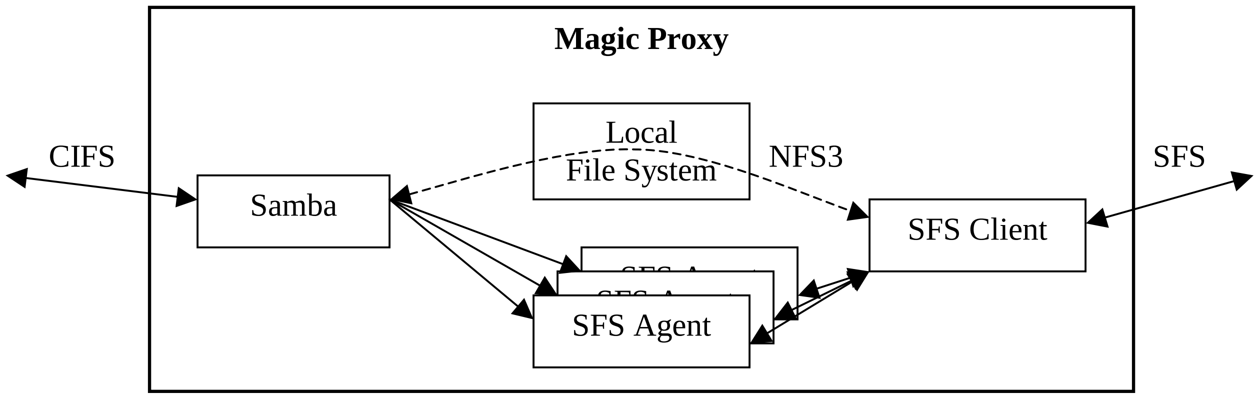

Figure 2: Magic Proxy detailed view

the password to do key manipulation. The decrypted key is then used by SFS-client to access files from the SFS-server.

This passing of the password from Samba to sfsagent required some thought. As plain text passwords are being used, and both sfsagent and Samba run as independent processes, this interface had to be handled very carefully. A simple command line argument was not deemed safe enough. It was decided that inter-process communication methods would be used. Specifically, the Unix inter-process messaging queue method was used. The Samba process creates a message queue and passes the password to it. The new sfsagent process can then receive the message from the queue.

Figure 2 shows the detailed view of the Magic Proxy.

4.5 How it all Works

For someone to be able to use the Magic Proxy, the super-user for the proxy machine has to first create an account for that user on the proxy and then set up a key for him. The user then gets the self-certifying path for served SFS directory and maps that directory on the Windows machine, substituting the ':' for the '+'. For example, for the self-certifying pathname '/sfs/schnapps.lcs.mit.edu:erpenm2vte9g7rpnsimgnidsj6fw4pz9', the user would map the directory '\\proxy_path\sfs\schnapps.lcs.mit.edu+erpenm2vte9g7rpnsimgnidsj6fw4pz9'. Upon entering his username and password, he would have access to that directory.

5 Analysis

A major part of work was done to understand and configure SFS and Samba to work with each other. Once the SFS-Samba code was running, it was realized that it fulfilled a lot of the objectives and goals of the project.

Our system is secure. All connections between Samba and the SFS servers are encrypted by SFS. This achieves a high level of security. The only weak link is between the Windows client and the proxy. Windows sends plain text passwords to the proxy. This implies that the Windows-proxy connection has to be heavily trusted. However, this can be easily achieved by using a protocol like IPSec.

The Magic Proxy is very easy to use. Users of the proxy do not need to install any additional software on their existing Windows setup. Mapping an SFS directory is as simple as mapping any directory in Windows. Once the mapping is done, the user can access the mapped directory as any other Windows directory. The proxy running in the background is totally invisible to the user.

The implementation is very clean. Small, manageable changes were made to SFS and Samba code. As stated, most of the work is done by Samba and SFS. The main functionality of any of the programs was not changed. Most importantly, none of the Windows code, or Linux/Unix code had to be changed. The proxy runs on a standard Unix or Linux implementation and talks to standard Windows machines and SFS servers running on standard Unix/Linux machines.

The system can support multiple users accessing it at the same time. It invisibly connects multiple users and runs of an independent sfsagent for each user. The sfsagent then does all key management for each user.

In retrospect, implementing the proxy as compared to a SFS client for Windows was a good idea. Implementing an SFS client for Windows would have provided a comparable end product. However, it would have involved a lot of complex coding for a result that would not have been phenomenally different. As mentioned earlier, SFS code is extremely complicated. Getting it to run on Windows would have been very time consuming and the final product would have been bug-ridden. SFS's native asynchronous library would have to be ported to Windows. Again, the complexity involved in that would have been immense. Finally, each person who wanted to use the system would have to install the software on his Windows machine. That would have been an additional complication.

6 Conclusion

In this paper we discussed the problems associated with sharing files over the Internet between Windows and Unix machines. Not only is it difficult to do, security is a major concern. We talked about the various tools available that offer security and file sharing. We then presented an implementation that used these tools together to create a proxy, a Magic Proxy to achieve the objective of securely sharing files. We showed that this was easy and very possible. We also showed that the final product is easy to setup and easy to use. It is also very flexible in that it lets the user easily use multiple directories residing on multiple servers easily and securely.

References

[1] Microsoft. Web Workshop – CIFS Overview. From http://msdn.microsoft.com/workshop/networking/cifs/

[2] Chris Hertel. Samba: An Introduction. From http://us1.samba.org/samba/docs/SambaIntro.html

[3] David Mazieres, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. *ACM Operating Systems Review*, 34(5):124-139, December 1999.

[4] S. Kent and R. Atkinson. Security architecture for the internet protocol. RFC 2401, Network Working Group, November 1998.