6.894 Final Lab Paper

Using SDSI to implement group access control in SFS

Min Wu George Savvides Jordan Grouevsky

December 12, 2000

Abstract

The current implementation of SFS does not allow access control for users who are unknown to the SFS server. Adding new users requires the manual intervention of the server's administrator which makes it inconvenient to allow access to not-yet-known members of a known group.

Our paper describes a way to overcome this limitation by integrating SDSI key management policies into SFS. In the described scheme, the administrator of an SFS server can permit users who are unknown to the server to use certificates to prove their membership to a specific known group and, subsequently, gain the corresponding level of access to the server. This obviates the manual intervention by the administrator, which would otherwise be necessary every time a new user needs to access the server.

The paper provides details of how this mechanism can be integrated into SFS. We do not have a working version of the changed system due to several implementation and testing difficulties.

1 Introduction

SFS ([1, 2, 3, 4]) uses self-certifying pathnames to build a mechanism that allows clients and servers to communicate securely over an insecure medium like the Internet. Several cryptographic protocols are used to ensure file system security. However, SFS does not enforce any accompanying user authentication mechanisms; in fact, user authentication has been completely decoupled from the file system security that SFS strives to achieve. The rationale behind this design approach is basically an end-to-end argument: user authentication should be handled by external programs using protocols opaque to the file system itself, so that different applications can efficiently meet their particular security needs. In order to satisfy this goal, SFS was designed to allow arbitrary key management policies to be implemented on top of the file system.

The current implementation of SFS provides a rudimentary authentication mechanism which allows the administrator of an SFS server to give different users different levels of access to the files stored by the server. As described below, the current mechanism suffers from severe limitations. The one we tried to address involves *group access*: the server's administrator has no way of granting access privileges to the members of a group whose composition is unknown, and potentially variable.

We propose the use of SDSI ([5, 6, 7]) certificates to implement a group management policy. The server's administrator can specify a set of group names and their associated credentials in essentially the same way that user names and credentials are defined in the current implementation. A hitherto unknown user can then use SDSI certificates signed by a Certificate Authority known to the SFS server, in order to prove her membership to a specific group. The SFS Server then automatically adds an entry for the user, with the same credentials as the group she belongs to, followed by a reasonable expiration date.

While the idea seems simple enough, the actual implementation of our proposed scheme ended up being significantly more difficult than we had expected. On top of that, the Athena environment proved to be badly suited to our needs. For example, it does not provide NFS3 Server Support which is necessary to test modifications to our server code. Moreover, the C++ SDSI library does not even compile on Athena (a newer, experimental version does, but that one has other problems). As a result, we can only present the theoretical aspect of our approach, as well as pseudocode which illustrates how our proposed modifications could be implemented. We hope that this paper will be useful to anyone who chooses to continue our work in the future.

The rest of the paper is structured as follows:

- Section 2 describes the current authentication mechanism in SFS and its limitations
- Section 3 describes our proposed scheme to allow group access in SFS
- Section 4 describes in pseudocode what should change, and where
- Section 5 provides our suggestions for future work

2 Current user authentication mechanism

2.1 Description

If user Alice wants to access an SFS server, she must first have an account on the machine on which the SFS server is running. That is, her `username` must be in the system's `passwd` file. This must be done *manually* by the administrator of the server.

Then Alice needs to register a public key with the server. In order to do that, she should:

- log into the machine (e.g. using *ssh*) and make sure that `sfssd` is running
- use the `sfskey` command to contact the SFS authentication server to:
  - generate a new Rabin-Williams public/private key pair
  - register the key pair
- have her local `sfsagent` securely download the private key from the server using the SRP protocol, so that it can take care of all future authentication requests by the SFS server.

`sfsagent` keeps the user's private key in memory and uses it to sign authentication requests. On the server side, `sfsauthd` maintains a database mapping public keys to user credentials. When `sfsauthd` receives a valid request from the file server, it replies with a set of UNIX credentials – a user ID and list of group IDs. Based on these `UIDs` and `GIDs`, the server can decide whether or not the user's request should be honored.

2.1.1 An example

Here is how this would actually be done on `random-beatings.lcs.mit.edu` by user `Alice`, who has just connected using *ssh*:

```
> sfskey register -u Alice
Registering new user Alice@random-beatings.lcs.mit.edu. Creating new key for
Alice@random-beatings.lcs.mit.edu.
       Key Name: Alice@random-beatings.lcs.mit.edu
Enter passprase:
          Again:

sfskey needs secret bits with which to seed the random number generator.
Please type some random or unguessable text until you hear a beep: DONE
  UNIX password:
```

At the end of this procedure, Alice's brand-new public/private key pair has been generated and registered with the server. She needs to download the key pair to the SFS client she is using, so that her `sfsagent` can use it to reply to authentication requests. Here's how this is done:

```
> sfsagent Alice@random-beatings.lcs.mit.edu
random-beatings# sfssd: accepted connection from 18.34.0.229
Passphrase for Alice@random-beatings.lcs.mit.edu/1024:
```

2.2 Limitations

Before a new user can access the SFS server, he should be added to the list of the host machine's known users. Therefore, the administrator of the SFS server needs to be involved any time a new user wants to access the server, check the new user's credentials and manually add an entry for the user in the `passwd` file.

This is a major limitation, especially when the administrator of the SFS server wants to grant access to entire groups with large and variable membership lists. If an SFS server is supposed to give read rights to all MIT students, for example, the server must somehow maintain a complete and up-to-date list of all MIT students, a formidable task if adding and deleting group members can only be done manually by the server's administrator.

3 Allowing group access

In our proposed scheme, every user is assumed to have a public/private key pair for SDSI certificates and the ability to obtain certificates that prove his/her membership to a group from certificate authorities that are recognized and trusted by the SFS server's authentication server. Our proposed modifications should allow the SFS administrator to grant access to members of each specific group by manually creating an entry for the group only. This can be achieved using SDSI certificates, as described below.

3.1 Creating a group

Suppose that the administrator of an SFS Server wants to grant all MIT students read access to the files on the server. She first creates an account for a user called `MIT-students` on the local machine by inserting the appropriate entry in the UNIX `passwd` file. She then registers `MIT-students` to the SFS server, giving it the desired level of access. This does not require any modifications to the current SFS code: at this point, `MIT-students` is simply a user.

3.2 Group certificates

Each member of the group is equipped by a SDSI certificate which effectively binds his name and public key to the name of the group he belongs to. For example, Alice, being an MIT student, has a certificate that looks like this:

```
(cert
       (issuer (public-key (hash sha1 |PWKULKycrQ/Pxu9qWBSY2Q==|)))
       (group (name "MIT-students"))
       (subject
          (name "Alice")
          (public-key (rsa-pkcs1-md5 (e #010001#) (n|ALOzv9lJoA5XmpKFA8S
            NeTMwb40c6tNgaaeKmi6s75RNPSWAf8NwRw9aSr1ZZZXUK2dIFy23C
            bo+/iPjerp7gLLi5r61w9WSGKo5mmYT+6Vs9Pyxq6wazUG2Euz/11aWgog
            m9GLRH4zbHkcqPciorWRKf9bhGF2Eqm/aYpCXw1x3|)))
        (signature (hash sha1 |HdgtSDS4fdgp9Bcfg/d/A1e=|)) )
```

Understanding what the certificate means

- `(issuer (public-key (hash sha1 |PWKULKycrQ/Pxu9qWBSY2Q==|)))`
  is the message digest of the issuer's (e.g. MIT Certificate Authority) public key.
- `(group (name "MIT students"))`
  is the group the user belongs to.
- The information about the user and her public key is contained in:

  ```
      (subject (name "Alice")
      (public-key (rsa-pkcs1-md5 (e #010001#) (n|ALOzv9lJoA5XmpKFA8
  SNeTMwb40c6tNgaaeKmi6s75RNPSWAf8NwRw9aSr1ZZZXUK2dIFy23Cbo+/iPjerp7gLL
  i5r61w9WSGKo5mmYT+6Vs9Pyxq6wazUG2Euz/11aWgogm9GLRH4zbHkcqPciorWRKf9bh
  GF2Eqm/aYpCXw1x3|)))
  ```

- The issuer's signature for the whole certificate is given by:

  `(signature (hash sha1 |HdgtSDS4fdgp9Bcfg/d/A1e=|))`

The certificate above binds Alice to her public key and is issued by the MIT Certificate Authority.

3.2.1 Using Group Certificates in SFS

If we link the SFS Server code to the SDSI Library, we can use the already existing functions to verify the authenticity of the certificate and extract the relevant information (student's name, public key, name of group s/he belongs to). Before this can be done, the SFS administrator should, of course, enter the public keys of the known and trusted Certificate Authorities into the SFS server. That need only be done once, though.

3.2.2 Granting access to the user

Returning to our example, after it has been established by the SDSI Library that user Alice is a member of `MIT-students`, the server should grant her the same level of access that the "generic" user called `MIT-students` has. The main difficulty here is that Alice is unknown to the underlying UNIX system, therefore she cannot simply use `sfskey` to interactively generate a Rabin-Williams public/private key pair, and then log into the machine using her UNIX password and retrieve the private key through the SRP protocol as it is done in the current implementation.

We can avoid these difficulties by implementing another protocol, which allows the user's `sfsagent` to contact the SFS Server's `authserv` and pass it a *signed and timestamped* request structure which includes a SDSI certificate. When it receives such a structure from the user, `authserv` does the following:

- It checks the certificate for validity (that is, whether it was signed by one of its trusted Certificate Authorities)
- It verifies that the request was indeed sent by the user described in the certificate, by checking the signature of the request against the public key contained in the certificate. It also checks the timestamp to prevent replay attacks.
- It generates a Rabin-Williams public/private key pair for the user, similarly to the current mechanism. However, instead of using the user's keystrokes as a source of randomness, it uses some other, local source of high entropy such as the system clock. This way the key pair can be generated on the SFS server without requiring any interaction with the user.
- It sends the private key to the user's `sfsagent` *securely*, by embedding it in a structure that also contains the SFS Server's public key, signing it with the SFS server's private key, and encrypting it with the user's public key (available from the certificate).The user's `sfsagent` can check the validity of the SFS Server's public key by comparing its hash to the hash contained in the server's self-certifying path. It can then verify the authenticity of the message by checking the signature.
- It retrieves the credentials of `MIT-students` and adds the user to the `sfs_users` database with the same credentials.

The user's `sfsagent` can now authenticate its user to the SFS Server's `authserv` any time this is required.

Note Since the SFS server currently depends on the UNIX credentials, and there is only one `MIT-students` entry in the UNIX `passwd` file, we need to allow an SFS user to have several different Rabin-Williams public/private key pairs, one for each newly-added member of the group. This way, every time the SFS server needs to make a call involving one of the group members to the underlying UNIX system, it will actually be a call involving the existing `MIT-students` entry in the `passwd` file.

4 Implementation

The numerous problems with SFS and SDSI that we encountered during the last couple of weeks prevented us from implementing a working version of our modified code. Here we present pseudocode to illustrate what should change, and where.

4.1 Making certificates available to the local sfsagent

Suppose that user `Alice` wants to make her newly-acquired certificate available to her local sfsagent. She can use a function like this:

```
>sfscertificate add Alice certfile keyfile
```

This loads `Alice`'s SDSI certificate from file `certfile` to `sfsagent`, as well as her private key which will be used to sign the request structure (which includes the certificate)

The following procedure can be used to load the certificate into the agent's memory

```
void sfscertificate::add ( str username, fd certfile, fd keyfile )
{
    load user's certificate into memory;
    load user's private key into memory;
    create a structure containing the certificate and a timestamp, and sign it using
      the user's private key
    make the signed structure available for retrieval by the username;
}
```

4.2 Sending a certificate to the remote authentication server

Client Side

We can do this by using `sfsagent -c` in order to run a procedure `sfscertificate::cert_send` as follows:

```
void sfscertificate::cert_send ( str username, str server )
{
    retrieve the associated signed structure from memory (containing timestamp+certificate)
        using the username ;

    //send it to the remote server using RPC;
    sfs_connect_cb->call ( SFSAUTHPROC_CERT, &arg, &sres, wrap (this, &certcon::certcb);
}
```

Server side

The server's `authserv`:

- Receives the certificate from the user, checks the signature with the certificate authority database (`certdb`) which contains entries of this type:

```
MIT administrator :
      (public-key (rsa-pkcs1-md5 (e #010001#) (n|ALOzv9lJoA5XmpKFA8S
          NeTMwb40c6tNgaaeKmi6s75RNPSWAf8NwRw9aSr1ZZZXUK2dIFy23C
          bo+/iPjerp7gLLi5r61w9WSGKo5mmYT+6Vs9Pyxq6wazUG2Euz/11aWgog
          m9GLRH4zbHkcqPciorWRKf9bhGF2Eqm/aYpCXw1x3|)))
```

- Generates a new entry in SFS server's `sfs_users` file for `Alice` and assigns it the same UNIX credentials as for `MIT-students`.
- Specifies an expiration time for this entry, for example one day.
- Generates a new temporary Rabin-Williams key-pair for the new user, writes it into `sfs_users`. The structure of `sfs_users` is as follows:

```
MIT student : (tmp RW key pair) : Alice : (Alice's public key) : (validity interval) :...
MIT student : (tmp RW key pair) : Bob : (Bob's public key) : (validity interval) :...
MIT student : (tmp RW key pair) : john : (john's public key) : (validity interval) :...
```

  The columns above correspond to the following:

  1. The username that appears in the UNIX `passwd` file
  2. The temporary Rabin-Williams key pair
  3. The actual username
  4. The user's Public key
  5. The validity interval for this entry

- Sends the Rabin-Williams private key back to user. (Note: the private key *includes* the public key)

We need to add a new case in `authclient::dispatch` , under the `SFSAUTH_PROGRAM` case, in order to dispatch the authentication to the `authclient:docertificate` procedure in the case where we send a certificate. The proposed modification, under the `SFSAUTH_PROGRAM` case is:

```
case SFSAUTHPROC_CERT:
  docertificate (sbp);
  break;
```

The `authclient:docertificate` procedure is given below:

```
authclient::docertificate ( svccb *sbp )
{
    verify user's signature;
    verify user's certificate;
    generate a new temporary Rabin-Williams key-pair for user;
    add an entry to sfs_users;
    encrypt user's Rabin-Williams private key by user's public key;
```

```
    sign using server's private key;
    send back to user's agent;
}
```

At this point, the user's `sfsagent` gets this temporary Rabin-Williams private key and uses it to answer subsequent authorization requests by `authserv`.

4.3 Changes to sfsauthd_config:

```
Hostname nameS
Keyfile path
Userfile [-ro|-reg] [-pub=pubpath] [-mapall=user] path
SRPfile path
Denyfile path
# add more items
# Store administrators' public key for certificate verification
Certdb path
```

4.4 Creation and manipulation of SDSI certificates

The implementation of SDSI by Matthew Fredette ([5, 6]) provides a rather extensive set of functions that can be used to create and manipulate SDSI certificates. Unfortunately, we couldn't get the library to compile on Athena, and a new version that does run in the Athena environment implements *certificate chain discovery* (which we are not using) but has other useful features disabled. Here is how the library can be used:

4.4.1 Generating a public/private key pair

While SDSI certificates do require a user to have a public/private key pair, the SDSI library itself does not generate such keys. However, one can use *ssh* or *pgp* to create an initial key pair and then, through the SDSI shell, run `pgp-ssh-to-sdsi pgp` or `pgp-ssh-to-sdsi ssh` to convert the key to a format that SDSI understands.

4.4.2 Creating Certificates

If user Alice wants a certificate that proves that she is an MIT student, she needs to first generate a public/private key pair as described above, and then somehow transfer it securely to a Certification Authority (e.g. by saving it on a floppy disk and bringing it personally to an administrator's office). The administrator can then use `sdsi2ui` to start the Graphical User Interface and issue a certificate binding Alice and her public key to `MIT-students`, and bearing the signature of the Certification Authority.

Note The Graphical User Interface is disabled in the new version of SDSI (the only one that runs on Athena)

4.4.3 Signing and verifying SDSI Certificates

The library contains the appropriate functions: for signing, one can use `sdsi2_signature_compute` and for verifying signatures one can use `sdsi2_signature_check`.

4.4.4 Extracting information from SDSI Certificates

The SDSI library does not functions that can be used to parse the S-Expressions (in which the certificates are expressed) and retrieve the user name (subject), public key, issuer's public key and other useful information. However, such functions should not be hard to implement. For example, given a certificate

```
(cert
       (issuer (public-key (hash sha1 |PWKULKycrQ/Pxu9qWBSY2Q==|)))
       (group (name "MIT-students"))
       (subject
          (name "Alice")
          (public-key (rsa-pkcs1-md5 (e #010001#) (n|ALOzv9lJoA5XmpKFA8S
            NeTMwb40c6tNgaaeKmi6s75RNPSWAf8NwRw9aSr1ZZZXUK2dIFy23C
            bo+/iPjerp7gLLi5r61w9WSGKo5mmYT+6Vs9Pyxq6wazUG2Euz/11aWgog
            m9GLRH4zbHkcqPciorWRKf9bhGF2Eqm/aYpCXw1x3|)))
        (signature (hash sha1 |HdgtSDS4fdgp9Bcfg/d/A1e=|)) )
```

one can write a function that would extract the username ("Alice") as follows:

```
char* username_from_SExpression (sdsi2Sexp *sexp)
  {
    look for string "name"
    skip the space
    start copying until reaching ")"
 }
```

5 Suggestions for future work

- As we already mentioned, because of time constraints and several unforeseen problems with SFS and SDSI, we were unable to implement and test the modifications suggested above. We think that implementing the proposed scheme would be a useful first step in integrating SDSI-style group access control into SFS and we remain confident that this is indeed a feasible task.
- Moreover, our proposed scheme could be extended to take advantage of the numerous capabilities that SDSI offers. For example, our scheme assumes that the new user, who wants to access an SFS server for the first time, knows *in advance* which certificate should be used, and initiates the appropriate protocol to transfer it to the server. The new version of the SDSI library implements mechanisms that allow the two sides to interact and discover which, if any, certificates owned by the user can establish his right to perform the requested operation on the server. This protocol, which is described in detail in [8], can be used to automate the task of determining whether a certificate should be sent, and which one, thereby making the whole process completely transparent to the human user.

- Another suggestion would be to add "smart" *caching* of users on the server's side, so that future accesses from the same users, within a reasonable time period, do not incur the overhead of passing and verifying certificates. This, however, requires that the suggestion made above is implemented first, since it requires an interactive mechanism that establishes what the user's agent would need to present to the SFS server in order to gain the requested level of access.

6 Acknowledgements

We would like to thank the TA, Jinyang Li, for her invaluable assistance with SFS and all those hours she spent trying to fix the problems we encountered. We would also like to thank Dwaine Clarke, one of the co-authors of [8], for taking the time to explain to us at length the capabilities and limitations of the SDSI library we were using ([5, 6]).

References

[1] David Mazières, *SFS 0.5 Manual*, (MIT Laboratory for Computer Science)

[2] M. Frans Kaashoek, David Mazières, *Escaping the Evils of Centralized Control with self-certifying pathnames*, (MIT Laboratory for Computer Science)

[3] David Mazières, *Self-certifying File System*, (Ph.D Thesis, Massachusetts Institute of Technology, May 2000)

[4] M. Frans Kaashoek, David Mazières et al., *Separating key management from file system security*, (MIT Laboratory for Computer Science)

[5] Matthew Fredette, *The SDSI 2.0 Library and Tools, Release 0.4.5*, (MIT Laboratory for Computer Science, 9 July 1999)

[6] Matthew Fredette, *An Implementation of SDSI - The Simple Distributed Security Infrastructure*, (Master's Thesis, Massachusetts Institute of Technology, May 1997)

[7] Ronald L. Rivest, Butler Lampson, *SDSI - A Simple Distributed Security Infrastructure*, (MIT Laboratory for Computer Science and Microsoft Corporation, April 1996)

[8] Ronald L. Rivest, Dwaine Clarke et al., *Certificate Chain Discovery in SPKI/SDSI (work in progress)*, (MIT Laboratory for Computer Science, Intel, BBN, November 22, 1999)