maidsafe distributed file system

David Irvine, david.irvine@maidsafe.net, maidsafe.net limited (registered in Scotland Sc 297540)

September, 2010 ***Abstract*—Distributed file system's require server's or control nodes. Access to a file system is a security issue that can apparently only be controlled by some kind of authority and this is always a point of failure. These file-systems also require a mechanism to index the file-system. This paper presents a distributed file-system without centralised control or indexing. This file-system also utilises a distributed locking mechanism for data integrity for multi access to any file.**

***Index Terms*—security, freedom, privacy, file-system**

Contents

List of Tables

List of Algorithms

I. Introduction

FILESYSTEMS are a relatively new and slow changing part of computing. There is disagreement between operating systems and even version of operating systems how to handle access to data via a file-system. This has proven problematic over the years and led to many short term fixes that gain ground and then appear to vanish again. Such systems as CIFS, Andrews, SMB and many others appear to be an answer and for many reasons, occasionally political they lose ground again. It is the intention of this paper to provide a universal file system that implements a minimum set of features that will operate cross platform.

This system will represent itself to a user as a native file-system on any platform and as such requires low level drivers and code to be installed alongside any application using it.

A significant advance in distributed locking is employed which allows shares to be easily setup and maintained.

In addition there is a solution to the problem of location of data, or path sizes, which is limited on every operating system. This paper presents a mechanism to overcome this and allow for an almost infinite number of levels of directory structure to be implemented.

A. Conventions Used

There is scope for confusion when using the term "key", as sometimes it refers to a cryptographic key, and at other times it is in respect to the key of a DHT "key, value" pair. In order to avoid confusion, cryptographic private and public keys will be referred to as K_{priv} and K_{pub} respectively, and DHT keys simply as keys.

- Node $\equiv$ a network resource which is a process, sometimes referred to as a vault in other papers. This is the computer program that maintains the network and on its own is not very special. It is in collaboration that this Node becomes part of a very complex, sophisticated and efficient network.
- H $\equiv$ Hash function such as SHA, MD5, etc.
- $PBKDF2_{[Passphrase][Salt][IterCount]} \equiv$ Password-Based Key Derivation Function or similar
- XXX_{priv}, $XXX_{pub} \equiv$ Private and public keys respectively of cryptographic key pair named XXX
- $AsymEnc_{[K_{pub}]}(Data) \equiv$ Asymmetrically encrypt Data using K_{pub}
- $AsymDec_{[K_{priv}]}(Data) \equiv$ Asymmetrically decrypt Data using K_{priv}
- $SymEnc_{[PASS]}(Data) \equiv$ Symmetrically encrypt Data using PASS
- $SymDec_{[PASS]}(Data) \equiv$ Symmetrically decrypt Data using PASS
- $Sig_{[K_{priv}]}(Data) \equiv$ Create asymmetric signature of Data using K_{priv}
- $+ \equiv$ Concatenation
- STORE $\equiv$ Network or other key addressable storage system

II. Overview

A. Distributed directories

To enable a huge amount of data to be organised into a directory structure (as we are used to) then a new method of data management requires to be employed. In this case we have created a system of divorcing the structure of any data from a tree in one direction. In this paper we present a

system where a directory structure may be traversed forward from any point and with efficient implementation back to that point, but no further back unless new knowledge of the structure is gained.

This has the effect of allowing directory tree's to be free forming rather than tied to any root or base level, although in a way there is a base level that every tree can be traversed from, if implemented correctly and a user decided to traverse a tree from his own root directory or root share directory, in some cases , explained later. In this system a user can have a tree but it is his free forming tree and not a distributed overall root directory.

1) Process: We have a $parent$ directory and wish to create a $child$ directory.

1) Create a unique KEY for the $child$ directory (i.e. $H(random(\#))$) until a unique KEY is found (by checking against $STORE$).
2) In $parent$ we add the entry for the $child$ KEY and send $parent$ to $STORE$.

2) Encryption of directory entries: This uses a process very similar to the *self encrypting data* [2] paper.

This process is as follows

1) Encrypt the $child$ directory as any other file in [2]
2) Create an encryption key for the resulting data map, $H(parent_{KEY} + child_{KEY})$.
3) Create an obfuscation block, $H(child_{KEY} + parent_{KEY})$.
4) Repeatedly concatenate obfuscation block until equal to data map size.
5) XOR data map with result of 4
6) Encrypt Data map
7) $\text{SymEnc}_{[\text{H}(\text{parent}_{\text{KEY}}+\text{child}_{\text{KEY}})]}$ (obfuscated DM)
8) the data map is then digitally signed and sent to $STORE$

B. Advantages of distributed directory

With the advent of distributed directories then several issues clear themselves up.

- We can have an almost infinite traversal of directories now with no limit on depth
- To share data all that is required is sharing a location of a directory and then all the directories below that are automatically shared
- directories now follow a distributed paradigm and are more suited for mass distribution

C. Data locks

If the $STORE$ is a file-system or database then data locks are a standard feature. Here we discuss locking in a Distributed Hash Table (DHT) which is problematic. This section assumes a DHT of a similar capability as *maidsafe distributed hash table [5].*

To ensure writes to data are atomic we require a locking mechanism that is solid and allows the network to recover from stale locks, which tend to be an issue. In maidsafe DHT this can be achieved quiet simply and this is efficient due to the speed of the network, via managed connections.

To write data a node requires to request a lock from the κ closest nodes to the data (this requires no dead nodes to exist in the DHT or bad things will happen). On receiving a lock each node will confer with the other nodes, if all accept the lock then it is in place. If there is any collision both requests for a lock are rejected. In this case the nodes will back-off for a random time and try again.

On receiving a lock a node will read the data again, to confirm it is the same version that has been updated and will then update the value.

There should be a system wide lock duration *constant* in place that will remove any locks that have gone stale (as they will).

1) Securing the shared data: to ensure that only people allowed to amend data the κ nodes will confirm the signature of any lock request and consequent data amendment requests with the owner of the key that signed the value in place. If this matches the process continues, if the signature validation fails the requests are silently dropped.

D. Private shared directory structure

To create a private shared directory structure now is very simple. It follows a process which uses technology as described in *"peer to peer" Public Key Infrastructure*[6] as follows.

1) Create a share:

1) Manager creates $SHARE_{pub}$ & $SHARE_{priv}$
2) $H(SHARE_{pub} + Sig_{SHARE_{prv}})$ is stored on $STORE$, this is called the share ID.

The directory entries for the share are now signed with the share ID and sent to $STORE$.

2) Join a share:

1) USER requests permission from manager to join XXX share, via an encrypted message (signed) to $MANAGER$
2) For read only access the MANAGER send the user the KEY and password for the share root directory
3) For admin access MANAGER sends $SHARE_{priv}$ to USER via an encrypted message

E. Public shared directory structure

Each user may have a public directory structure. This is a very simple version of the above and includes the signature of the directory entry but no encryption. This way anyone can see the data but only the owner can edit it, requiring no locks of course.

In a system where users have an ID then the root directory should be the hash of the username, then anyone can publish information and it will be found by anyone who merely knows their username. Data will be passed around and browsers addons to find the hash of a username can be created to allow widespread access to data on public shares.

F. Anonymous directory structure

A user who creates a directory structure with an anonymous created key pair can publish information anonymously

and pass the root KEY to anyone via any mechanism. This is untraceable, especially if a new key pair were created for each 'share'.

III. Conclusions

This paper has introduced a method of storing data in a distributed network in a manner that is addressable, searchable and very scalable. It is apparent that such system s could in fact supplement or more, the existing world wide web for data sharing. It is not difficult to see that applications that make use of massively shared data and data presented on a native format to users would be an exciting proposition.

References

[1] David Irvine, self authentication, david.irvine@maidsafe.net
[2] David Irvine, Self Encrypting Data, david.irvine@maidsafe.net
[3] David Irvine, peer to peer public key infrastructure, david.irvine@maidsafe.net
[4] David Irvine, maidsafe.net, a new network paradigm , david.irvine@maidsafe.net
[5] David Irvine, maidsafe distributed hash table, david.irvine@maidsafe.net
[6] David Irvine, "peer to peer" Public Key Infrastructure, david.irvine@maidsafe.net

David Irvine is a Scottish Engineer and innovator who has spent the last 12 years researching ways to make computers function in a more efficient manner.

He is an Inventor listed on more than 20 patent submissions and was Designer / Project Manager of one of the World's largest private networks (Saudi Aramco, over $300M). He is an experienced Project Manager and has been involved in start up businesses since 1995 and has provided business consultancy to corporates and SMEs in many sectors.

He has presented technology at Google (Seattle), British Computer Society (Christmas Lecture) and many others.

He has spent many years as a lifeboat Helmsman and is a keen sailor when time permits.