Stackable File System Using NFS Ganesha

Arun Olappamanna Vasudevan, *CS, Stony Brook University,*

***Abstract*—NFS Ganesha is a userland implementation of Network File System. This article explores stackable file system feature supported by NFS Ganesha version 2.0. A file-encrypting stackable file system CRYPTFS.**

***Keywords*—network file system, file system, file system abstraction layer, proxy, NFSv4, NFS, FSAL, stackable**

I. Introduction

GANESHA NFS being a userland implementation of Network File System protocol is a convenient place to try out new features and algorithms. Duo Yi and Youlong Cheng implemented security features in proxy server [1] using Ganesha NFS. The implementation is mostly in NFSv4 protocol, which makes it highly inflexible and non-modular. Stackable FSAL might help implement features in a cleaner way.

Stackable FSAL was explored by implementing a basic file-encrypting stackable file system in NFS Ganesha version 2.0.

II. Background

The NFSv4 Security Proxy project involved security features - antivirus, logging, and filtering based on policy. Implementation was done at development stages of NFS Ganesha version 2.0. To match the latest version of NFS Ganesha, the security proxy project had to be ported.

Implementation was done in NFSv4 protocol layer introduced difficulty in porting – changes in NFSv4 operations rename, open, and close had modifications in the way access mode was checked. TODO: What exactly is change.

Implementing in NFSv4 layer also makes it inflexible. Irrespective of whether FSAL_PROXY is used, these features are present. Using stackable FSAL module is a solution.

III. Project Design

In order to design stackable file system, an overview of NFS Ganesha architecture is discussed first.

A. FSAL - File System Abstraction Layer

There are several types of file systems supported by Ganesha. The FSAL is an abstraction layer that abstracts each file system to a set of common operations that are handled by each module.

Each module can be configured using special key-value pairs in configuration file that is parsed by NFS Ganesha. For instance, FSAL_PROXY required Srv_Addr to be defined in configuration file.

Arun O. V. is a graduate student with the Department of Computer Science, Stony Brook University, Stony Brook, NY, 11079 USA e-mail: aolappamanna@cs.stonybrook.edu, web: http://www.fsl.cs.sunysb.edu/~arunov/

1) PROXY FSAL: This is an abstraction that implements proxy machine between a client and a server. FSAL_PROXY uses NFSv4. Figure 1 shows how requests from clients are handled by FSAL_PROXY by contacting server.

1) Init module invokes config parser
2) 'proxy.ganesha.conf' is parsed
3) Library mentioned as value of 'FSAL_Shared_Library' is loaded
4) dlopen() invokes constructor that registers FSAL of library
5) FSAL added to fsal_list, call backs registered for configuration (init_config) and creating export (create_export)
6) EXPORT block in config file corresponds to a line in /etc/exports in Linux. Export entry is created with the FSAL mentioned in EXPORT block. In this case PROXY.
7) Proxy operations are registered with the export entry (function pointers for read, write, open, close, etc.)
8) After config file is parsed, all FSALs in fsal_list are initialized
9) Initialization of FSAL Proxy
10) Configuration items specific to FSAL Proxy, such as server address (Srv_addr) is obtained from config parse tree
11) Thread spawned to connect to server and initialize RPC
12) RPC socket that's used for send RPC request and listen
13) Worker threads spawned for handling requests from client
14) NFSv4 request from client received by worker thread
15) After access permission (supports even authentication) checks of client on export entry corresponding to request, NFSv4 service function (usually nfs4_Compound) is called
16) All metadata is maintained in cache inode and dentry data structures in AVL trees. Request received from NFSv4 operations.
17) Request is forwarded to FSAL operations registered to particular export entry
18) Server request through RPC socket
19) NFSv4 request to Server
20) NFSv4 response from Server
21) Response passed to PROXY handler
22) Response passed to cache inode
23) Response passed to NFSv4 protocol handler
24) Response passed to worker thread
25) svc_sendreply() to client

2) Stackable FSAL: Every EXPORT block in config file has a FS_Specific entry that is passed to create_export() function of FSAL of the export entry. The FS_Specific entry is again an FSAL name. This opens possibilities for using the FS_Specific

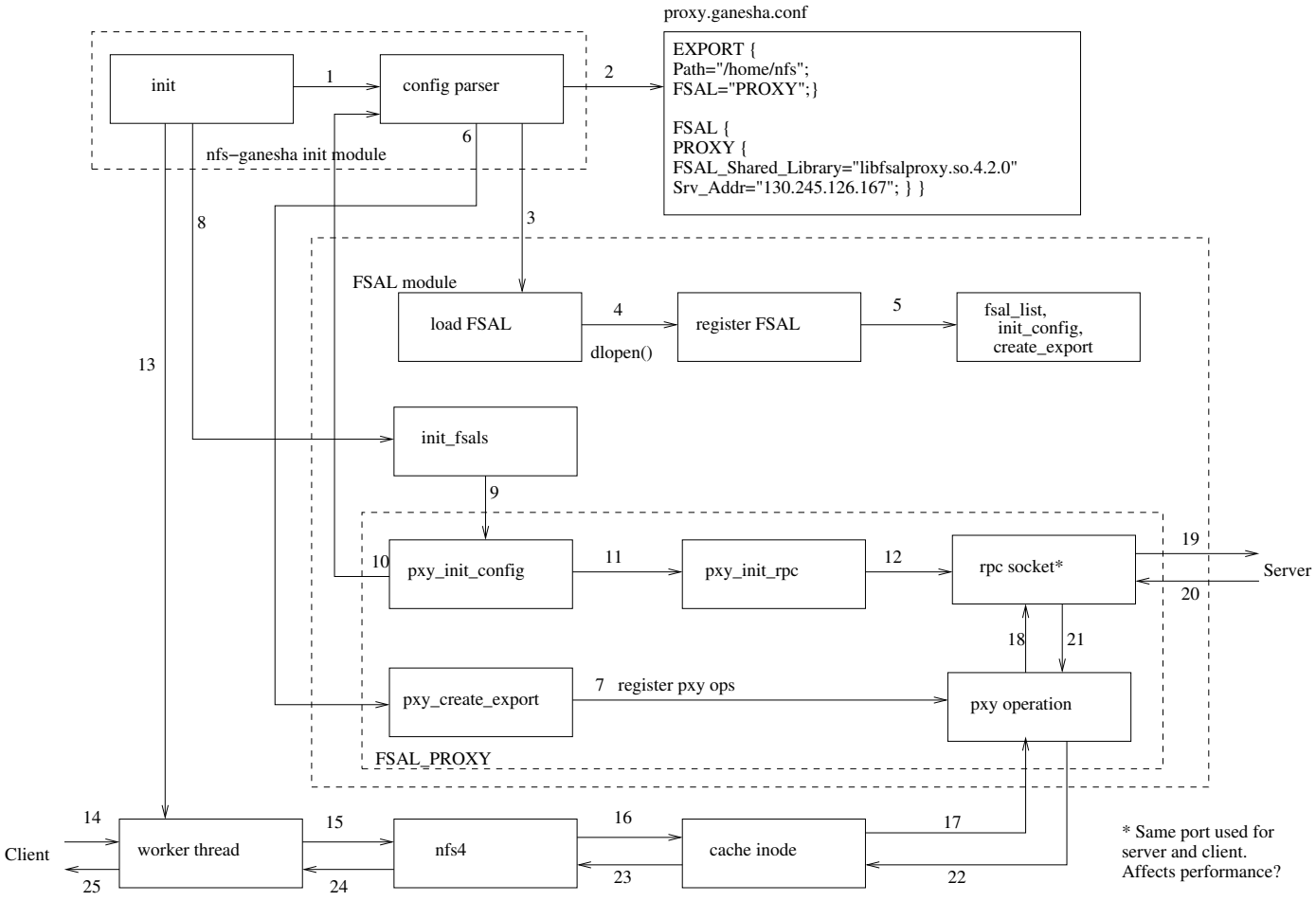

Fig. 1. Architecture of FSAL_PROXY

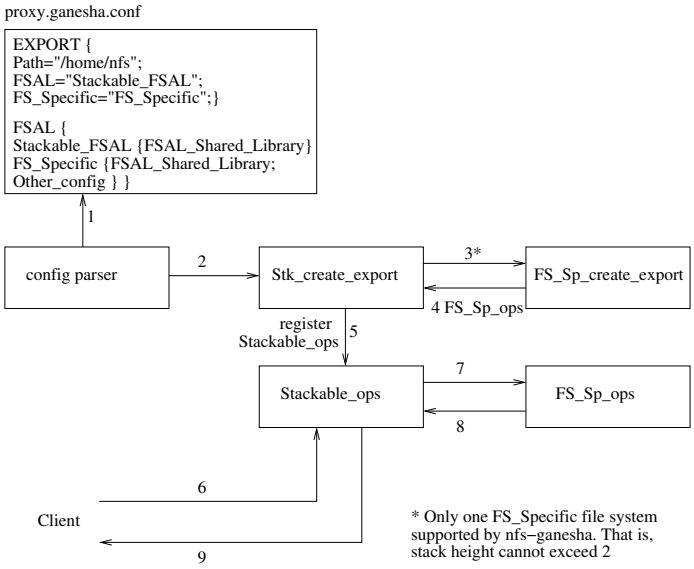

Fig. 2. Architecture of Stackable FSAL

for stacking on top of a particular FSAL. Figure 2 shows architecture and working of a general stackable FSAL.

1) Config file parsed
2) EXPORT block corresponds to Stackable FSAL. Stackable FSAL's create_export() is called with a parameter FS_Specific.
3) FS_Specific is another FSAL module. It's create_export() method is called.
4) Handlers of FS_Specific FSAL is returned
5) Stackable FSAL handlers are registered for the export entry
6) When a client request comes, Stackable FSAL handler is called
7) After performing its own operation, FS_Specific FSAL's handler is called
8) Response from FS_Specific FSAL handler
9) Response passed back to client

B. *CRYPTFS Stackable file system*

CRYPTFS is a stackable FSAL module that secures data using cryptography. CRYPTFS when used in a proxy machine

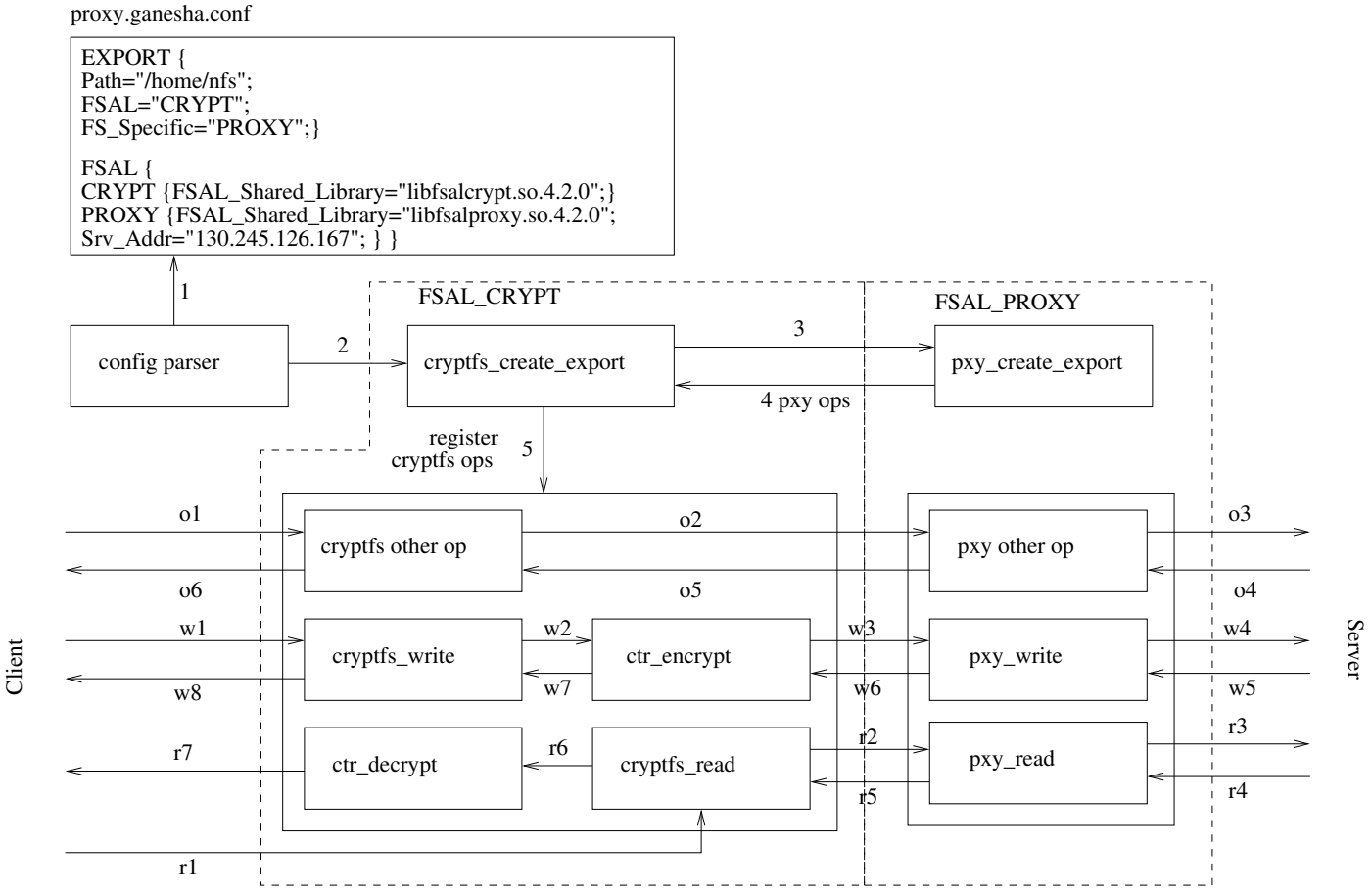

Fig. 3. Architecture of CRYPTFS stacked on PROXY

encrypts data that is written to server and decrypts data when read from server. Architecture of CRYPTFS in proxy is as shown in figure 3.

Control flow is similar to a general stackable FSAL. All operations are merely wrappers that call underlying PROXY FSAL except read and write. Write involved DES encryption in counter mode (see figure 4) and read involves decryption (see figure 5).

IV. Conclusion

- To use stackable FSAL for a NFSv4 Security features can be implemented by using a File Handle to File Name Map in stackable FSAL. After digging cache inode data structures, I think they are right that file name cannot be retrieved from file handle. Needs to be seen if there are other ways of doing this (possibly using cache inode data structures).
- The existing restriction on FSAL Stackable to have only one underlying file system can be worked around by compromising on flexibility (yet to be checked). Also, I think it's worth exploring a change in config parsing to allow multiple layers. Further investigation needs to be done on this.

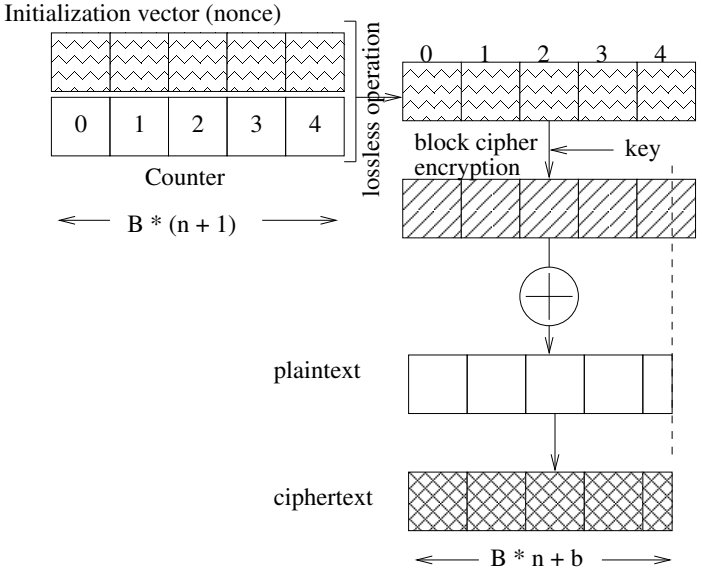

Fig. 4. Encryption in CRYPTFS

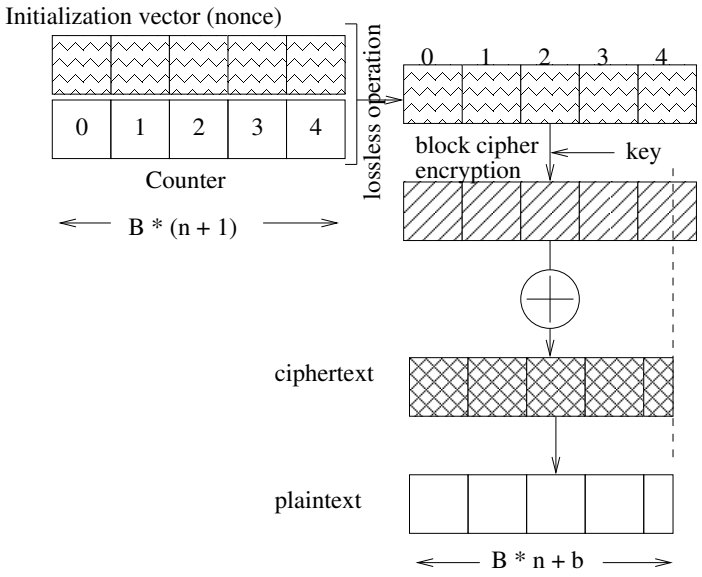

Fig. 5. Decryption in CRYPTFS

- Implementation of CRYPTFS has some performance improvements that I have identified. Also, impact on performance by inclusion of CRYPTFS stackable file system is yet to be measured.
- Performance of NFS Ganesha Proxy with different ports of communication with server and client machines is also worth study.

ACKNOWLEDGMENT

Ming Chen has been extremely supportive through the duration of work on NFS Ganesha and Stackable FS.

REFERENCES

[1] D. Yi and Y. Cheng, “Design and implementation of a nfsv4 security proxy,” Aug. 2013.