Enhancing NFS Cross-Administrative Domain Access

Joseph Spadavecchia and Erez Zadok
Stony Brook University
`{joseph,ezk}@cs.sunysb.edu`

Abstract

The access model of exporting NFS volumes to clients suffers from two problems. First, the server depends on the client to specify the user credentials to use and has no flexible mechanism to map or restrict the credentials given by the client. Second, there is no mechanism to hide data from users who do not have privileges to access it. Although NFSv4 promises to fix the first problem using universal identifiers, it does not provide a mechanism for hiding data and is not expected to be in wide use for a long time.

We address these problems by a combination of two solutions. First, *range-mapping* is a mechanism that allows the NFS server to restrict and flexibly map the credentials set by the client. Second, *file-cloaking* allows the server to control the data a client is able to view or access beyond normal Unix semantics. Our design is compatible with all versions of NFS, including NFSv4. We have implemented this work in Linux and made changes only to the NFS server code; client-side NFS and the NFS protocol remain unchanged. Our evaluation shows a minimal average performance overhead and, in some cases, an end-to-end performance improvement.

1 Introduction

NFS was originally designed for use with LANs, where a single administrative entity was assumed to control all of the hosts in that site and create unique user accounts and groups. The access model chosen for exporting NFS volumes was simple but weak. In a different administrative domain, the password database may define different users with the same UIDs; a UID clash could occur if files in one domain are accessed from another. Worse, users with local root access on their desktops or laptops can easily access files owned by any other user via NFS, by simply changing their effective UID (i.e., using `/bin/su`).

Therefore, NFS servers rarely export their volumes outside their administrative domain. Moreover, administrators resist opening up access even to hosts within the domain, if those hosts cannot be controlled fully. Today, users and administrators must compromise in one of two ways. Either volumes are exported across administrative domains and security is compromised, or the volumes are not exported across administrative domains, preventing users from accessing their data. Neither solution is acceptable.

Current NFS servers implement a simple form of security check for the super user, intended to stop a root user on a client host from easily accessing any file on the exported NFS volume. However, current NFS servers do not allow the restriction and mapping of any number of client credentials to the corresponding server credentials. We provide a mechanism for globally restricting access and hiding data on the server.

We present a combination of two techniques that together increase both security and convenience: range-mapping and file-cloaking. *Range-Mapping* allows an NFS server to map any incoming UIDs or GIDs from any client to the server's own known UIDs and GIDs. This allows each site to continue to control their own user and group name-spaces separately while allowing users on one administrative domain to access their files more conveniently from another domain. Range-mapping is a superset of the usual UID-0 mapping and Linux's *all-squash* option which maps all UIDs or GIDs to –2.

Our second technique, *File-cloaking*, lets the server determine which ranges of UIDs or GIDs should a client be allowed to view or access. We define *visibility* as the ability of an NFS server to make some files visible under certain conditions. We define *accessibility* as the NFS server's ability to permit some files to be read, written, or executed. Cloaking extends normal Unix file permission checks by allowing administrators to restrict the visibility and accessibility of users' files when those files are exported via NFS. Cloaking is a superset of the NFS server export options `nosuid` and `nosgid` which prevent the execution of set-bit files.

Range-mapping and cloaking complement each other. Together, they allow NFS servers to extend access to more clients without compromising the existing security of those files. Whereas ACLs can allow a greater degree of flexibility than cloaking, ACLs are not available on all hosts and all file systems, are not supported in NFSv2, and are partially implemented in NFSv3. Furthermore, ACLs are often implemented in incompatible ways; this is one reason why the new NFSv4 protocol specification lists ACL attributes on files as optional.

Our system is implemented in the Linux in-kernel NFS server. No changes were made to the NFS client side and our system is compatible with existing NFS clients. This has the benefit that we can deploy our system fairly easily by changing only NFS servers. Our source code is Open Source Software and protected under the GPL.

We performed a series of general-purpose benchmarks and micro-benchmarks. Range-mapping has an overhead of at most 0.6%. File-cloaking overheads range from 72% for a large test involving 1000 cloaked users—to an *improvement* of 26% in performance under certain conditions, reflecting a 4.7 factor reduction in network I/O.

2 Design

Range-mapping and cloaking are features that offer additional access-control mechanisms for exporting NFS volumes. These features were designed with three goals: compatibility, flexibility, and performance.

We maintain two types of compatibility. First, we are compatible with all NFS clients by requiring no changes to them. Range-mapping and cloaking are performed entirely by the NFS server. The modifications to the server force range-mapping and cloaking behavior on the clients. Second, we maintain compatibility with standard Unix semantics. We specify range mappings and file cloaking using a syntax that leaves traditional Unix behavior the default.

We provide additional flexible access-control mechanisms that allow both users and administrators to control who can view or access files. We allow administrators to mix standard Unix and cloaking semantics to define new and useful policies such as making all world-writable files inaccessible, hiding all files for which a user has no group-read access to, and more.

Both range-mapping and cloaking are defined on the server. NFS clients do not contain range-mapping or cloaking configuration information. This satisfies the compatibility design goal that clients remain unmodified (even in configuration).

2.1 Range-Mapping

Range-mapping is a mechanism for unifying localized ID namespaces in a distributed environment. A range-mapping definition is composed of one or more *range-maps*. An NFS range-map takes a contiguous range of IDs on the client and maps them to a contiguous range on the server. IDs can be either UIDs or GIDs. To be flexible, two types of mappings are allowed: N-to-N, and N-to-1. An N-to-N range-map allows N contiguous client IDs to be mapped to N contiguous server IDs. Such a range-map is convenient because many modern user management systems allocate IDs in contiguous blocks. An N-to-1 mapping is a non-bijective mapping from a contiguous range of N client IDs to a single server ID. This type of mapping is typically useful for security purposes.

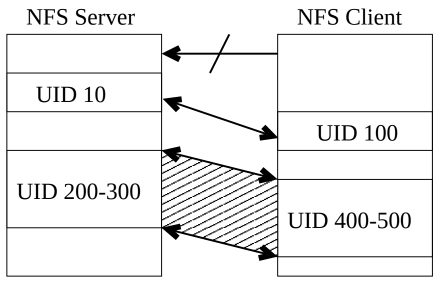

Figure 1: Range-mapping client UID 100 to server UID 10 and client UIDs 400–500 to server UIDs 200–300. All other UIDs are restricted.

Range-mapping is done in bidirectionally: *forward range-mapping* and *reverse range-mapping*. Forward mapping is done when a client sends a request to the NFS server and the server maps the user's client UID and GID to the corresponding server UID and GID. Reverse mapping is done when the server responds to the client (i.e., when reading files from disk) and must map the user's server UID and GID back to the corresponding client UID and GID.

2.2 File-Cloaking

Cloaking is a mechanism that expands visibility and access-control policies. A cloaking definition consists of one or more *cloak-lists*. A cloak list is an unordered list of *ID range* and *cloak-mask* pairs. An ID-range is a contiguous range of server UIDs or GIDs, and the cloak-mask is a bitmap defining what type of cloaking policy to enforce on those IDs. Allowing ranges of IDs to be specified makes it convenient to define cloak lists. The cloak-mask adds flexibility to the system by allowing standard Unix and cloaking semantics to be coalesced.

File-cloaking abstracts the concept of file permissions to allow the distinction between user and administrator access-control. There are three default file-cloaking rules. First, files are always visible to their owners. Allowing files to be hidden from their owners is not useful semantics. Second, files that are not visible are not accessible. If a hidden file can be accessed then its existence can be verified. Third, files without any permissions are only visible to the owners. This rule keeps users from circumventing cloaking by removing all permissions. These rules mean the following: accessibility implies visibility and invisibility implies no accessibility (by the second default rule of file-cloaking).

File-cloaking entries include a 9-bit *mask*. The mask corresponds to the following bits: SETUID, SETGID, sticky, group read, group write, group execute, other read, other write, and other execute bits, respectively. This mask is logically ANDed with the UNIX mode bits on the file. If the AND succeeds then the file is hidden (i.e., it is cloaked).

The following example shows a cloaking definition:

```
/home  *.example.com(rw, \
                     cloak_list = \
                     uid  777    0  1000 \
                     gid  700  100   200)
```

In this example, there are two cloaking definitions. The first places the restriction that only UIDs 0–1000 may see or access their files. That is, files on the server owned by UIDs 0–1000 cannot be seen or accessed by any user other than the owner. The second definition states that files owned by UIDs 100–200 are only visible if there is world access to them or if there is group access to them, and the user listing the file belongs to the group of the file.

3 Evaluation

We ran our benchmarks between an unmodified NFS client and an NFS server using five different configurations set to illustrate worst-case behavior:

1. **VAN:** A vanilla setup using an unmodified NFS server.
2. **MNU:** Our modified NFS server with all of the new code included but not used.
3. **RMAP:** Our modified NFS server with range-mapping configured in `/etc/exports`.
4. **CLK:** Our modified NFS server with cloaking configured in `/etc/exports`.
5. **RMAPCLK:** Our modified NFS server with both range-mapping and cloaking configured in `/etc/exports`.

3.1 General Purpose Benchmark

Figure 2 shows the results of our am-utils compile benchmark. This benchmark exercises a wide variety of file operations, but the operations affected the most involve getting the status of files (via `stat(2)`), something that does not happen frequently during a large compilation; more common are file reads and writes. Therefore the effects of our code on the server are not great in this benchmark, as can be seen from the individual results.

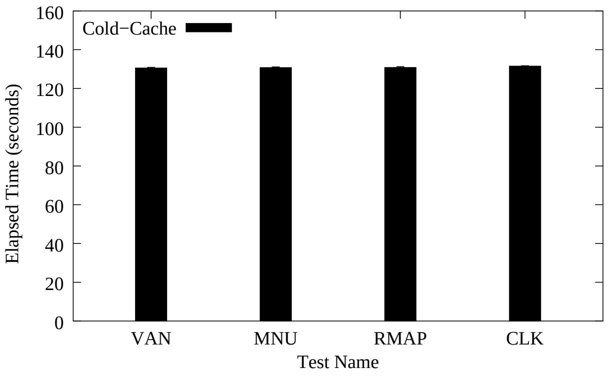

Figure 2: Results of the am-utils benchmark show a small difference between all of the tests—less than 1% between the fastest and slowest results.

Our benchmarks here show an overall performance difference of less than 1% between the best and worst tests. These results suggest that range-mapping and cloaking have a small effect on normal use of NFS mounted file systems.

3.2 Micro-Benchmark Results

Figure 3 shows results of a our first micro-benchmark: a recursive listing (`ls -lR`) we conducted on an NFS-mounted directory containing 1000 entries. The `/etc/exports` file on the server was configured with range-mapping and cloaking such that the user listing that directory on the client would always see all files. This ensures that the amount of client-side work, network traffic, and server-side disk I/O for accessing the directory remained constant, while making the server's CPU experience different workloads.

When our code is included but not used (MNU vs. VAN) we see a 17.6% degradation in performance. This is due to the need to check to see if range-mapping or cloaking are configured for this client. Although simple, this these checks reside in a critical execution path of the server's code—where file attributes are checked often. Adding range-mapping (RMAP vs. MNU) costs an

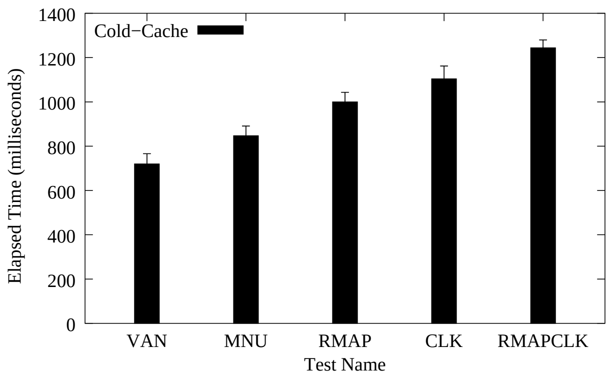

Figure 3: ***Results of recursive listing of directories containing exactly 1000 entries on the server. The tests were configured to result in exactly 1000 files being returned to the client each time (i.e., worst case).***

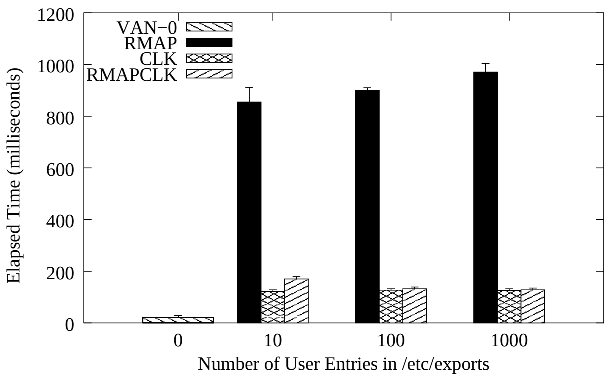

Figure 4: ***Results of recursive listing of directories containing a different number of files while the server is configured with a different number of mapped or cloaked entries.***

additional 18.1% in performance. This is due to scanning of possibly long linked lists and changing credential numbers. The cloaking test (CLK vs. MNU) costs an additional 30.3% in performance. This is due to the fact that, to ensure compatibility with all NFS clients, cloaking must guarantee that NFS clients do not use cached file attributes. Therefore the client gets from the server all of the files each time it lists the directory. Finally, the worst-case situation (RMAPCLK vs. VAN) has an overhead difference of 72.3%.

The micro-benchmarks in Figure 3 show the worst case performance metrics, when both the server and client have to do as much work as possible. The next set of micro-benchmarks was designed to show the performance of more common situations and how our system scales with the number of range-mapped or cloaked entries used. These are shown in Figure 4.

The range-mapping bars (RMAP) show the performance of listing directories with 1000 files in them, but varying the number of users that were range-mapped. For example, range-mapping with 10 users implies that each user owns 100 files. Given two orders of magnitude difference for the number of entries used for the three RMAP bars, the overall overhead difference is just 13.6%.

The bars for cloaked configurations (CLK) show a different behavior than range-mapped configurations. Here, all 1000 files were owned by cloaked users, but the user that listed the files on the client was not one of those cloaked users and therefore was not able to see any of those files; what they listed appeared on the client as an empty directory. This test fixed the amount of work that the client had to do (list an empty directory) and the server's work (scan 1000 files and apply cloak rules to each file). What changed in this test were the number of cloaked user entries in `/etc/exports`. The bars show a small 4.1% performance difference between the largest and smallest lists.

The last set of bars in Figure 4 shows the performance when combining range-mapping with cloaking (RMAP-CLK). Since all of the users' files were cloaked and range-mapped, and the user that listed the directory on the client was one of those users, then that user saw a portion of those files (the files they own). This means that the amount of work performed by the client should decrease as it lists fewer files and has to wait less time for network I/O. The RMAPCLK bars indeed show an *improvement* in performance as the number of cloaked user entries increases. The reason for this improvement is that the savings in network I/O and client-side processing outweigh the increased processing that the server performs on larger cloak lists. Listing the same directory when we use 100 cloaked and range-mapped entries is 22.3% faster than the directory with 10 entries, because we are saving on listing 90 files. Listing the directory with cloaked 1000 entries is only an additional 4% faster because we are saving on listing just 9 files.

To find out how much cloaking saves on network I/O, we computed an estimate of the I/O wait times by subtracting client-side system and user times from elapsed times. We found that for a combination of cloaking and range-mapping with 10 users, network I/O is reduced by a factor of 4.7. However, since cloaking forces clients not to cache directory entries, these immediate savings in network I/O would be overturned after the fifth listing of that directory.

For more information as well as sources to the work described here, see `www.fsl.cs.sunysb.edu`.