Engineering AFS Locker Model – A Technical Overview

This document provides a technical overview of the architecture for the Engineering AFS Locker Model that we have adopted for allocating AFS storage space to faculty, staff, and students in the College of Engineering.

Engineering AFS Locker Model – A Technical Overview 1
"Locker" Defined 2
Locker Quotas 2
Locker Types 3
Why we do what we did when we did what we did, er do 6
The "Webserver" Access Problem 6
Solving the "Webserver" Access Problem 6
Separate filesystem locations for "web" and "non-web" content. 7
Get rid of system:anyuser 7
"Discourage" the locker owner from accidentally using system:anyuser 8
Separate the "user access" path from the "server access" path 8
A Segue Into AFS Volumes 8
Separate the "user access" path from the "server access" path 10
Get rid of the "admin" permission in strategory locations (or How to Avoid the Filesystem-side "Security Hole" with AFS, webservers and IP ACLs) 11
Make sure that server-side programs can't get around the permissions restrictions 12
Where We Go From Here (or the "Great Migration of 2002" Event) 13
Document Change Log 13

"Locker" Defined

A "Locker" is a managed collection of one or more volumes, with standard sub-directories (volume mount points actually), access groups, and permissions. All lockers are reviewed and "renewed" on at least an annual basis. The goals of the "Locker Model" are to keep "private" private and off of "the web"; provide some common access standards and locations; allow new or expanded web services, such as `WRAP` and `PHP`, and still maintain private data areas; and to facilitate management of locker requests and space allocation of ITECS-managed AFS space.

All lockers except for "Website Lockers" (see the Locker Types below) are built using the concept of a "Container" volume. The "container" is just a volume created for the express purpose to hold mount points for other volumes, and any administrative files or logs. Locker owners don't have write or "(a)dmin" permissions at the container level – this helps us maintain the integrity of the AFS permissions (i.e. keep "`system:anyuser`" out of the filesystem – for more details see the discussion on permission issues later in this overview).

This "container" layout has the somewhat odd looking result that we "force" access into a "sub-directory" of where one thinks it ought to be. However, with a little explanation it starts making sense, especially if you have "private" and "public" areas or if we make additional mount points in their container volume (see the layout information below).

Locker Quotas

Lockers themselves don't have a defined quota. However, Locker volumes have a hard quota cap of `1GB`. We'll roll new volumes as space needs increase for any given group, project, department (or summarized: Locker).

Locker owners will be responsible for laying out their content so that the `1GB` "boundaries" within a given locker are kept in mind as space needs increase (e.g. it may not make sense to have a 1GB volume and a 100MB volume – depending on the content it may make more sense to lay things out with two 550MB volumes).

Quotas are allocated using "the computer geek like base2 megabyte" which is that 1 megabyte = 1024 kilobytes = 1024x1024 bytes. The IEEE megabyte standard really is 1 megabyte = 1000 kilobytes = 1000x1000 bytes (see http://physics.nist.gov/cuu/Units/binary.html). However, AFS allocates quota in "blocks" - which are 1024 bytes, but then most quotas are then assigned on a 1 megabyte = 1000 AFS blocks (= 1000x1024 bytes). The real deal for us is that we can tell at a glance if we set the quota (or more accurately that we didn't) on a volume if it's not some funny looking number that's a multiple of 1024 – all our scripts internally multiply the megabyte count by 1024 AFS blocks.

Confusing? Yeah we know - blame binary arithmetic.

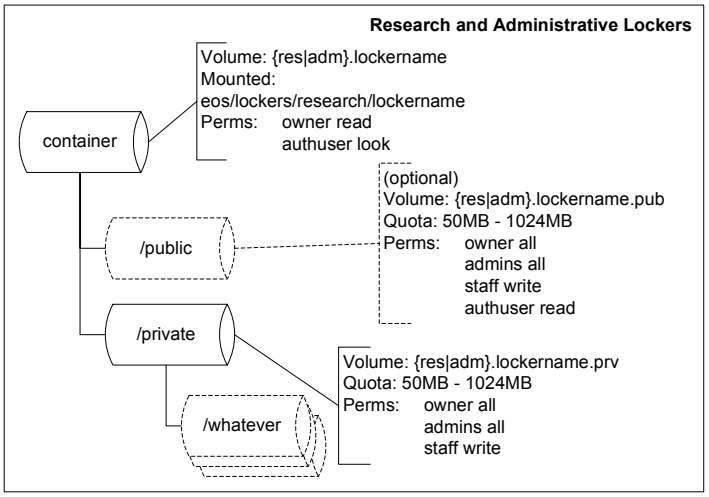
Locker Types

There are five types of lockers that we have identified – each have a different layout and framework and are meant to address different needs. The exception is that "Research" and "Administrative" lockers are just a separation in name based on the intended usage – they look the same internally.

The best resource for those familiar with AFS are the "Locker Specification" sheets which define the standard volumes, names, mount points, groups, and permissions for each of the currently defined Locker "types." These are the reference guides for all the management scripts and tools and for eyeballing that things are correct when examining a locker. Below is a short description and layout of what those specification sheets cover.

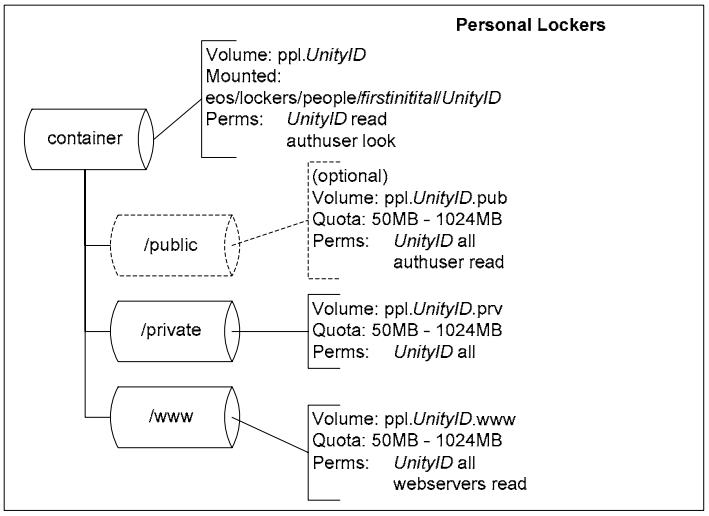
The five defined locker types:

1. **Research**: Lockers for research groups and projects and individual research Research lockers are not web accessible. Located at `/afs/eos/lockers/research`
2. **Administrative**: Lockers for administrative groups, projects, and support space allocation. Administrative lockers are not web accessible. Located at `/afs/eos/lockers/admin`

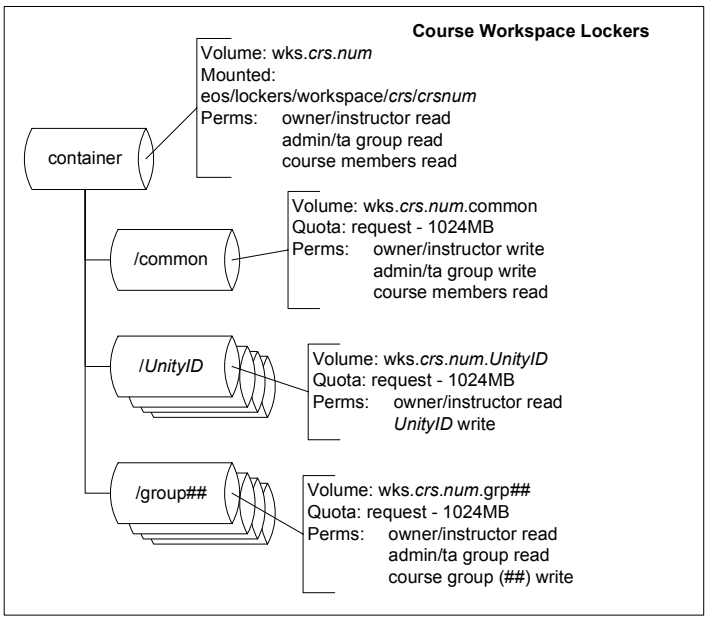


Research and Administrative Lockers have two standard PTS groups associated with them. An "Admin" group that has the "`a`" permission over the content and ownership (usually) of the staff group (by creating it as a "`:`" group of the admin group e.g. `owner-admin:staff`). The staff group has `write` access only. See the specification sheets for the group naming standards.

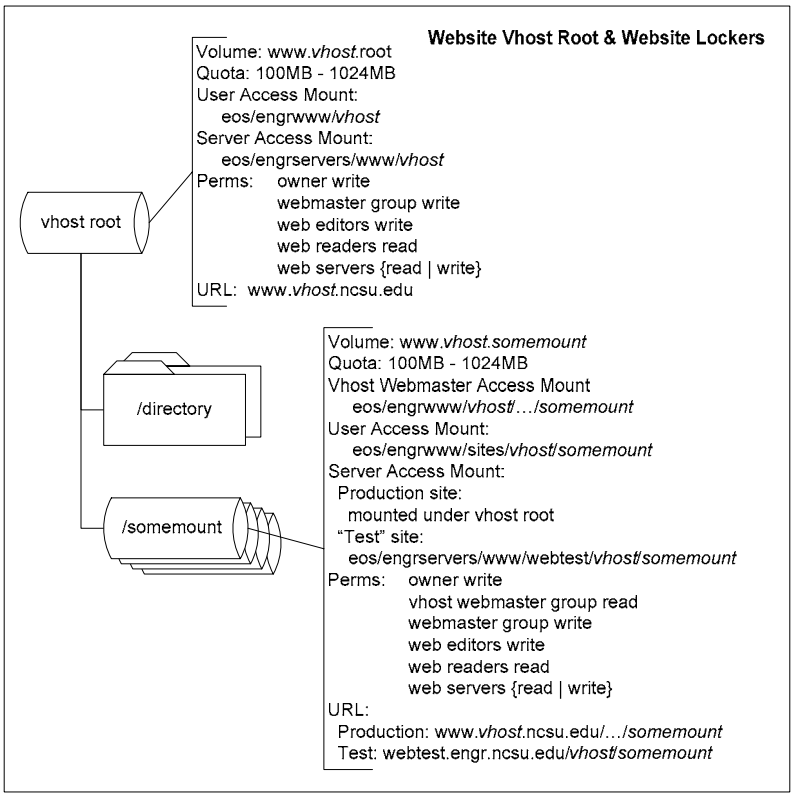
3. **Personal** (or **People**): Personal-use lockers for Engineering Faculty/Staff and potentially students. Personal lockers have directories for non-web accessible and web accessible content. Located at `/afs/eos/lockers/people/`*`firstinitial/UnityID`*



4. **Course Workspace**: Additional space, allocated on a semester-by-semester basis at the request of an instructor to provide additional and augmented quota space for student work and projects. Course Workspace lockers are not architected for section separation at this time.



5. **Website Lockers**: Web-accessible "space" requested and supervised on a vhost-by-vhost basis by college webmasters. Website Lockers are "just space" - but have an access group mechanism to allow different "webmaster" groups that would have responsibility for subsections of a large vhost (e.g. `www.engr.ncsu.edu`)



Website Lockers have several standard PTS groups associated with them. An "Webmaster" group that are the recognized webmasters for the content and have ownership of the editors and readers group (by creating it as a "`:`" group of the admin group e.g. `owner-www:wwwedit`, `owner-www:wwwread`). The editors group has `write` access to the locker, the readers group `read` access. See the specification sheets for the group naming standards.

Why we do what we did when we did what we did, er do

The specification sheets do little to explain the rationale or design behind the mounting and permissions model of the Engineering AFS Locker Model – which is covered in some explanation below – the explanation assumes you are familiar with AFS usage, permissions, and filesystem namespace layout.

The "Webserver" Access Problem

One of the problems with the existing `/afs/eos/*` filesystem namespace is that large sections of `/afs/eos/info` `/afs/eos/service` and `/afs/eos/www` provide for both AFS client access and for webserver access – indiscriminate of the content – and there's no real easy way to separate things in those directories – and we aren't really all that sure who was what where (does that sentence sound confusing? It does to me, and I wrote it - that pretty well describes the situation in itself).

Webserver access is traditionally done by pointing webservers with AFS clients on them at `/afs/eos/`*`something`* and setting *`something`* to be `system:anyuser` read. The access mechanism for ITD and ITECS-managed webservers later became based on making PTS entries for the tcp/ip addresses for the webservers and including them in standard PTS groups and ACL's within the filesystem. There is still a great deal of `system:anyuser read` in the ACLs throughout the namespace – meaning any old workstation with an AFS client and something listening on port 80 that knows about `/afs` (or for that matter `\\system-afs\all` on a Windows-based machine) can deliver content out of AFS.

That's great – until you want to do something with `WRAP` or provide some service guarantees that the content in location `/afs/eos/`*`something`* is not delivered out to Tippancanoe or ResNet or the office next door via a webserver or any "workstation running a process on port 80 and an AFS client".

And because we have so much system:anyuser look or system:anyuser read – because of the way AFS permissions are copied to child directories anytime you create a new directory in any given volume – we have a double problem of needing to clean up permissions in a lot of places if we even figured out a way to keep content where it was.

Solving the "Webserver" Access Problem

So to solve this access problem – we have to architect a few things. First, separate the "private" or "authenticated access-only" content (not meant to be delivered via http) from the "web-deliverable" content. Second – provide some means for providing limited access web content (ideally using WRAP) and server-side scripting solutions (cgi, php, etc.)

A new architecture also gives us a chance to start the directory structures over with a cleaner set of permissions, and thereby have *those* permissions copied (or inherited if you will, but it's not true inheritance) with newly created subdirectories.

So based on what we know – we need to do a few things:

1. Separate filesystem locations for "web" and "non-web" content.
2. Get rid of `system:anyuser`
3. "Discourage" the locker owner from accidentally using `system:anyuser`
4. Separate the "user access" path from the "server access" path
5. Get rid of the "admin" permission in strategory locations
   (or How to Avoid the Filesystem-side "Security Hole" with AFS,
   webservers and IP ACLs)
6. Make sure that server-side programs can't get around the path and permissions
   restrictions

Okay. No problem. Those "rules" give us the framework for the resultant Locker model.

Separate filesystem locations for "web" and "non-web" content.

We've done this by *almost* separating completely the user access points for "web" and "non-web" content in the filesystem.

All website lockers are accessed at `/afs/eos/engrwww/`

All Research, Administrative, Workspace, and Personal lockers are accessed at
`/afs/eos/lockers/{research|admin|workspace|people}`

The one exception (so far) to this separation is that Personal Lockers have a sub-volume for web content that is useful to keep associated from an user access point by having those mounted at
`/afs/eos/lockers/people/`*`UnityID`*`/www`

The www volumes in the personal lockers aren't tied to a review by a vhost webmaster (though technically there is a vhost webmaster group and vhost root website locker for the `http://people.engr.ncsu.edu` vhost).

Get rid of system:anyuser

Getting rid of `system:anyuser` includes two major activies.

1. We remove it from the user documentation and support "vocabulary" and reduce the "mindshare" of using it in a new Engineering AFS Locker.
2. We don't include it in the default permission set of the newly created lockers. We've architected things to depend on `system:authuser` and owner-managed access groups

We need to take this a step further though, and architect something to "discourage" the use of `system:anyuser` at all in the filesystem.

"Discourage" the locker owner from accidentally using system:anyuser

We've architected the locker layout to discourage the use of `system:anyuser` at all. A locker owner can apply it to his or her files – but at least through the standard ITECS provided User Access Paths – `system:anyuser` won't be able to get to Locker content.

By using container volumes at the top of most lockers, or removing the "`a`" permission from user use in the Website lockers – we effectively wall off `system:anyuser` out of the ITECS-managed filesystem. At the root mount points for the hierarchy for each of the standard locker types: (`/afs/eos/lockers/{research|admin|people|workspace}`) the permissions are limited to `system:authuser look` (and include the standard `system:administrators all` permissions and permissions for our auditing account – using the `itecs-admin:lockeraudit` group).

Additionally, our auditing scripts will flag any use of `system:anyuser` and send email to the Locker owners asking them if they *really* intended to assign `system:anyuser` access to their Locker space.

It's still possible to bring up a webserver running under an account that has tokens associated with it and the webserver can then walk down `system:authuser` paths. If someone figures out how to do that and makes it work – really work – first we'll bust them for account sharing (then likely hire them for their ingenuity).

It's also possible to play some mounting games with the volumes elsewhere, which we mitigate in the ITECS-managed filesystem namespace by separating "user" access from "server" access.

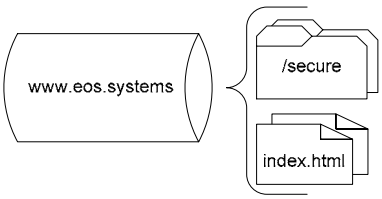
Separate the "user access" path from the "server access" path

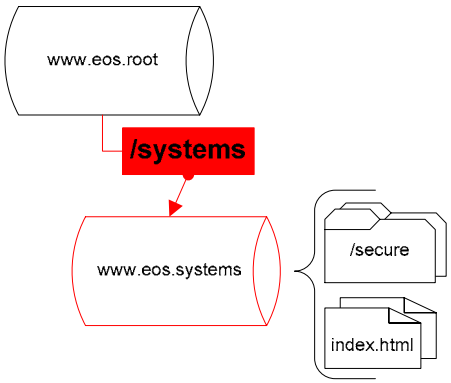
Volumes and volume mounts are the biggest strength of our AFS infrastructure. However, they also present biggest challenges in exporting sections of `/afs/eos` out to the web and still maintaining integrity of the security of the filesystem with regard to private vs. non-private content and with regard to Web security methods like WRAP.

A Segue Into AFS Volumes

Before we get too far into talking about where user access is separated from server access – it's important to understand why. Yes this is likely a lot of verbiage and potentially quite boring for the experienced AFS administrator – but it's pretty important to go over this. This is where some of the hoops that we jump through with the Engineering AFS Locker Model come from (the "why" for the hoops will really become apparent in the "Get rid of the 'admin' permission" section).

An AFS Volume is a self-contained "container" of files and directories (and thereby permissions) Any given volume lives on a single server and AFS partition on that server – AFS manages the volume itself – it's written on top of UFS – but in such a way that normal "ufs" commands (ls, etc.) don't see actual files that represent the volumes (they do display some header/tracking files).





Volumes can contain mounts/references for other volumes.

Consider the Website Vhost Root Locker for the Eos website. The Website Vhost Root Locker will normally contain the directories and files delivered out from the www.eos.ncsu.edu vhost. However – www.eos.ncsu.edu is a "large" web vhost and different groups and projects will want to have their own "websites" coming out from underneath the `www.eos.ncsu.edu` vhost name – with their own "webmasters."

The ITECS/Systems group will have their own web "site" delivered as a sub-URL of `www.eos.ncsu.edu` – so we've created a website locker for the group, named `www.eos.systems`.

The vhost webmasters for the www.eos.ncsu.edu vhost have approved the URL `www.eos.ncsu.edu/systems` for this locker and have approved that it can go production on the web without delivery through our test vhost first. So in order to mimic the "URL" paths and cut down on the number of Apache aliases we have to create - the `www.eos.systems` website locker is mounted at `/systems` within the vhost root locker for the `www.eos.ncsu.edu` vhost.

For Website Lockers a locker = a volume. And wherever the `www.eos.root` volume is mounted, the `www.eos.systems` volume will "go with it" - because like Prego™ "it's in there (in the `www.eos.root` volume).

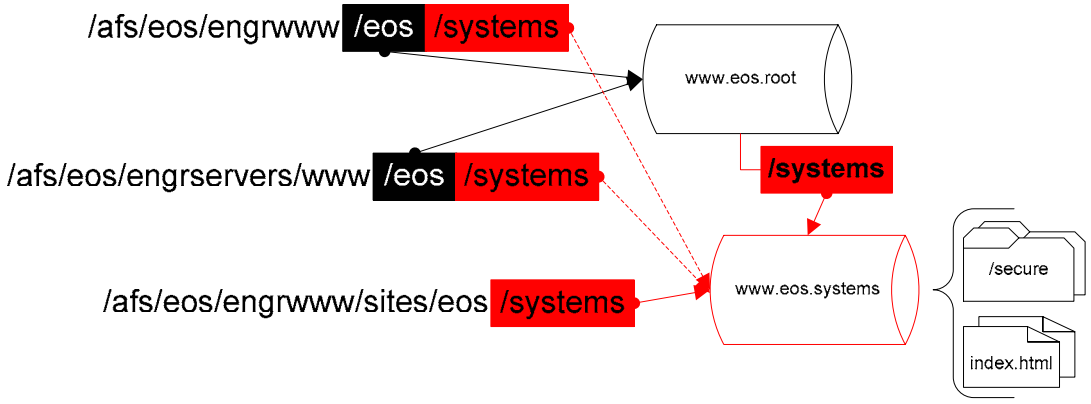
Pretty powerful eh? Well - volumes can be mounted in more than one place. This gives us a great deal of flexibility in segregating access to volumes (and dealing with a potential administration headache – there's no list kept of where a given volume is mounted by the AFS servers – so you have to either limit mount points or have standards – we chose standards – that still won't stop arbitrary mounting – which is what the auditing scripts will also need to check).

So returning from our reverie on AFS Volumes, let's finally talk about the user access and server access separation.

Separate the "user access" path from the "server access" path

This "multiple mounting" functionality gives us the opportunity to separate the "user access" path from the "server access" path.

The "`www.eos.root`" volume (Website Vhost Root Locker) is mounted at `/afs/eos/engrwww/eos` and `/afs/eos/engrservers/www/eos`. Because the `www.eos.systems` volume (Website Locker) is mounted at /systems within the `www.eos.root` volume/locker – it's "immediately" available as `…/engrwww/eos/systems` and `…/engservers/www/eos/systems`.



In the Engineering AFS Locker Model – the Website Locker "user access" path (i.e., it has `system:authuser` look down to the point that the owners/webmasters have permissions) is `/afs/eos/engrwww`. The "server access" path (i.e. where the web servers have look permissions down to the root of the Website Lockers) is `/afs/eos/engrservers/www`

One additional mount point for the `www.eos.systems` volume is at `/afs/eos/engrwww/sites/eos/systems` – this provides the "webmasters" for the systems subsite a path to get at their content without needing permissions in the Vhost Root Locker for `www.eos.ncsu.edu` (the systems website might actually be the URL `www.eos.ncsu.edu/we/aregoing/to/kalamazoo/systems`, and be mounted as such in the `www.eos.root` volume but it is still going to be named `www.eos.systems`, and will still be mounted at `/afs/eos/engrwww/sites/eos/systems` – so we may have some naming arbitration to deal with at a later date).

So far we've only covered Website Lockers. Research, Administration, Course Workspace, and Personal lockers are decidedly simpler – no servers will point into the `/afs/eos/lockers/{research|admin|workspace|people}` spaces. We use the multiple-mount functionality to create a second mount point for the Personal Locker "www" sub-volume that the webservers can access at `/afs/eos/engrservers/www/people/`*`UnityID`*. The "user access" path remains `/afs/eos/lockers/people/`*`firstinitial/UnityID`*.

All of this has the by product that ***only ITECS-managed webservers will access our web content***. However, it will be necessary at times when other departments will offer services we don't. (e.g. the Information Technology Real Media servers). We can use this multiple-mounting feature to our advantage here. By creating an Website Locker, or having a framework for a new "locker type" we can create those lockers in a standard "user access" path for us – and mount the Locker/volume in a new location in `/afs/eos/engrservers/`*`something`* and provide other servers access to Locker content on a case-by-case basis.

Get rid of the "admin" permission in strategory locations (or How to Avoid the Filesystem-side "Security Hole" with AFS, webservers and IP ACLs)

In addition to allowing one to change the ACLs on a directory in AFS, the AFS "administrate" privilege (the "`a`" in `rliwdka`) allows the holder to create mount points for volumes in the directory where they have the "a" permission.

Where this becomes very problematic is with webserver-accessible areas of AFS.

Consider the following scenario. NC State Student Ingred S. Neaky has a course workspace locker for her work in EOS101. The instructor for EOS101 has given Ingred 500MB for the EOS101 content. Ingred is never going to use all that space – and decides to give several friends access to that space, because they need room to store all the graphics files from www.despair.com and they are running out of quota. All of sudden, the "Course Workspace" becomes a private dumping ground – taking away from other students with legitimate work needs. So we remove "a" from the course workspace lockers, or have auditing scripts run to check the permissions there – this is just limited to those lockers right?

Consider then, Mr. Issac S. Ecure. – who has both a research locker mounted at `/afs/eos/lockers/research/engr/itresearch` and a website locker at `/afs/eos/engrwww/sites/engr/itresearch` (delivered through the URL `www.engr.ncsu.edu/ResearchInIT`).

If Issac had the "`a`" permission in the website locker – he could create a mount point for the private volume for the `/afs/eos/lockers/research/itresearch` locker and mount it in his website space. If he sets the permissions in the research locker private space to `system:anyuser read` or sets the permissions to allow the webservers to `read` that space – then his private research locker is on the web.

No big deal right? We really aren't all that fascist about this. Although we've told Issac boss that the private data is never exported out to the web – and Issac co-worker had placed a spreadsheet with a lot of Social Security numbers in it in the private research locker – because it was *private* – the privacy/security issue there is on the ResearchInIT group. You can't always protect people from themselves.

However, we have a responsibility to "protect" groups from each other. What if Issac mounted some other group's private locker into his webspace? This is more problematic – but it still should not be that big a deal right? That private locker would have to have `system:anyuser read` or would need to have the webservers to have `read` permission in the locker private locker, and we've discouraged that, so the odds are this is not going to be a problem (and if we have auditing – we'll catch it right?).

Well, what if Issac mounts another group's website locker under his website locker – those lockers have webserver read implicitly. All their content will be delivered out via Issac's URL – and even if he doesn't have permissions there, the webservers will. This is still not that big a problem, it is public web content anyway, we don't have indexing turned on, so you can't walk the filesystem from the webserver itself – and if the other website locker has `WRAP` content – the .htaccess files will still be honored if you try to access the content through the standard http:// mechanisms.

Still we'd like to avoid all this if we can, and removing "a" from the website locker solves all these problems. And creating a "webmaster" group, an "editors" group, and a "readers" group – should catch 95% of the access list needs – the webmasters can add UnityID's to the groups, and those groups already have permissions in the filesystem.

Make sure that server-side programs can't get around the permissions restrictions

We have one last problem (famous last words) to solve. The webservers have `read` permissions in all the website lockers. Any process running on the webserver will then have read. This is perhaps *the* reason we remove "`a`" permissions from the Website Lockers.

Any PHP scripts or CGI scripts will then have the ability to read anything the webservers can. Meaning you can bypass `WRAP` entirely. You could deliver any https:// content through http://. The list goes on. Let your imagination run free – especially with scenarios outlined in the section above.

So – what we have to do is contain server-side programs to one's locker. Removing the ability to make mount points helps – but we still have to prevent scripts from walking up into `/afs/eos/engrservers/www` as a whole (by removing `system:anyuser` and not placing the webservers in ACLs in `/afs/eos/lockers` and `/afs/eos/engrww` – we avoid the scripts from reading those spaces at the least).

With PHP, we can limit execution to the "." directory and below. With CGI scripts we can't do this. So what we have to do for CGI scripts is have dedicated hosts, with a different mount path in `/afs/eos/engrservers/cgi`. We haven't created a framework for CGI scripts within the Engineering AFS Locker Model – that will be worked on during the 3rd Quarter of 2002.

Where We Go From Here (or the "Great Migration of 2002" Event)

This is a technical framework for the Lockers, the biggest thing we have to do is get all of the space that we've allocated in /afs/eos migrated. We also are in the process of working on a number of management scripts and tools.

- Migrate all /afs/eos/www content to Website Lockers
- Architect a Locker framework/webserver setup for cgi needs
- Migrate all non-web content to /afs/eos/lockers/{research|admin|personal}
- Create the course workspace management framework
- Create request forms for Lockers
- Work on user documentation and presentations
- Create an auditing framework

We still have a lot of work ahead – but this framework gives us a foundation and structure to work in and – we believe – several years or more of scaleable AFS manageability.

Document Change Log

Date/Version	Notes
2002-05-16 / v1.0	Initial Release
2002-05-16 / v1.01	Clean-up, addition of sections on standard group names, addition of discussion regarding quota assignments
2002-05-22 / v1.02	Correction to how AFS writes files to disk – I was under the incorrect impression that AFS didn't write on top of UFS.