N/A	A. Deason	
Internet-Draft	M. Meffie	
Intended status: Experimental	T. Keiser	
Expires: July 17, 2010	Sine Nomine	
	January 13, 2010	

Methods of Specifying Restrictions on AFS3 ACLs
draft-deason-afs3-acl-restrictions-01

Abstract

The AFS-3 ACL 'a' bit gives users unfettered power to grant, or revoke, privileges, with no provision for enforcing site policy. This memo provides several alternative mechanisms for creating restrictions on what powers the 'a' bit denotes. Three alternative mechanisms for restricting the power of the 'a' bit are proposed: a method for overlaying the ACL with a site-controlled ACL; a method for masking the ACL with a site-controlled privilege mask; and a finely granular meta-acl mechanism for restricting to whom privileges may be delegated, and which privileges may be given to different classes of principals. This memo will serve as a basis for the ACL restriction discussion with the AFS-3 protocol working group. The intended goal of this discussion is to reach consensus on standardization of one or more solutions, and then publish a BCP status memo.

Status of this Memo

---

1. Introduction

Currently sites may give users administrative rights on certain directories in AFS, such as home directories and shared project directories. Users should not, but can, give overly permissive ACLs to those directories. For example, a user could give write and even admin permissions to the system:anyuser group ('fs sa $HOME system:anyuser rlidwka').
This can which can lead to problematic situations, especially for directories that can be served over http. As it stands today, the only possible way for AFS administrators to prevent this (at least in OpenAFS) is to monitor the fileserver's audit log, and correct ACLs that are overly permissive. But this is suboptimal, and is an after-the-fact check.
If you see a viable solution to this problem not listed here, or see any problems with our methods, please let us know. Or if a solution to this problem is valuable to you or your organization, also please let us know.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.) [RFC2119].

---

2. Proposed solutions

Discussions have shown that preventing this is not a simple issue, and that there are a few ways to go about it, each with advantages and disadvantages. Here we will outline 3 general approaches, and show how to use them to meet certain illustrative use cases.
Since the rest of this is quite long, here's a quick summary of the conclusions. We have three methods: 'method A' is the "volume-level ACL overlays" idea, 'method B' is the "volume ACL masks" idea, and 'method C' is the "volume ACL policies" idea. While none of these in themselves cover all corners of all possible use cases, we would probably implement either C by itself, or A and B together to cover a large enough majority of use cases. Of course, unless a serious problem is found, there is no reason to not implement all three.
The bottom line is that I find method C to be the most flexible and the least confusing to end-users, but it is the most confusing to administrators, and it is the slowest (when changing the volume-level permissions). Using methods A+B has the opposite pros/cons.
Here are the details. Each method has an explanation for what it generally is and how it works, followed by its use in a few simple use-case scenarios, followed by the pros/cons.

2.1. Method A: volume-level ACL overlays

With this method, we maintain a single additional ACL in the volume metadata, which is applied to access checks in the volume after performing the per-directory ACL check. It can be thought of as similar to the OpenAFS fileserver's -implicit flag, but more generalized.
For example, if we wanted a volume where system:backups was guaranteed to have 'rl' rights, and system:evilusers was guaranteed to not have _any_ rights, the volume-level ACL overlay would look like this:

```
positive:
 system:backups rl
negative:
 system:evilusers rlidwka
```

Thus, any time an access check is done on an ACL anywhere in the volume: after we do the normal directory ACL check, we look at this volume-level ACL. If the accessing user is in system:backups, they will get rl rights, and if they are in system:evilusers, all of their rights will go away.

2.1.1. How do I prevent system:anyuser/system:authuser write access?

To prevent system:anyuser from having write access, we will need to allow specifying the 'anonymous' user in the volume-level ACL, which refers only to unauthenticated accesses. Then, you just give negative write rights to the anonymous user. The command would look something like:

```
vos setacl -vol user.adeason -acl anonymous idwka -neg
```

For system:authuser, you cannot prevent write access with this method. It is a limitation of this approach. (Giving system:authuser negative idwka rights would revoke those rights from _all_ authenticated users, which is probably not what you want to do.)

2.1.2. How do I ensure nobody in group.foo gets write access?

Just grant negative idwka access to group.foo on the volume. Something like:

```
vos setacl -vol user.adeason -acl group.foo idwka -neg
```

Members of group.foo will now not be able to write anything in the volume.

2.1.3. How do I guarantee group.bar read access?

Same as above, just grant positive read rights. Something like:

```
vos setacl -vol user.adeason -acl group.bar rl
```

2.1.4. Method A advantages

*Changing the volume-level rights is quick.

*Minimal end-administrator confusion; this is relatively simple to understand.

*Simpler than method C to implement.

2.1.5. Method A disadvantages

*We have no way to restrict access of special groups like system:authuser or host IP groups. To get this, we'd have to combine with method with method B or C.

*There is no way to override the volume-level ACL, and have an administrator force e.g. system:anyuser write access on a particular directory.

*Higher end-user confusion. With legacy 'fs listacl', there is no way to see that there is a volume-level ACL in play, and users may have no idea why access is failing for certain cases. Of course, releasing new tools can fix that.

*Performance impact is an extra O(n) ACL calculation for the volume-level ACL overlay in the critical path. But those ACLs should be small anyway.

2.2. Method B: volume ACL masks

With this method, we maintain a mapping of users to a rights mask. Any time an ACL access check is performed, if a positive ACL entry matches a user in that table, the acquires rights are masked to the rights mask in the table.
For example, if we wanted to prevent users from giving away write access to system:anyuser, and prevent users from giving admin access to system:authuser, we could have a table like so:

```
system:anyuser rl
system:authuser rlidwk
```

So any time an ACL entry for system:anyuser appears, everything is treated as if the rights in that ACL entry were logically ANDed with 'rl'. So no user can gain more than 'rl' rights on a directory simply by being in system:anyuser.

2.2.1. How do I prevent system:anyuser/system:authuser write access?

Set the rights mask for them to just 'rl'. Something like:

```
vos setaclmask -vol user.adeason -user system:anyuser -mask rl
```

So any time an ACL entry for system:anyuser appears in the volume, everything will act as if the rights were limited to rl.

2.2.2. How do I ensure nobody in group.foo gets write access?

You cannot _prevent_ access for an arbitrary group with this method, but you can make it harder to do accidentally. You can set the rights mask like so:

```
vos setaclmask -vol user.adeason -user group.foo -mask rl
```

Which restricts any rights for group.foo on any ACL to be restricted to 'rl'. However, a user can intentionally work around this by simply placing group.foo in another group:

```
pts creategroup adeason:foo
pts adduser group.foo adeason:foo
fs setacl $DIR adeason:foo rlidwka
```

Since group.foo itself never apears in the ACL, the ACL mask is bypassed.

2.2.3. How do I guarantee group.bar read access?

You cannot. This method cannot grant additional rights.

2.2.4. Method B advantages

*Changing the rights mask is quick.

*Runtime performance overhead in the critical path is O(1).

2.2.5. Method B disadvantages

*Not as useful for non-'special' groups. That is, it can be trivially worked around, unless you only use this for groups like system:anyuser, system:authuser, or host IP groups.

*No way to override the ACL masks on a particular directory, if the administrator wants to.

*Higher end-user confusion with legacy client tools. There's no way to see what the ACL rights are restricted to until newer client tools are deployed.

2.3. Method C: volume ACL policies

With this method, we maintain policies of who is allowed to set what ACLs in a volume. That is, unlike methods A and B, we perform additional access checks at SetACL time, not at the time when the files are accessed. We would have 4 volume-level ACLs that define what users are allowed to add positive rights ('add-positive'), remove positive rights ('remove-positive'), add negative rights ('add-negative'), and remove negative rights ('remove-negative').
For example, to allow nobody but system:powerusers to grant idwka rights to system:anyuser, we'd have a policy for system:anyuser that would look like this:

```
add-positive:
  system:powerusers rlidwka
  system:anyuser    rl
```

After that policy is set, any time a user not in system:powerusers tries to grant system:anyuser more than rl rights, they will get an EACCES error. This does not change the existing ACLs in the volume; an administrator will need to run an auditing tool to make sure that existing ACLs comply with the volume policy.

2.3.1. How do I prevent system:anyuser/system:authuser write access?

You would call something like this

```
vos setpolicy -add-positive \
  -user system:anyuser      \
  -set-rights rl            \
  -for-user system:anyuser  \
  -in-volume user.adeason
```

to prevent people from giving system:anyuser write access. To ensure that existing ACLs don't permit write access, you would need to run something like

```
vos auditpolicy -vol user.adeason
```

2.3.2. How do I ensure nobody in group.foo gets write access?

To prevent an arbitrary normal group from getting write access, things are slightly different. You would need to prevent users from taking away negative idwka rights, and then assign negative idwka rights to all directories in the volume. So, something like

```
vos setpolicy -remove-negative \
  -user system:anyuser         \
  -set-rights rl               \
  -for-user group.foo          \
  -in-volume user.adeason
```

Would allow users to only remove 'rl' rights from group.foo in negative ACLs. Then you would need to set negative idwka ACLs on all directories in the volume.

2.3.3. How do I guarantee group.bar read access?

Prevent normal users from taking away read access from group.bar, and from granting negative read access for group.bar:

```
vos setpolicy -remove-positive \
  -user system:anyuser         \
  -set-rights idwka            \
  -for-user group.bar          \
  -in-volume user.adeason

vos setpolicy -add-negative \
  -user system:anyuser      \
  -set-rights idwka         \
  -for-user group.bar       \
  -in-volume user.adeason
```

Then, grant read access for group.bar in all directories in the volume.

2.3.4. Method C advantages

* More flexible for a variety of situations. In particular, this allows administrators (or an arbitrary administrator-defined set of users) to override the volume policies, and set e.g. system:anyuser write on a particular directory if they so wish.

* No performance overhead at access-check time. All of the additional access checks are done during SetACL.

* Minimal end-user confusion. ACLs accurately represent exactly what rights different users have. Trying to set an ACL that violates the policy will result in EACCES, so they know the SetACL operation didn't work. It may be confusing as to why it did not, but at least it is clear that the ACL was not changed.

2.3.5. Method C disadvantages

* Volumes need to be 'audited' (or all of the ACLs just need to be set) to make sure they comply with the ACL policy, which can be very slow.

*Higher end-administrator confusion. This by far the most confusing method for administrators to set ACL policies.

---

3. Summary

As I mentioned, we could just do all of these, since they are potentially best suited to different scenarios. Either method C by itself or methods A and B together do seem to cover most of the immediately-apparent use cases, though. To summarize the general areas in which the different methods are better or worse:

	Better	Worse
flexibility:	method C	method A+B
end-user confusion	method C	method A+B
end-admin confusion:	method A+B	method C
policy-change speed:	method A+B	method C

There are other pros and cons, but I think those areas are the only ones where it matters much. If you see any problems that aren't listed here, or if you particularly want one of the described methods, please let us know.

---

4. Acknowledgements

The authors would like to thank Jim Rowan for discussing problematic interactions between the proposed ACL policy management techniques and PTS user-managed groups, and Jeffrey Altman for helping to better frame the problem statement and proposing alternative implementations.

---

5. IANA Considerations

This memo includes no request to IANA.

---

6. Security Considerations

The existing security model is known to be flawed. This draft attempts to improve the situation by limiting the extent to which end users can modify file system permissions. However, it is known that this is not sufficient to address all possible ACL attack vectors. Two key areas of concern are authorization for modification of policy metadata, and interaction with user-managed PTS groups.
How modification of policy data will be authorized in an environment using RBAC is not clear; it is known that system:administrators is not always the appropriate group of principals. In highly secured environments there may be a desire to restrict modification of policy to a security-related group, rather than the group responsible for maintaining the AFS server plant. This is not addressed in the memo, although it could be addressed by means of additional per-volume metadata.
There are proposed attack vectors by which a user-managed group can be used to get around ACL restrictions. While these attacks can bypass a naive ACL policy specification, it is possible to circumvent these techniques through the use of negative access control policy entries.

---

7. Normative References

[RFC2119]	Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

Authors' Addresses

	Andrew Deason
	Sine Nomine Associates
	43596 Blacksmith Square
	Ashburn, Virginia 20147-4606
	USA
Phone:	+1 703 723 6673
Email:	adeason@sinenomine.net
	Michael Meffie
	Sine Nomine Associates
	43596 Blacksmith Square
	Ashburn, Virginia 20147-4606
	USA
Phone:	+1 703 723 6673
Email:	mmeffie@sinenomine.net

	Thomas Keiser
	Sine Nomine Associates
	43596 Blacksmith Square
	Ashburn, Virginia 20147-4606
	USA
Phone:	+1 703 723 6673
Email:	tkeiser@sinenomine.net