OSD Object Fencing Changes to r09

David Nagle, Panasas, Inc.

Enabling Revocation of Object Access in OSD R9

Overview

There has been a great deal of discussion on the reflector about object fencing, revocation of object access and object version number (called the security version tag in V9 of the OSD spec). In this document I'd like to discussion the need for revoking access to an object, the requirements, and compare how to implement this using the object version number and alternatives.

Revocation of Object Access

There are several reasons why the OSD interface must support quick and efficient revocation of object access. First, file systems need a mechanism for enforcing coherence. While this can be done with an external lock manager, the performance cost of external lock managers can be very high[1]. Second, OSDs need a mechanism for denying access to damaged objects. Since the OSD will often be the entity that discovers a failure, it is important that the OSD implement a mechanism by which the OSD can deny access (i.e., fence) to an object without external manager intervention. Finally, RAID above the object interface needs efficient object access revocation in able to support efficient reconstruction of objects (otherwise, RAID across OSDs will also require an external lock manager).

After many discussions at CMU and among the company members of the NSIC NASD working group, it was decided to associate an "Object Version Number" (OVN) with every object in the OSD.   Encoding the Object Version Number in the Capability ensured that the OVN was checked on every access. However, the OVN was NOT a security mechanism. It was designed to enable efficient management across a distributed system of OSDs and clients, employing a field in the security capability to guarantee:

1) Efficient execution of OVN checking at the OSD
2) Rogue clients cannot bypass the consistency requirements of the file system(s), the data correctness of the RAID algorithms, nor access corrupt objects stored on the OSD.

To meet the needs of the file system, RAID across the OSDs, and OSD internal consistency, the OVN was to be implemented and used as follows.

1. The Object Version Number is encoded in every capability and checked on every access to the OSD.

[1] To help enable efficient file system coherence, Seagate and the University of Wisconsin implemented locks inside the device (i.e., SCSI drive), but this solution suffered from failure drive failure/recovery problems because the locks were stored in volatile memory.

2. If the OVN encoded in the commands CDB matches the OVN stored on the associated OSD object, the command is allowed to proceed; if the OVN did not match, the command was immediately terminated
3. Outside entities (e.g., manager, client) with the appropriate permissions (i.e., change-version-number-allowed bit[2] set in a capability), or the OSD itself, can modify the OVN.
    a. File system managers change the OVN to immediately prevent client access to an object. Typically this would be done when:
        i. File permissions changed
        ii. File system coherence policies require that only specific client(s) would be granted access to an file/object[3].
    b. RAID controller changes the OVN to prevent access while reconstructing parity of data.
    c. OSD changes OVN to prevent access to a damaged object (either damaged data or attributes)

The benefit of this solution is that a single mechanism, the Object-Version number, is able to efficiently and security support a wide set of requirements. Further, because all revocation is done through a single mechanism, both the OSD implementation and standard are greatly simplified. Finally, the OVN allows for rapid revocation, minimizing the performance cost for file systems, RAID controllers, and clients[4].

Binding of Security and Revocation in the Current R9 Spec

The security model is written to assume that every command includes a cryptographically secured capability. The one exception is specified on page 26/Table 6, when the no security option (NOSEC) does not provide a capability.

In the OSD security model, the capability enumerates all operations a client may perform against the object specified in the capability. The signing of the capability is the SECURITY feature that ensures that 1) the capability has not been tampered with and 2) the capability is issued by a client that also holds the capability key.

The capability itself is used to convey file-system specific policy information between the (security) manager, the client and the OSD. For example, file systems typically support read-only files. In the OSD model, the capability is the place where the file system policy manager (usually embedded in the security manager) would inform the client that a file was read-only. By encoding this information inside a

[2] I didn't see this in the current spec, so it may need to be added

[3] As mentioned before, it would be possible for an external lock manager to provide the same functionality at the additional cost of software complexity and messaging. By using a mechanism at the OSD, file systems can immediately revoke access by changing the object version number, avoiding the cumbersome and expensive task of contacting every possible client that might hold access to an object.

[4] It is important to note that revocation should not typically be a common operation, especially in the context of RAID reconstruction or object corruption (at least we hope these events are rare). Therefore, it is not essential that a system support fast/efficient processing once an object version number has changed. It is fine to require extra trips to a RAID controller or manager because these messages should be infrequent. However, it is essential that we provide a fast and efficient mechanism for revoking access. Managers and RAID controllers should not be burdened with coherence algorithms that do not scale as the number of OSDs and/or clients scale. And, since revocation checks must be performed on every access, efficient OSD mechanisms that enable low-cost OSD implementations are essential.

capability, the system provides a degree of protection against client errors (e.g., a client issues a write command against a read-only object).

More importantly, commands that do not include a capability completely circumvent the object-access revocation mechanism designed into the OSD model with the use of an object-version number (called the SECURITY VERSION TAG in v9 of the OSD spec). Revocation of access to an object is not a security issue, it is a coherence issue that distributed systems require to ensure correct functionality (see XXX for a full description of the history and use of access revocation in the OSD model).

Therefore, it is important to separate the notion of security from the OSD capability. Specifically, we recommend that capabilities be attached to all commands, regardless of the security level. The major change is that NOSEC level would still require a capability to be sent on every command. However, the capability would not be signed and no signature checking would be performed at the OSD. Of course, this leaves open the possibility that rogue clients could tamper with a capability. However, in the NOSEC model, clients are trusted not to tamper with the capability. However, it is essential that the manager, client and OSD exchange capabilities so that managers can enforce file system policies, including read/write access and coherence.

Proposed Solution

1. A statement about object-fencing, describing that an OSD must increment the object version number when the OSD detects object damage.
2. A Capability-based permission bit that allows commands to modify the object version number
3. Security level NONE should provide a capability, but will not be signed.
    a. Remember that a capability is permission. Security only enters into the picture when the capability is signed.

I have annotated the R9.0 OSD spec (see below) with the changes we believe are necessary to provide for revocation.