Proceedings of the 11th USENIX Security Symposium

San Francisco, California, USA
August 5-9, 2002

Access and Integrity Control in a Public-Access, High-Assurance Configuration Management System

Jonathan S. Shapiro John Vanderburgh
shap@cs.jhu.edu vandy@srl.cs.jhu.edu
Systems Research Laboratory
Johns Hopkins University

Abstract

OpenCM is a new configuration management system created to support high-assurance development in open-source projects. Because OpenCM is designed as an open source tool, robust replication support is essential, and security requirements are somewhat unusual – *preservation* of access is as important as prevention. Also, integrity preservation is a primary focus of the information architecture. Because some of our supported development activities target high-assurance systems, traceability and recovery from compromise are also vital concerns.

This paper describes the mechanisms used by OpenCM to meet these needs. While some of the techniques used are particular to archival stores, others have potentially broader applications in replication-based distributed systems.

1 Introduction

OpenCM – the Open Configuration Management System – is a new configuration management (CM) system created to support high assurance, open source software development. It uses cryptographic naming and authentication to achieve distributed, disconnected, access-controlled configuration management across multiple administrative domains.

High-assurance CM systems must support requirements for audit, traceability, and process enforcement [ISO98]. In particular, higher evaluation assurance levels require that every modification to the trusted computing base be validated (a form of audit) by a second person. Implicit in this requirement is the need for provenance tracking: knowing who performed which check-ins. If the same repository is to be used successfully for trusted and untrusted code bases, access controls must be present on branches. For validation to be cost-effective, correct change sets should be straightforward to generate. This requires that the CM information architecture provide strong integrity guarantees to guard against falsification of prior configurations.

Open source projects introduce a different, and to some degree competing, set of requirements. These projects commonly span traditional administrative and corporate boundaries. Open source developers make heavy use of disconnected development at home or while traveling. These practices create multiple vulnerabilities:

- Code in the developer workspace may be tampered with by a malicious party. With current CM systems, there is no means to audit and recover if such code is committed.
- The inability to commit while disconnected encourages larger change sets that commingle multiple changes. This facilitates developer mistakes.
- Existing CM systems require developers to have a login account on the server to revise objects in the repository. Allowing such broad access from untrusted (and potentially compromised) clients invites compromise of the server as well.

Further, the multi-organizational nature of open source teams means that a supporting CM system must not rely on a single source of administrative authority for account generation or access control; the authorization and protection model must provide for controlled commingling of administrative domains.

The EROS [SSF99] project is developing a high-assurance operating system using an open source development process. To support this project, a new CM system supporting the combined requirements of open source and high assurance was required. Examination of existing configuration management systems suggested that none could meet our requirements, as none provides careful provenance tracking or supports integrity checks across hostile replicates. OpenCM [Sha02] was created to resolve this deficiency.

This paper describes the first-generation access and integrity control mechanisms of OpenCM, which provide a safely replicatable store while avoiding the need for distributed trust. We describe the usage model, threat model, guarantees provided, and discuss some implications of the OpenCM access control mechanism. We also identify two vulnerabilities that have emerged from oversights in the intial design, and the changes that are being made in OpenCM to overcome these vulnerabilities. While the focus of this paper is OpenCM, we believe that the underlying information architecture is a general-purpose schema that provides a wide-area, integrity-checked distribution and naming system for online archival content.

2 OpenCM Usage Model

OpenCM is a client/server application. Developers typically work on individual workstations with the repository hosted on a centrally managed server. In small projects these may be the same machine. Typical use is similar to that of CVS [Ber90]: the developer checks out a baseline version of some branch, makes modifications, and commits them back as the new state of that branch. As with CVS, the model is “change, then integrate” rather than “lock, then change.” Experience with both models suggests that post-integration is more effective for small development groups. Reasonable users might disagree with this view, and lease-based locks are being contemplated for a future version of OpenCM.

2.1 Differences from CVS

Key differences between OpenCM and CVS are as follows:

- OpenCM captures a complete audit trail of all modifications, provides fine-grain access controls of reads and writes, and preserves content integrity when replicating across hostile repository servers.
- OpenCM manages configurations, not collections. Every “commit” is a unique, atomic action. A cleanly reconstructable trail of versions is therefore preserved – even across renames.
- OpenCM supports disconnected commit. A developer can check into a local repository when the reference repository is unreachable, allowing development to proceed and change history to be tracked when developing in remote locations. Subsequent integration preserves the history trail as well as the changes.
- OpenCM is designed for use as a software distribution infrastructure. Servers can selectively replicate some or all of various branches for redistribution or local use.
- OpenCM uses SSL/TLS client authentication for authorization. It is therefore independent of the underlying operating system, supporting multi-organizational development without requiring the repository server to support “foreign” users.
- OpenCM leverages its integrity checking mechanisms to reduce the number of network transactions required when performing developer operations such as update and revert.

From an assurance perspective, three differences between OpenCM and CVS are especially important:

First, OpenCM operation is not based on “patches.” Patches (as generated by diff) describe a change in the content of a line of development without conveying the history or provenance of the changes. This is insufficient to support audit and traceability. OpenCM instead uses an object-based change description that preserves the entire connected graph of a development process, allowing the history of all integrated changes to be reviewed if OpenCM is properly used. In the process OpenCM preserves a complete audit of who performed each change.

Second, OpenCM provides access controls on both branches and “files.” The second is a misnomer, which will be explained in Section 5.4. A single project supported by a one or more OpenCM repositories can have distinct development branches and audited branches, and can provide some degree of support for “social” constraints (e.g. technical writers typically should not modify C source code). While separate branches can be used to keep selected users out of trouble, this can be inconvenient in a tight-knit project team.

Third, OpenCM provides mechanisms for end-to-end integrity checks between the originating repository for a branch and the end client. While it is possible for malicious replicates to inject bad data, such injections can be reliably detected by the client given only a modest amount of externally transmitted, non-sensitive information (a signature verification key).

2.2 Threat Model

Given that end users typically develop on untrusted machines, OpenCM does not attempt to prevent the introduction of bad code. The design goal is to ensure that all development changes are performed by authenticated users, and that an audit trail is preserved for all changes. In the event that a client system is compromised, the design goal is to (a) quickly disable that user’s authority to modify, and (b) retain enough information to successfully audit the changes made under the now-compromised authority.

A direct consequence of untrusted clients is that OpenCM is vulnerable to denial of resource attacks. A compromised client may be used to upload an unbounded amount of state to a repository. This is unavoidable if untrusted clients are to be supported. Two mechanisms can be used to mitigate this:

1. As a result of the object naming strategy, OpenCM repositories store duplicate content only once.
2. Quotas can be imposed on new state introduced per-transaction and on total transaction duration. We have not (yet) implemented this.

Each of the preceding vulnerabilities results from a functional requirement. In both cases, the “recovery” mechanism is the same: disable the compromised user, perform an audit of the suspected changes, and garbage collect the damage.

With the inherent conflict between availability and resource attacks acknowledged, the remaining threats against OpenCM are relatively few:

1. OpenCM relies on the Secure Sockets Layer (SSL/TLS) [DA99] for transport security and client authentication. Any vulnerability in SSL is a potential vulnerability in OpenCM.
2. More realistically, an attacker might attack the pass-phrases of the user keys. This is a widely recognized and ongoing weakness. [MT79, FK89, Wu99]
3. An attacker may seek to compromise the OpenCM repository from underneath by compromising the operating system or the server daemon.
4. An attacker may seek to impersonate a repository, attempting to pass off bad (and perhaps compromised) content as valid. In high assurance applications, impersonating a repository that serves trusted content is a particularly urgent concern.

SSL is critical infrastructure to a very large number of applications. This tends to make it widely attacked, widely tested, and quickly repaired. We are not cryptography experts, and prefer to let the experts address these issues.

Our primary concern with SSL is the second vulnerability: weak passphrases. Here we have chosen to compromise. While stronger authentication (e.g. S/Key [Hal94] or OPIE [MAM95]) is certainly possible, we suspect that it is not helpful in practice. If an attacker has compromised the end system, which would be necessary to steal the private key, we must assume that they have left a Trojan horse as well. In that case stronger encryption merely promotes false confidence.

A second potential concern with SSL is that the operating system cannot assist in access enforcement. Given the decentralized nature of the OpenCM authorization model, it is not possible for the operating system to do so.

OpenCM provides scalability by enabling replication across untrusted repositories. At some cost in propagation delay, this allows load distribution across replicates. Experience from other projects suggests that this is a heavily used form of software distribution [Pol96]. For high-assurance software, this distribution method introduces potential vulnerabilities, and the integrity of the distributed content must be protected. OpenCM uses a combination of cryptographic techniques to achieve this (Section 3.5).

2.3 Guarantees

Provided that a signature verification key can be distributed via a trusted path, OpenCM provides the following guarantees:

(1) The user can verify that any object obtained from a repository is valid. By "valid," we mean that an integrity check can be performed that reveals whether this object is complete, and that it was checked in by an authorized modifier of the branch. Valid does not imply correct – verifying the code is beyond the scope of OpenCM.

(2) While all objects received can be authenticated, no guarantees are provided about whether the object is up to date unless the user obtains it from the originating repository. If the object is obtained from a replicate repository, it is guaranteed to have come from earlier valid state of the branch.

(3) If a user's authentication key or client is compromised, total integrity exposure is limited to the set of branches that the user can modify; OpenCM as a whole is not compromised.

(4) Integrity verification is designed to be possible even if the user obtains certain types of *partial* copies of a branch. For example, the user may choose to replicate only selected versions of a branch, and can validate that the versions obtained are authentic.

(5) Provided the originating repository is not compromised, the complete history of each branch originating at that repository will be available from that repository. This has implications for merge management.

(6) The repository records authentication information for every change. In the event of user key compromise, this information is sufficient to allow audit of suspicious changes.

(7) Impersonating a repository requires both stealing the repository's private key and compromising the IP routing mechanisms near the client.

3 Information Model

To provide these trust guarantees, OpenCM takes advantage of the archival character of configuration management data. Archival information has two unusual properties that tremendously simplify integrity checking. First, most objects in an archival store are persistent and unchanging; we refer to these objects as *frozen*. Second, objects that *can* be modified allow modifications only under certain constraints.

Depending on the application, two management strategies for modifiable objects are possible:

- Eventual consistency, in which modifications are performed locally and eventually make their way by replication to some (possibly federated) master repository.
- Source-controlled objects, where changes for a given object are permitted only on an object-specific "owning" repository. A sequence number can be used to resolve replication disputes for such objects.

Configuration management applications fall under the second category, because a total ordering on the sequence of changes made to a given branch is required, and this cannot be guaranteed by eventual consistency.

3.1 The Repository Schema

The basic OpenCM repository is built on a relatively generic schema consisting of five object types: mutables, revision records, users, groups, and frozen content (Figure 1). Every mutable carries its own name, the names of its controlling read and write group(s), (which are in turn mutables), the number of revisions that have been performed on this mutable, a human-readable name and description, and a sequence number indicating how many times the mutable has been in some way altered (used in replication). Mutables also carry a "flags" field. At present, the valid flags are "frozen," indicating that the mutable cannot be revised, and "notrail", indicating that historical revision records for this mutable need not be preserved. A mutable can be legally modified only by its originating repository, and is signed using that repository's signing key after each revision.

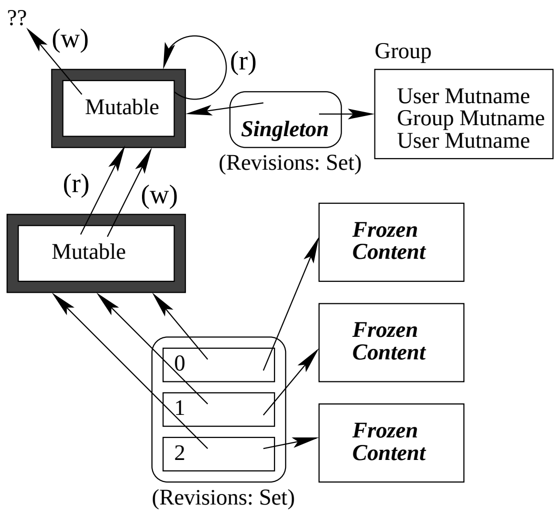

Figure 1: Repository Schema.

Every mutable has associated with it zero or more revision records. Each revision record contains a sequence number, the name of its associated mutable, a date stamp, a pointer (a cryptographic hash) to the frozen content associated with that revision, and a cryptographic signature performed using the originating repository's signing key.

The repository layer knows only two types of (frozen) content objects. *Users* hold public keys and home directory mutable names. *Groups* hold a set of user or group mutable names.

Content objects in the OpenCM repository cannot be modified, and are therefore referred to as "frozen." Because these objects are frozen, their semantics depends exclusively on their content, and there is no reason to keep multiple copies of objects whose content chances to be identical. Frozen objects are therefore named by their cryptographic hash.

Using cryptographic hashes achieves compression and integrity checking at the cost of imposing a restriction on the application-level schema: the content model must be acyclic. Cycles in object names based on cryptographic hashes cannot be resolved without combining the objects into a single bundle. In the OpenCM repository, cycles can be managed by having a frozen object that contains the name of a mutable object. The OpenCM application does not require this.

3.2 OpenCM Content Schema

The content schema of the OpenCM application is shown in simplified form in Figure 2. Branches are mutable. Each branch consists of a linked list of configuration objects that in turn hold Entities.

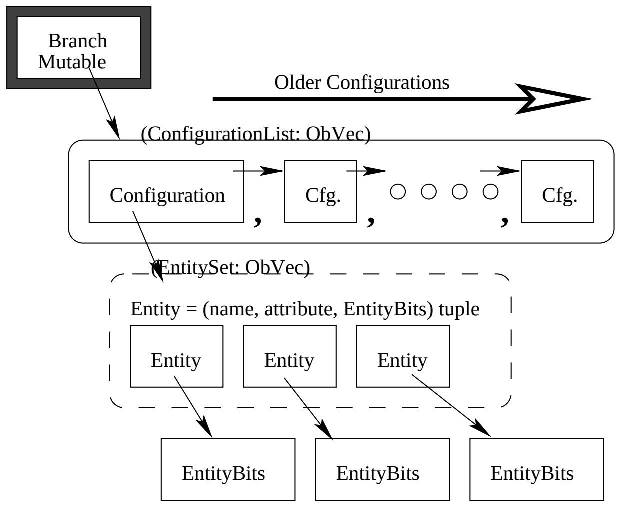

Figure 2: OpenCM Information Architecture.

A Configuration is simply a set of Entity objects. Each Entity provides a binding between a name, a set of attributes (client-side workspace permissions, for example), and an EntityBits object name. The EntityBits object describes the content, as opposed to the metadata, of an object. The separation of Entity and EntityBits is purely a convenience. It allows the repository to record permissions and rename operations without needing to re-record the associated object content.

The Entity/EntityBits combination represents a *single version* of a given object. In OpenCM, all versioning is performed on configurations. Committing a change to a single object is accomplished by creating a new EntityBits object, a new Entity object, and a new Configuration object. The new Configuration is identical to the old one with the exception that the EntityBits name for the previous version of the modified object is replaced by the EntityBits name for the new version. While there is no implied or required ordering of the Entity objects within a Configuration, unordered collections are serialized in such a way that their object names are sorted. This maximizes the likelihood that the repository will be able to identify common content between two objects that can be leveraged for storage compression.

While the described schema is clearly specific to the CM application, the essential enabling properties for integrity validation are relatively generic:

- The content model is acyclic. More precisely, cycles can be present only by having a frozen object contain the name of a mutable object.
- Each mutable object is signed whenever it is changed.

Any information pool that can be reduced to these constraints can use the techniques described in this paper to provide distributed integrity checks across untrusted replicating stores.

3.3 Frozen Object Naming

The most important integrity mechanism of OpenCM is built into its object naming strategy. Frozen objects are named in the repository by the cryptographic hash of their content (currently SHA-1) Thus, a Configuration object is named by its hash expressed as a string of the form

```
frz.sha1.01cb4c...7245
```

where "frz" is a non-normative prefix indicating the type of the named object (used primarily for repository debugging), and "01cb4c...7245" is the SHA-1 cryptographic hash of the frozen object.

Using a cryptographic hash in this way has several desirable attributes.

First, cryptographic hashes simultaneously provide a unique naming scheme for all frozen objects and allows the content delivered by any repository to be checked for integrity failure. No practical technique is currently known by which to generate a string whose cryptographic hash collides with a previously known cryptographic hash. Further, the likelihood that such a string, if generated, would pass higher level content checks (such as syntax checking during compilation) is vanishingly small.

Second, cryptographic hashes are *universally* unique. Partitioned (e.g. disconnected) repositories can generate names for frozen objects without fear of collision. This is helpful, as it prepares the ground for later replication. The only case in which frozen object names should collide is the case in which an object has already been copied from one repository to another. In that case, the content should be identical, so no conflict resolution mechanism is required.

Finally, the use of a universally unique naming scheme allows efficient replication. Before fetching a frozen object from a source repository, the replication engine can check with the destination repository to see if the object is already present.

OpenCM currently uses SHA-1 hashes, and we have performed extensive testing of real repositories without collision. However, the hashing strategy name is encoded within the hash as recorded. In the unlikely event that a collision ever occurs, an alternative hashing strategy can be employed to generate a fallback name. Given the distributed and semi-connected nature of OpenCM, however, such a collision cannot necessarily be detected.

3.4 Mutable Object Naming

Regrettably, mutable objects cannot be named by cryptographic hashes of their content, because changes to the object would lead to object name changes, breaking links to these objects. To name mutable objects, OpenCM relies on cryptographically strong random number generation. Mutable object names are strings of the form:

```
opencm://7a5d...93/27da...05
```

where "7a5d...93" is the originating repository's name and "27da...05" is a cryptographically generated unique name for this mutable assigned by the originating repository. A repository's name is generated by taking the SHA-1 hash of its initial public key (see Section 4.1). This eliminates the risk of inadvertantly disclosing the signing key [Dav01].

The choice of a URI format for mutable names is not accidental. We plan to maintain a repository registry under the `opencm.org` domain. If the "7a5d...93" repository has been registered, then

```
7a5d...93.registry.opencm.org
```

will resolve to the IP number of the serving host.

Good random number generators are not universally available, and where available are not always properly installed. At present, OpenCM relies on the OpenSSL implementation as its source of random numbers. Unfortunately, current versions of OpenSSL rely on the underlying native random number generator. The `/dev/random` and `/dev/urandom` generators are reasonably good, but generators on other platforms are quite variable. The resulting exposure is less than it might at first appear, because mutable object names are generated on the originating repository, and can therefore be tested to prevent collision. The inclusion of the repository's name in the mutable object name therefore reduces the problem of name collision to elimination of collisions among repository public keys.

3.5 Revision and Mutable Signing

To ensure that mutable object integrity can be verified, a digital signature is computed each time the mutable object is changed. The signed content includes the object's name as well as its content, and the name includes the repository's public key. This makes object substitution detectable. In the usual case, the mutable object retains its change history. A mutable object consists of:

```
sequence-number
mutableURI
r-group
w-group
nRevisions
signature-of-preceding
```

The associated revision objects consist of:

```
revision number
mutableURI
contentName
authorURI
date
signature-of-preceding
```

Provided that the repository's signature checking key can be reliably determined, the digital signature provides both authentication and integrity checking of the mutable and revision objects. Frozen objects are named by the cryptographic hash of their content, which provides an inherent integrity check. Since the `contentName` references a frozen object, the authentication of the digital signature effectively includes the entire graph of frozen objects reachable from the `contentName` object.

4 Authentication

OpenCM authentication is built on SSL client authentication. Every user has (at least) one X.509 key, and wields this key in response to the SSL client authentication challenge. We are in the process of implementing an `OpenCM-agent` utility similar to the `ssh-agent` [Ylo96] to serve as the user's proxy for key management.

4.1 Server Authentication

While OpenCM is built on SSL/TLS, we have chosen to avoid reliance on certificate authorities for key authentication. The human association provided by user certificates is not required by this application, and existing certificate authority mechanisms do not provide a reliable means to preserve repository identity across key updates. OpenCM therefore uses self-signed certificates.

At present, OpenCM implements repository authentication in a fashion similar to SSH [Ylo96]. The client makes its first connection without knowing the repository's public key, and records the public key provided by the repository to detect later substitutions. Security-conscious users can preload the client-side public key cache by explicitly inserting the correct repository key prior to connection. While adequate as a first implementation, this solution is unsatisfactory for secure operation by everyday users.

The next release of OpenCM will use a certificate *registry* mechanism: each repository will have both an online repository key and an offline registry update key. The update key is used exclusively to sign registry updates. A repository registry service publishes a set of (repository name, IP address, current public key, previous public key, registry public key) tuples for each repository. The SHA-1 hash of each update is signed by the registry update key, providing a checkable sequence of updates. The initial public key can be checked by comparing its SHA-1 hash to the server's name.

By registering a public repository with a modest number of independent registries, server public keys can be adequately published and the risk of hostile registries can be mitigated. In order to forge a server that publishes trusted content, an attacker must obtain the private key, control a colluding key registry, and be able to redirect registry connections from the client to this registry. The last requires compromising either the client or the IP routing infrastructure near the client.

Certificate registries require neither a hierarchy nor a carefully managed certificate authority key. They are therefore robust against both political interference and registry compromise. The cost of this is that the per-repository registry update key is analogous to a title instrument, and *must* be guarded.

4.2 User Authentication

OpenCM distinguishes two levels of access control: access to the *repository* vs. access rights on *objects*. Access to the repository is effectively an authentication control. The repository access permitted to any given user is stored independently by each repository, and can be updated only by members of the repository administrative group. Read access allows the corresponding user to read the repository, subject to the further constraints of the access control lists on any objects the user attempts to access. Write access conveys the authority to upload objects to the repository (i.e. to consume storage resources). This access is honored only on the repository of origin. Mutable objects are subject to the further constraint of per-object access controls.

Because repository access is controlled on a per-repository basis, User objects can be replicated for the sake of traceability and display without granting authority on the destination repository. In public replicate repositories, it is usual to grant replicated users read access on the replicate repository.

OpenCM also provides a "dog house" for keys that are believed compromised. If a user's authentication key has expired, or if it appears in the dog house, it will not be authenticated. Compromised and expired keys are retained for purposes of checking historical signatures.

The use of cryptographic authentication renders OpenCM administratively "agnostic." An outside user (e.g. one from another company) can be "introduced" to a repository simply by adding their user key to the valid readers list. If they are an active (modifying) collaborator, they can also be added to the valid writers list. While these are preconditions to accessing the OpenCM repository at all, neither of these actions grants the user the ability to fetch or modify anything on the repository. Introduction merely makes the key available so that individual project administrators can choose to add this user to their respective project groups. Note, however, that the resulting authority is entirely limited to OpenCM. The outside user has no ability to log in or to run programs outside of the control of OpenCM. OpenCM authentication is "user to service" rather than "user to server."

5 Access Control

All object references in the OpenCM repository originate with mutable objects. Frozen objects are, in effect, the *content* of the mutable objects that reference them. Therefore, the access control mechanisms appropriate to each are different.

5.1 Access to Mutables

The OpenCM access control mechanism for mutable objects is similar to conventional ACLs with a twist: access control lists are first-class, mutable objects, and are themselves subject to access control lists.

Every mutable object names a "reader" and a "writer." These slots may legally contain either the mutable URI of a user key or the mutable URI of a *group*. Group membership is transitive: a user is a member of a group G_0 if (a) they are directly listed as a current member of G_0 or (b) they are a member (recursively) of some group G_1, and G_1 is in turn a directly listed member of G_0. Due to replication, it is possible for locally undetectable loops to arise in the group containment relationships. The membership expansion algorithm is careful to detect and deal with cycles.

Groups are themselves mutable objects. Like all mutables, groups are initially created as readable and writable by their creating user. The creating user is also inserted as a member of this group. User U can create a group G, make G the reader or writer of some branch, and then add other users to G, granting them read (write) authority while retaining the ability to revoke that authority. It is common in this situation to make the group's `r-group` slot name the group itself (i.e. make it self-readable) so that users can see which groups contain them.

The purpose of transitive groups is to facilitate delegation. By adding a group H as a member of G, where H is readable and writable by some other user, user U can revocably delegate access control to this other user. This is particularly important in cross-organization collaborations, where each participating company or entity may need to make its own local decisions about access control.

It should be noted that delegation of this type is impossible to prevent. Any user with read access to any object and write access to the repository has sufficient authority to create a new line of development derived from any existing state – this is required to allow branch creation. The new branch, however, is owned by its creating user, which leaves that user free to alter the access rights of the branch.

Given this, the question to ask is not "How shall we prevent authorized users from behaving badly?" but rather "How shall we ensure that when such things are done reintegration remains possible?" By giving the user an opportunity *not* to break the revision trail, OpenCM preserves the option of later re-integration.

5.2 Access to Frozen Objects

The readability test for frozen objects is *reachability*. If an authenticated user has read permission on a mutable object, any frozen object reachable from that mutable object is likewise readable. There are no ACLs on frozen objects.

This point is a frequent source of confusion about the architecture, and it may be better understood given a brief digression on the implementation of access control lists. Imagine an unchanging (frozen) content object for which we wish to maintain a revisable access control list. To achieve this, there must be some place where a mutation can occur. Either the access control list itself must be mutable or there must be some third, mutable container object that records the association between the content object and its access control list. The two designs are functionally interchangeable. In either case, the content object has in effect been rendered mutable. Extending the content model to be a graph rather than a single blob of bytes does not change the basic requirements for access control, nor does it inherently change the security of the access control model (but see Section 7).

5.3 Impact of Replication

Replication and first-class groups interact in a potentially surprising way. If a group G_1 in repository R_1 contains as one of its members another group G_2 in repository R_2, replication will have the side effect of copying the reader keys reachable from G_2 onto R_1. This in turn has the effect of allowing those users to read objects on R_1 subject to the constraints of their respective access control lists. In effect, control of local objects can be delegated to groups that originate on a remote repository. These groups may in turn be controlled by remote users. This is either a bug or a feature, depending on point of view.

We do not yet have enough experience with OpenCM to understand what the real impact of this will be. If it proves to be a source of difficulty in practice, fully local control can be restored by requiring that if G_1 is added as a member of G_0, the addition will succeed only if both objects have the same originating repository. If necessary, we will add a repository configuration option to enforce this constraint.

We expect, however, that such a configuration option would not often be used because it would interfere with disconnected development. When performing a disconnected commit to a locally created temporary branch, it is typically desirable to create this temporary branch using the same read and write groups as the original branch in order to allow others to see the development history when the temporary branch is replicated back to the master repository for integration.

5.4 Finer Access Controls

Experience in our research lab suggests that finer access controls are extremely useful. For example, we have students working on drivers for the EROS project. It is useful for them to be able to modify these drivers without being able to modify the kernel code. At present, we handle this by creating a distinct line of development (branch) for each student's work, but this ultimately impedes integration. The concern is error rather than malice: deleting the wrong file could cause a fair bit of disruption. Fine-grain access controls help reduce such errors.

Curiously, this type of access control is not really access control on files at all. Files in OpenCM are immutable, so there is no need to prevent their modification. Rather, these controls restrict the binding of file *names* in the client-side workspace. When we say "Fred can only modify .html files," we are really saying that each configuration defines a set of (client-name, object-name) pairs, and we are going to restrict Fred's selection of legal client-names to those that end in `html`.

OpenCM provides fine-grain access control in the form of a table of regular expressions. This table describes which subsets of the client namespace a given user or group can modify.

5.5 Summary of Access Checks

Reading an OpenCM object requires that:

1. User key is not in the dog house.
2. User key has read access to repository.
3. User key appears (transitively) in the read or write group of the mutable object they are trying to access.

Creating a new mutable object requires that:

1. User key is not in the dog house.
2. User key has write access to repository.

Committing a new revision *additionally* requires that:

1. User key appears (transitively) in the write group of the mutable object they are trying to revise.
2. For all client-side names in the configuration whose binding has changed relative to the previous version, the user is permitted to make binding changes for that name according to the fine-grain control table.

6 High-Assurance Development

The EROS project is attempting to construct a system that can evaluate successfully at the highest currently defined evaluation level (EAL7). OpenCM is designed to facilitate relatively open access, while providing accountability for modifications. In this section, we describe how OpenCM has been deployed within the Systems Research Laboratory to meet the EAL7 CM requirements.

The essential vulnerabilities in the system lie in (a) the possibility that the server host has been compromised, and (b) the possibility that the user's key has been compromised. The first presents a chicken and egg problem: until something like EROS exists in widely-available form, it is impossible to adequately protect the EROS code base. For now, we have settled for locking down the machine: OpenCM is the *only* application connected to the outside world on our high-assurance repository host, and periodic off-line backups are made of the repository.

The key to high-assurance development is to ensure that commits on the high-assurance branch are made using offline keys from a known-trusted machine. When performing these commits, we first inspect (as a group) the proposed changes, making note of the signature of the version under inspection. We then physically log in to a dedicated account on the CM server, perform an integrity check on the version to be merged, and perform the merge using the authority of a key stored on a floppy disk.

7 Vulnerabilities

There is little that can be done to protect a user if they can be convinced to ask initially for a non-authentic branch. In properly constructed cryptography, the best that can be achieved is to ensure that users get what they ask for.

Beyond this, the initial implementation of OpenCM suffers from two significant vulnerabilities embedded in the information architecture as originally designed. We describe them and possible solutions to them here.

7.1 History Backwalk

The first exposure concerns access controls on frozen objects. As discussed in Section 5.2, the access predicate for a frozen object is based on reachability. We made an initial, naive assumption that cryptographic hashes were unguessable, and that this provided sufficient protection to prevent unauthorized reads. The `GetFrozenObject()` repository operation therefore did not perform access checks. Our theory was that even if such a name leaked, only a single version of a single branch is exposed, and that repository-level authentication was a sufficient impediment to theft. In hindsight, this was mistaken.

In the OpenCM schema, every Configuration object includes the frozen object name of its predecessor configuration (the “Older Configurations” arrow in Figure 2). This “back pointer” is necessary to ensure that the merge algorithm works; its presence (or equivalent) and accessability is a functional requirement of the configuration management system. An unforeseen consequence is that any holder of a valid Configuration object name who can authenticate to any replicate repository can obtain the entire history of development up to that Configuration. For open source development, this is a non-issue, but for proprietary projects it may be a significant concern.

One solution would be to revise the object request interface to require the specification of a *path* anchored at a mutable so that the reachability test can be explicitly performed. Regrettably, this doesn’t help; an attacker with access to a client can extract such a path as easily as they can extract the configuration name.

A second solution might be to encrypt the cryptographic names stored in the client workspace using the client’s secret key. If the secret key is compromised, the attacker can obtain anything in any case, so this is effectively the best that is achievable. We are, however, uncomfortable with this solution, as it does not solve the problem for content stored in local repositories.

A third solution is to have each repository maintain an inverse mapping from every frozen object to its set of “containing” mutable objects. This is clearly feasible, but we are hoping for a simpler solution.

At this point, we consider this problem “still unsolved.” A number of workable strategies have been proposed, but it is unclear how best to address the issue. For our own use in open source projects, the problem is not pressing.

7.2 Mutable Names

As originally designed, mutable object names did not include the name of their originating repository. This yielded the possibility that a mutable object could be forged by providing completely false, signed content and binding it to the name of the original mutable. This flaw was recognized and diagnosed independently by Mark Miller and Chris Riley prior to the first code release. Miller provided the solution, which is to include the mutable’s name (including the repository name) as part of its signed content. This solution is incorporated in the first OpenCM release.

7.3 Server Compromise

It is of course possible for a repository server to be compromised. If the repository’s private signing key is stolen, false content can be introduced in the repository or existing content can be destroyed. While OpenCM cannot eliminate this vulnerability, it does provide a means for recovery. Mutual replication between two repositories can ensure that deleted content is recoverable. Audit can, with some pain, determine what has been changed improperly, allowing it to be removed or recovered. Registry updates can then be used to introduce a new signing key while preserving the repository identity.

8 Future Plans

OpenCM is currently working, and has been in use in our lab for several months on a number of software projects. While it is meeting our needs for file-based development, a number of opportunities exist for future enhancement. Of these, the most pressing is the need for a secure scripting language.

Scripting is needed in OpenCM for two reasons. First, various transformations on data streams can usefully be done on checkout and commit. It would be useful if the implementation of these transforms can be accomplished in a machine-independent way but outside of the OpenCM TCB (which is already too large for comfort). Second, there are automatable consistency, access control, and process enforcement policies whose enforcement we would like to embed in the tool, but in many cases these policies are project-specific. Use of a safe scripting language seems like a reasonable approach. For this application we are considering integration of W7, a Scheme-derived security kernel created by Jonathan Rees [Ree96]. We are also considering integration of a native implementation of the E capability-secure scripting langage [MMF00], whose syntax may prove more approachable to many users.

We are also interested in creating an OpenCM client for workspace-oriented programming languages, as has been done for (among others) VisualAge Java and SmallTalk.

9 Related Work

9.1 CM Systems

There is a great deal of related prior work on configuration management in general. As this paper focuses on access control, we synopsize it only briefly here. Interested readers may wish to examine the more detailed treatment in the original OpenCM paper [Sha02] or various other surveys on this subject.

RCS and SCCS provide file versioning and branching for individual files. Both provide locking mechanisms and a limited form of access control on locks (compromisable by modifying the file). Neither provides either configuration management or substantive archival access control features. Further, each ties the client name of the object to its content, making them an unsuitable substrate for configuration management.

NUCM uses an information architecture that is superficially similar to that of OpenCM [dHHW96]. NUCM "atoms" correspond roughly to OpenCM frozen objects, but atoms cannot reference other objects within the NUCM store. NUCM collections play a similar role to OpenCM mutables, but the analogy is not exact: all NUCM collections are mutable objects. The NUCM information architecture includes a notion of "attributes" that can be associated with atoms or collections. These attributes can be modified independent of their associated object, which effectively renders every object in the repository mutable. NUCM does not provide significant support for archival access controls or replication.

Subversion is a successor to CVS currently under development by Tigris.org [CS02]. Unlike CVS, Subversion provides first-class support for configurations. Like CVS, Subversion does not directly support replication. Subversion's access control model is based on usernames, and is therefore unlikely to scale gracefully across multi-organizational projects without centralized administration.

WebDAV The "Web Documents and Versioning" [WG] initiative is intended to provide integrated document versioning to the web. It provides branching, versioning, and integration of multiple versions of a single file. When the OpenCM project started, WebDAV provided no mechanism for managing configurations, though several proposals were being evaluated. Given the current function of OpenCM, OpenCM could be used as an implementation vehicle for WebDAV.

BitKeeper incorporates a fairly elegant design for repository replication and delta compression. To our knowledge, it does not incorporate adequate (i.e. cryptographic) provenance controls for high-assurance development. Further, it does not address the trusted path problem introduced by the presence of untrusted intermediaries in the software distribution chain.

9.2 Other

Various object repositories, most notably Objectivity and ObjectStore, would be suitable as supporting systems for the OpenCM repository design. This is especially true in cases where an originating repository is to be run as a distributed, single-image repository federation. Neither directly provides an access control mechanism similar to OpenCM.

Both Microsoft's "Globally Unique Identifiers" and Lotus Notes object identifiers are generated using strong random number generators. Miller *et al.*'s capability-secure scripting language *E* [MMF00] uses strong random numbers as the basis for secure object capabilities. The `Droplets` system [Clo98] by Tyler Close has adapted this idea to cryptographic capabilities encoded in URLs.

The Xanadu project was probably the first system to make a strong distinction between mutable and frozen objects (they referred to them respectively as “works” and “editions”) and leverage this distinction as a basis for replication [SMTH91]. In hindsight, the information architecture of OpenCM draws much more heavily from Xanadu ideas than was initially apparent. The OpenCM access control design is closely derived from the Xanadu Clubs architecture[SMTH91], originally conceived by Mark Miller.

OpenCM’s use of cryptographic names was most directly influenced by Waterken, Inc’s *Droplets* system [Clo98]. Related naming schemes are used in Lotus Notes and in the GUID generation scheme of DCE.

10 Acknowledgements

The Xanadu Clubs architecture [SMTH91] was originally conceived by Mark Miller and subsequently refined by Jonathan Shapiro. Comments and feedback on this paper were provided by David Chizmadia, Mike Hilsdale, Mark Miller, Chris Riley, and Anshumal Sinha.

Mark Miller’s diagnosis of the mutable substitution problem came at a critical and fortuitous moment *before* we shipped the first release. At a minimum, it saved us the embarassment of an incompatible version 2 shipping weeks after version 1.

11 Conclusions

OpenCM supports the requirements of high-assurance development in an open-source environment. It uses cryptographic naming and authentication to achieve distributed, disconnected, access-controlled configuration management across multiple administrative domains and to provide strong integrity guarantees. OpenCM supports multi-organizational project teams through use of domain-agnostic cryptographic authentication and disconnected commit. It also provides delegation and strong provenance tracking.

While there are many interdependencies in the design, there are no clever or excessively complicated algorithms or techniques in the system. The fundamental insight, such as it is, is that successful distribution and configuration management can be built on only two primitive concepts – naming and identity – and that cryptographic hashes provide an elegant means to unify these concepts and provide a basis for integrity checks.

The OpenCM schema is not limited to configuration management applications. It is a general-purpose information model that provides wide-area, integrity-checked distribution and naming system for online archival content. Further, it is relatively neutral with respect to demands on the underlying storage-system. The one serious “missing link” in the existing OpenCM architecture as a general-purpose content substrate is the absence of a self-assuring, eventually consistent collection mechanism; we believe we see a means to realize such collections. It is our plan to pursue the use of the underlying architecture for other information spaces.

The core OpenCM system, including command line client, two local file system repository implementations, and remoting support, consists of 19,134 lines of code. Roughly 20% of this code is serialization support that could be automatically generated. In contrast, the corresponding CVS core is 52,055 lines (both sets of numbers omit the diff/merge, RCS, compression libraries, comments, and blank lines). In spite of this simplicity, OpenCM works reliably, efficiently, and effectively. It also provides greater functionality and performance than its predecessor. One of the significant surprises in this effort has been the degree to which a straightforward, naive implementation has proven to be reasonably efficient.

OpenCM was released at the USENIX 2002 conference. Software is available from the OpenCM web site at `http://www.opencm.org` or the EROS project web site at `http://www.eros-os.org`.

References

[Ber90] B. Berliner. CVS II: Parallelizing software development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 341–352, Berkeley, CA, 1990. USENIX Association.

[Clo98] Tyler Close. Droplets, 1998.

[CS02] Ben Collins-Sussman. The subversion project: Building a better cvs. *The Linux Journal*, February 2002.

[DA99] T. Dierks and C. Allen. The TLS protocol version 1.0, January 1999. Internet RFC 2246.

[Dav01] Don Davis. Defective sign & encrypt in S/MIME, PKCS7, MOSS, PEM, PGP, and XML. In *Proc. 2001 USENIX Technical Conference*, Boston, MA, June 2001. USENIX Association.

[dHHW96] A. Van der Hoek, D. Heimbigner, and A. Wolf. A generic peer-to-peer repository for distributed configuration management. In *Proc. 18th International Conference on Software Engineering*, Berlin, Germiny, March 1996.

[FK89] David C. Feldmeier and Philip R. Karn. UNIX password security - ten years later. In *CRYPTO*, pages 44–63, 1989.

[Hal94] Neil M. Haller. The S/KEY one-time password system. In *Proceedings of the Symposium on Network and Distributed System Security*, pages 151–157, 1994.

[ISO98] *Common Criteria for Information Technology Security*. International Standards Organization, 1998. International Standard ISO/IS 15408, Final Committee Draft, version 2.0.

[MAM95] Daniel L. Mcdonald, Randall J. Atkinson, and Craig Metz. One time passwords in everything (opie): Experiences with building and using stronger authentication. In *Proc. 5th USENIX Security Symposium*, Salt Lake City, UT, 1995.

[MMF00] Mark S. Miller, Chip Morningstar, and Bill Frantz. Capability-based financial instruments. In *Proc. Financial Cryptography 2000*, Anguila, BWI, 2000. Springer-Verlag.

[MT79] Robert Morris and Ken Thompson. Password security: A case history. *CACM*, 22(11):594–597, 1979.

[Pol96] J. Polstra. Program source for cvsup, 1996.

[Ree96] Jonathan A. Rees. A security kernel based on the lambda-calculus. Technical Report AIM-1564, 1996.

[Sha02] Jonathan S. Shapiro. CPCMS: A configuration management system based on cryptographic names. In *Proc. FREENIX Track of the 2002 USENIX Annual Technical Conference*. USENIX Association, 2002.

[SMTH91] Jonathan S. Shapiro, Mark Miller, Dean Tribble, and Chris Hibbert. *The Xanadu Developer's Guide*. Palo Alto, CA, USA, 1991.

[SSF99] Jonathan S. Shapiro, Jonathan M. Smith, and David J. Farber. EROS: A fast capability system. In *Proc. 17th ACM Symposium on Operating Systems Principles*, pages 170–185, Kiawah Island Resort, near Charleston, SC, USA, December 1999. ACM.

[WG] E. James Whitehead, Jr. and Yaron Y. Goland. WebDAV: A network protocol for remote collaborative authoring on the web. In *Proc. of the Sixth European Conf. on Computer Supported Cooperative Work (ECSCW'99), Copenhagen, Denmark, September 12-16, 1999*, pages 291–310.

[Wu99] Thomas Wu. A real-world analysis of kerberos password security. In *Proc. 1999 Internet Society Network and Distributed System Security Symposium*, February 1999.

[Ylo96] Tatu Ylonen. SSH — secure login connections over the Internet. pages 37–42, 1996.