Fine-Grained Access Control in the Chirp Distributed File System

Patrick Donnelly and Douglas Thain
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, Indiana 46556
Email: pdonnel3, dthain@nd.edu

***Abstract*—Although the distributed filesystem is a widely used technology in local area networks, it has seen less use on the wide area networks that connect clusters, clouds, and grids. One reason for this is access control: existing filesystem technologies require either the client machine to be fully trusted, or the client process to hold a high value user credential, neither of which is practical in large scale systems. To address this problem, we have designed a system for fine-grained access control which dramatically reduces the amount of trust required of a batch job accessing a distributed filesystem. We have implemented this system in the context of the Chirp user-level distributed filesystem used in clusters, clouds, and grids, but the concepts can be applied to almost any other storage system. The system is evaluated to show that performance and scalability are similar to other authentication methods. The paper concludes with a discussion of integrating the authentication system into workflow systems.**

***Keywords*-distributed; filesystem; authentication; grid; proxy; ticket;**

I. Introduction

Large scale distributed computing systems such as clusters, clouds, and grids provide end users with access to virtually unlimited computing power at the touch of a button. However, a challenge of operating in these environments is that trust decreases as the scale of a system grows. The user of a small private cluster usually has a high degree of trust in the integrity of every single node. But, the user of a campus-scale computing system or an international computing grid cannot even name every node of the system, much less trust every single one with high value credentials.

This lack of trust makes it particularly difficult to scale up distributed file and storage systems beyond a certain size. Existing distributed systems either require that a remote node be trusted completely (as in NFS [1]) or that each client holds a high value user credential (as in AFS [2]). For a batch job to access the filesystem, this high value credential must be attached to the job, stored in a queue, and transferred to multiple remote execution sites. If the credential is captured by a malicious party, they will be able to act as the user across the entire filesystem until the credential expires. As a result, the distributed filesystem concept has seen little use across truly large-scale, wide-area distributed systems.

To address this problem, we have designed a system for fine-grained access control which dramatically reduces the risk of attaching credentials to batch jobs. We have implemented this system in the context of the Chirp [3] user-level distributed filesystem, but the concepts can be applied to almost any other storage system. The unprivileged end user can generate tickets that provide specific, limited access rights to directories on that server. Tickets can then be attached to jobs that run in wide area distributed systems. As the jobs run, they transparently access data through the filesystem interface (via Parrot [4] or FUSE [5]), but are limited to the rights indicated by the ticket. If the execution site is compromised or the ticket captured in any way, the attacker only gains the rights associated to that particular job, and not the user's entire range of abilities. If connected to a data-aware workflow system, the tickets can be generated automatically without any user intervention, thus transparently improving system security.

This paper describes our implementation of fine-grained access control in the Chirp distributed fileystem. In Section II, we give background information on Chirp and its supported authentication and access control mechanisms. Section III, we describe the cryptographic fundamentals of our work, and indicate when and where an attacker can attempt to exploit the system. In Section IV, we give details of the implementation of the ticket system within Chirp, and briefly outline the user interface. In Section V, we evaluate the performance scalability of the system, and show that it can scale at least to 100,000 outstanding tickets without significant performance impact. For Section VI, we discuss how the system can be integrated into the larger context of a user's workflow as in Figure 1. We close with a discussion of Related Work.

II. Background

A. Chirp

The Chirp [3] distributed filesystem is designed for exporting access to file data for use in grid computation. Chirp runs as a regular user process that makes available a directory tree on the local filesystem for networked access. Strong and flexible security mechanisms protect data through per-directory Access Control Lists (ACLs) and multiple authentication modes. Figure 2 gives a general overview of multiple software stacks of a single client accessing Chirp.

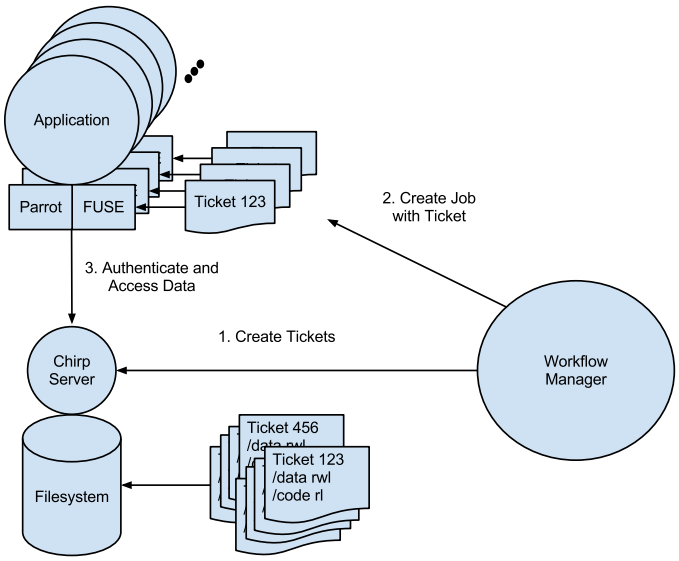

Figure 1: Workflow of Grid Jobs Using Ticket Authentication to Access the Chirp Distributed Filesystem

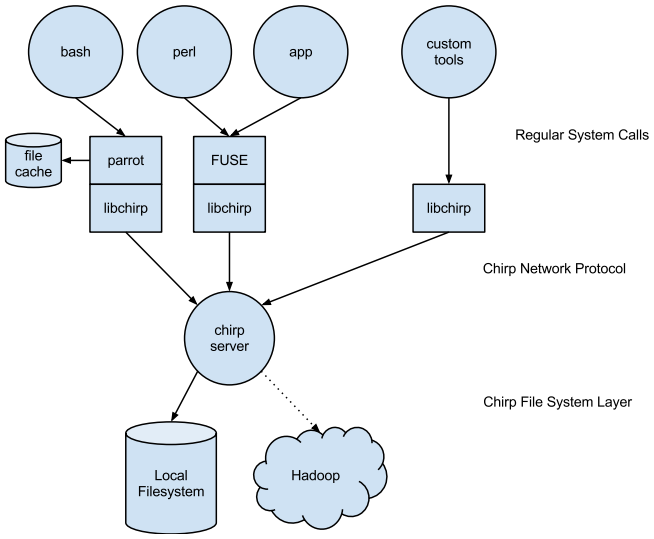

Figure 2: Chirp Distributed Filesystem

B. Authentication Options

The Chirp filesystem is designed to manage data for scientific workflows in a distributed environment. Chirp provides several authentication mechanisms to access this data easily and securely from compute nodes. Each authentication option determines the **subject** of the client. A subject is a tuple containing the authentication mechanism used to authenticate and the identity of the client. This is represented by a string as `mechanism:identity`.

In practice, it is common for many or all of the available authentication mechanims to be used. It is important to note the subject generated by the authentication mechanism does not grant any inherent rights by itself. Data is protected by access control lists, discussed in Section II-C, which give rights to an explicit subject.

Here, we discuss each authentication mechanism to familiarize the reader with current methods used to allow access to data on a grid.

- **Hostnames** The most flexible mechanism to authenticate with a Chirp server is to use hostname masks. Using a Reverse Domain Name Service (rDNS) query, the client's fully qualified domain name (FQDN) becomes the subject for the session. As an example, a client connecting from `foo.cse.nd.edu` would have the subject `hostname:foo.cse.nd.edu`.
- **Unix** Chirp also allows the user to authenticate as their system username. The server gives the client a random filename and requests the client to create the file. The owner of the created file is the subject of the client. By default this filename will be in a temporary directory such as `/tmp` on Unix. This will obviously only work when the client and server are on the same machine. A more useful setup would use a shared filesystem as a rendezvous location. On our campus grid, where we have AFS [2] installed, we make available a standard campus-writeable directory where all files used for authentication are created.
- **Kerberos** A user may also authenticate with Chirp directly using Kerberos. To setup Kerberos authentication, a Chirp server must have *superuser* privileges to access the system's host certificates. This establishes the Chirp server as a regular Kerberos compatible service which can authenticate with users.
- **Globus** Globus GSI authentication can be achieved by starting the Chirp server either with a regular user's

proxy certificate generated via `grid-proxy-init` or with the system host certificate accessed as *superuser*. Users may then authenticate normally with the Chirp server with a Globus credential.

C. Access Control Lists

Chirp offers per-directory access control lists (ACL) to manage access of data by an authenticated client. The following access rights are supported:

```
r    read          l    list
w    write         d    delete
p    put           a    admin
v    reservation
```

The rights above are self explanatory except for *reservation*. The *reservation* right allows a subject to reserve (create) a directory with the parenthesized rights following the `v`. The subject gains these rights in the new directory. For example, `v(rwlda)` would allow a subject to create a directory where the subject has the rights `rwlda`.

These access rights are set and retrieved on a directory using RPC `setacl` and `getacl`, respectively. A typical command to change the ACL might look like:

```
setacl / kerberos:pdonnel3@nd.edu rwl
```

The server maintains an ACL file for each directory which contains a variable number of subject and rights tuples. Subjects may include the globbing character `*`. The following is an example ACL file that is used in the root directory of a shared Chirp store:

File 1 An example Access Control List file.

```
unix:condor lda
unix:admin lda
unix:* lv(rwlda)
globus:/O=Computing_Lab/* lv(rwlda)
hostname:*.nd.edu lv(rwlda)
```

Later on we will use Chirp's access control list functionality to add restrictions for proxy tickets.

III. Desired Mechanism for Proxied Authentication in a Grid Environment

A grid is composed of various domains with local policies for authentication. On a university campus, this is an acute situation where many researchers and departments have their own systems with incompatible or private policies. It is not uncommon however for the researchers to form a network of machines which share computational resources for mutual gain. A typical setup may use Condor [6] to manage such a network of resources.

Further, it is common to have temporary services setup to manage various aspects of distributed computation on such a grid. Examples of such services include master worker frameworks or filesystem export servers such as Chirp. These services are transient by nature. It is, therefore, impractical to give them a dedicated ticket which is recognized by a centralized authority, such as Kerberos.

We consider the following desirable attributes when designing an authentication system which is easier to use in a grid.

- **Transferable** The ability to transfer a ticket with a job for access to a shared data resource such as NFS or AFS. This ticket should not require interactive assistance from the job submitter.
- **Ease of Use** A user should be able to allocate tickets with the service in a simple manner. A ticket should be a simple file that is transferred with a job.
- **Temporary** A ticket should have a limited lifetime to minimize risk if stolen. This is a common attribute for ticket authentication systems such as in Kerberos and Globus.
- **Secure** A ticket file should give as much assurance as possible that the client represents the user. Credentials should have limited authorization rights to lower risk.

We have developed a system which allows for the creation of a restricted proxy (hereafter simply called *ticket*) for a presently authenticated user. It is important to note the distinction between a ticket of an authenticated user versus a ticket of a user credential. A ticket for a user credential may contain a tuple of the shared session key, an expiration time, various restrictions, and is signed by the user's credential. This mechanism implies we have a method for validating such a credential, if the user even has one. On a campus grid, this is not the case. In contrast, an authenticated user is already recognized by the server and may *register* a ticket for authentication attempts by other clients.

For the Chirp filesystem, when a user is already authenticated, we would like to allow the user to designate a temporary ticket which enables proxying of the current authenticated subject. We will accomplish this through the use of public key encryption and restricted access control.

We have a few reasons for choosing public key encryption for authentication over other mechanisms such as shared symmetric key encryption or hashed message authentication codes (HMAC). First, we view storing the secret key on a Chirp server as an unnecessary risk as a compromise of the key on one server will compromise it everywhere else. The ability to have a ticket recognized (registered) by multiple Chirp servers is deliberate. Second, because Chirp sessions are not encrypted, transmitting the secret key in the clear exacerbates the risk of discovery. Finally, public key encryption allows for collaborators to exchange public keys to register manually on a server without revealing any secret key.

A. The Mechanism

A user which is presently authenticated with the Chirp server may present a public key to the server which is to allow another client to authenticate as the subject of the user. Authentication using the private and public key pair resembles SSH [7] authentication except it will not be mutual: the server is not validated by the client. The server will use standard *challenge response authentication* to confirm the client has the private key corresponding to the registered public key. From now on, we will refer to the ticket residing on the server as a **registered ticket**.

As part of registration of a ticket the user prescribes an expiration time after which any authentication attempt will fail. Further, clients authenticated using an expired ticket will have all subsequent file operations fail upon checking the ACL.

B. Authentication Steps

A user, *A*, must first create a private and public key pair which we denote as *SKA* and *PKA* respectively. These keys will be used for *challenge response authentication*. First, it is necessary to *register* the public key with the server, *CS*. The user also sets an expiration time *t*. Note that the user is already authenticated with the server when registering the public key.

$$A \rightarrow CS \quad PKA, t, A \tag{1}$$

Following registration of the public key, the user modifies the *access control rights* of the ticket. This is represented as a variable number of messages which change the *ACL mask* . The user uses a *hash* digest as an identifier for the ticket in subsequent messages. Initially a registered ticket has no access rights.

$$\begin{array}{ll} A \rightarrow CS & \{PKA\}_{hash}, path_1, mask_1 \\ A \rightarrow CS & \{PKA\}_{hash}, path_2, mask_2 \\ A \rightarrow CS & \{PKA\}_{hash}, path_3, mask_3 \end{array} \tag{2}$$

Once a ticket is registered, a client may use the ticket to authenticate with the server. This is done by querying the server first if it has a ticket registered with the given digest.

$$A \rightarrow CS \quad \{PKA\}_{hash} \tag{3}$$

If the server has a registered ticket with a matching *hash digest* of the corresponding public key, it allows authentication to continue and issues a challenge, or nonce.

$$CS \rightarrow A \quad I_{CS} \tag{4}$$

The client receives the challenge and will sign it with the ticket's private key. The result is transmitted to the server.

$$A \rightarrow CS \quad \{I_{CS}\}_{SKA} \tag{5}$$

After verification of the signed nonce, the server considers the authentication successful.

C. Recognized Vulnerabilities

Mutual authentication is absent in this proposed design although it does not need to be. In a system desiring mutual authentication, users registering a ticket will already be authenticated with a server; the users have the public key of the server to store with the ticket. The authentication steps then should follow the standard challenge response authentication outlined by Needham and Schroeder [8] for mutual authentication using Public Key algorithms. The client and server will not need to consult an authentication server (AS) to confirm their respective public keys because they are stored within the ticket during registration.

For our systems, the other Chirp authentication modes do not validate the identity of the server and our evaluation of the risk of masquerading servers in a typical grid environment is small. Further, in a cycle scavenging setup using a job submission platform like Condor, it is difficult to confirm the integrity of computation results and so confirming the source of data does not add any security to the overall system. Chirp servers also work on unencrypted channels so man-in-the-middle attacks are still a concern. So, we do not feel mitigating this risk warrants the added complexity to the system. In contrast, restricting access to data is of great interest.

Hash Collisions of tickets allows for the replacement of your own or another user's ticket. With a suitably strong hashing algorithm, this is exceeding unlikely. The worst case scenario is a job's inability to authenticate.

Theft of a ticket is central to the design considerations of the system. We resolve this open problem through expiration times of tickets and restricted access control.

IV. Portable Ticket Authentication in the Chirp Distributed Filesystem

The first step in setting up a ticket with a Chirp server is the creation of a public and private key pair. When we refer to the *ticket*, we are referring to the private key which clients use to authenticate. Next, the public key is *registered* with a server through an RPC, see Figure 3, which associates the key with your authenticated subject. In addition to the public key, a ticket is registered with an expiration time and a *subject*. Typically this subject is `self` which refers to the subject of the authenticated client. The client may register tickets for other subjects if the client is the *superuser* of the Chirp server (set via an option to the Chirp server).

When a client authenticates using a ticket, it becomes the subject of the ticket. That is, the subject you assume when authenticated with the Chirp server is the subject of the ticket and any client authenticating using that ticket. This subject is used when testing access control rights within a directory (subject to restrictions discussed below).

After a ticket is registered, it initially has no rights associated with it and can not be used to access data on the server. It is necessary to change the access control list

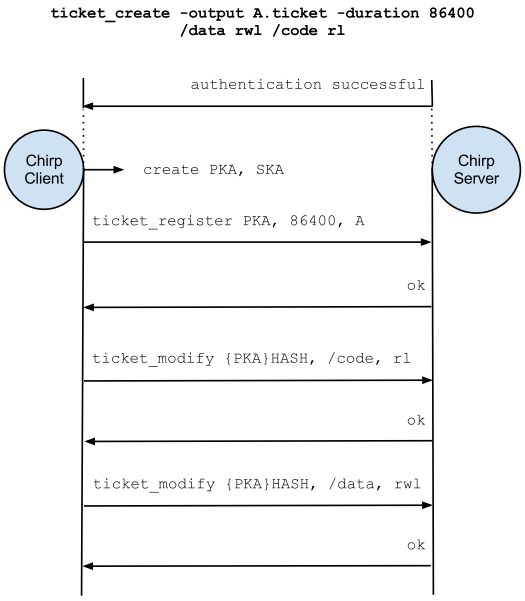

Figure 3: Creating a Ticket in Chirp

rights of the ticket to enable access. This is done through a `ticket_modify` RPC which sets an *ACL mask* for a given path. Recall from Section II-C that each directory has an ACL file which describes the access rights of various *subjects*. When resolving the access control rights of a client authenticated using a ticket, we mask the access rights of *ticket subject*, or the subject that registered the ticket, with the *ACL mask* of the ticket.

As an example, if we have a user *bob* that uses Unix authentication to register a ticket, the subject of that user is `unix:bob`. *bob* modifies the access rights of his ticket to have `rl` in the `/foo` directory. Assume `unix:bob` has `rwl` permissions in the `/foo` directory. When a client authenticates using *bob's* ticket, the access rights in the `/foo` directory for the ticket authenticated client will be the logical conjunction (AND) of the ticket's *ACL mask* and the ACL of `unix:bob`. This would result in `rl` permissions.

These operations are conveniently packaged with a `ticket_create` command within the Chirp client toolset:

```
ticket_create -output foo.ticket \\
    -bits 1024 -duration 86400   \\
    / rl /foo rwl
```

This command first creates a public and private RSA key pair. The public key is then registered with the corresponding access rights.

A. Managing Chirp Tickets

Once a ticket is registered, the user must give the ticket access to directories by setting an *ACL mask*. This is done using the `ticket_modify` RPC which may be used like so:

```
ticket_modify foo.ticket / rl
```

The above command gives the ticket *read* and *list* access to the root directory so long as the user who registered the ticket retains those access rights.

One can delete a registered ticket the `ticket_delete` RPC. Like `ticket_modify`, the subject of the ticket must match the subject of the current session. As an example command:

```
ticket_delete foo.ticket
```

A user may list the tickets registered on a server using the `ticket_list` RPC. The command returns a list of *ticket identifiers* which are essentially *hash digests* of the public keys of each registered ticket. These identifiers are usable in any `ticket_` RPC in place of the client ticket filename.

```
ticket_list self
```

The `ticket_get` RPC returns the information for a registered ticket including the *ticket subject*, public key, the expiration time left in seconds, and a variable number of *ACL masks*.

```
ticket_get foo.ticket
```

B. Authenticating with a Chirp Ticket

Authenticating using a ticket is as simple as transferring the ticket file with your job. Chirp provides options and environment variables as part of the client toolset to list the tickets to use for authentication. Generally, applications will not use the client toolset directly and instead will utilize Parrot [9]. Parrot is another application tool we develop used to transparrently connect an application to Chirp by intercepting I/O system calls and redirecting them. Parrot, through the use of the *libchirp* library, offers support for ticket authentication.

Figure 4 shows the process of authenticating with a Chirp ticket. The application will issue an IO system call which is redirected by Parrot to *libchirp* . *libchirp* will use the ticket, *foo.ticket*, to attempt authentication to the Chirp server. The authentication process itself is handled by sending the digest of the public key of the ticket. Recall that the ticket itself is simply a private key which can be used to generate the public key on demand. The digest is used for content addressable storage on the server. Upon receiving the digest, the Chirp server will attempt to open a top-level file `.__ticket.X` where `X` is the digest of the public key. This file contains the registered public key, expiration date, *ACL masks*, and other metadata.

If the Chirp server is successful in opening the registered ticket file, it will confirm the ticket is still valid by asserting the current time is less than the expiration time of the ticket. If the ticket is still valid, it will respond to the client with a cryptographic nonce (one-time-use random number) which the client must sign with its private key. Once the client

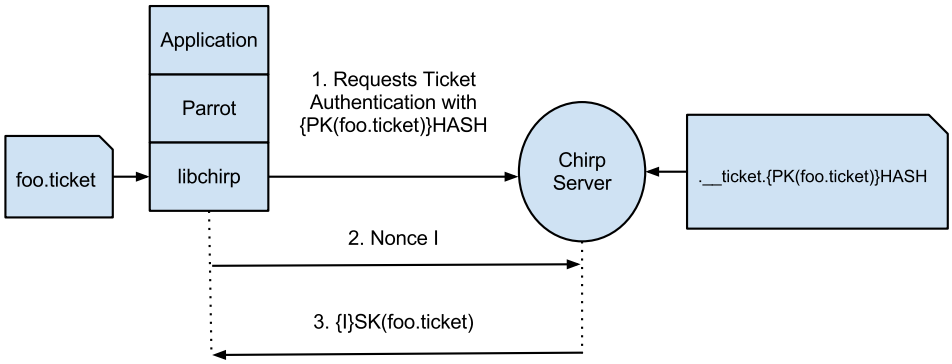

Figure 4: Authentication using a Chirp Ticket

transmits the signed nonce, the server confirms the signature and considers the authentication successful.

After authentication, most file operations require a check of a directory's ACL to confirm access rights. Any time this occurs the ticket is consulted to mask the rights of the user with *ACL mask* of the ticket. This prevents a user's ticket retaining rights the user may lose in the future. In addition to checking the *ACL mask*, the Chirp server will also confirm the expiration time for the ticket has not been exceeded since the client authenticated.

C. Garbage Collection of Tickets

Because the server maintains a list of tickets which expire, it is worth noting that garbage collection is necessary for the system to function. Periodically the server will go through each ticket it has registered and discard the expired.

V. Evaluation

Of particular interest for the evaluation of this authentication system is the ability to scale. We have conducted tests to analyze the cost of using tickets while doing various file operations. We also want to compare the latency and throughput of the authentication mechanisms.

Figure 5 shows the scaling of various operations: creating a ticket, reading an empty file, reading a 16MB file, and listing a registered ticket (`ticket_list`). The authentication is done over the *localhost loopback* interface with the server exporting a Linux *ext4* filesystem. We attribute the linear scaling for creating tickets to the filesystem cost of inserting files in a directory.

Authentication System	Latency (milliseconds)
Hostname	6
Unix	19
Globus	95
Ticket	89

The latency and throughput give an idea of the number of authentication attempts a client can perform serially and in parallel. Testing of the latency of each authentication system is also done over the *loopback* interface while the throughput scaling tests were done over our campus grid. The throughput is measured from continuous connections from several clients over a period of time. Hostname authentication can be viewed as a speed of light test for the system as most authentications succeed with a hostname cache hit. Using this graph, we can approximate the number of concurrent clients we can support.

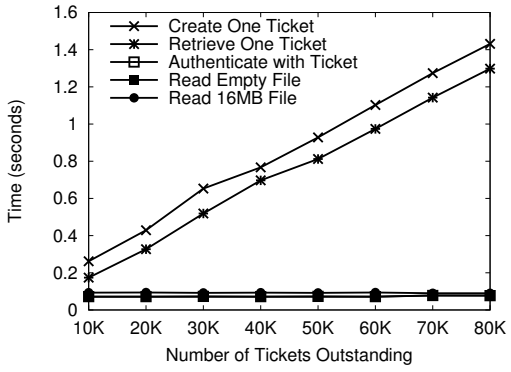

Figure 5: Various File Operations as Number of Registered Tickets increases in thousands.

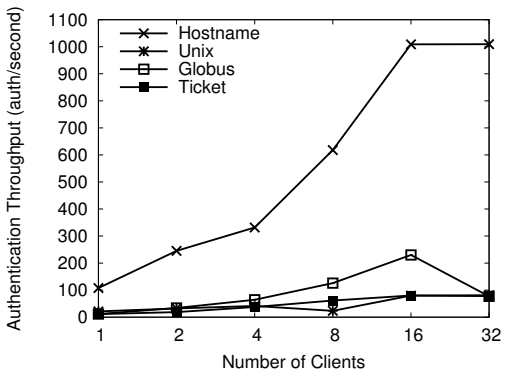

Figure 6: Authentication Throughput

VI. Complete System and Future Work

To setup a complete workflow using a ticket, a user identifies data needs manually and runs the necessary command to create a ticket with each server. As an example scenario, a user with a job that requires access to the `/data` directory would make a ticket `ticket_create -o my.ticket /data rwl`. They may then construct a job which includes `my.ticket` when dispatched to the compute node.

Jobs may retrieve data through Chirp command line tools or `libchirp` or through system call redirection via Parrot which offers a global file namespace for all catalogued Chirp servers. A client could then access data directly from the filesystem via filename constructions such as `/chirp/hostname:port/data`. As a final step to running the workflow, a user may delete the ticket manually using `ticket_delete` or let it expire naturally.

As future work we intend to augment ticket authentication to integrate more fully with our existing workflow managers. In particular, we desire tickets to be automatically created on behalf of the user based on the data needs of the task to be run. So, as in Figure 1 on page 2, we envision a workflow manager which contacts data servers to create a single ticket with restricted authorization for all the data a job needs. This new system will utilize tickets registered on multiple servers. Jobs will then have a single credential which is recognized by different servers without increasing risk.

VII. Related Work

There are a number of factors to consider when creating a suitable portable ticket authentication mechanism for a distributed filesystem. For our purposes, we confine our considerations in the context of a grid environment where we want to do computing, probably using some batch job submission platform, such as Condor [6].

The presumed **setting** where the filesystem resides impacts how authentication is handled. Distributed filesystems such as Hadoop [10] assume a local network where claimed user identities are implicitly trusted. On the other end of the spectrum, we have AFS [2] and Ceph [11] which are designed for access by thousands of concurrent users, e.g. on a campus grid, where authentication must be strictly enforced; these systems must provide for data integrity and confidentiality. AFS solves this by incorporating Kerberos [12] which independently and securely handles user authentication. Ceph also uses a scheme similar to Kerberos [13].

While AFS and Ceph are suitable as a data store in a managed grid or cluster, Chirp is designed for a grid where data for a distributed job system may be accessed in a temporary fashion and where the data is usually located on a resource not managed directly by the organization. In this environment, it is important for data to be protected from other users of the grid while having minimal infrastructure requirements so the data can be easily accessed by jobs on other nodes in the grid. In particular, Chirp does not require an authentication structure where the server and clients have persistent credentials. Both clients and the server are assumed to have transient location.

Distributed filesystems generally recognize only one **authentication** mechanism. LWFS [14] and Chirp notably depart from this style by allowing multiple authentication schemes including Kerberos and Globus GSS-API [15].

Access control has varying granularity across distributed filesystems. Granularity for access may be limited to file blocks, objects, files, and directories. Filesystems usually choose granularity based on how data is distributed to data nodes. LWFS and Object Storage Devices, complying to the ANSI T10 SCSI OSD standard [16], use coarse grained objects for rights associated with a capability. PVFS [17] implements POSIX Access Control Lists which provide fine-grained access control on a per-file or per-directory basis. Chirp also provides Access Control Lists which are managed on the exported filesystem. As discussed in Section II-C, the ACL is a simple file managed in each directory. This is necessary for allowing the Chirp server to be easily torn down and restarted or for allowing multiple front Chirp servers to export the same back-end filesystem.

Delegated access is an especially useful feature in a grid environment where the distributed filesystem allows access by otherwise unprivileged users or by ”anonymous” compute jobs. Kerberos version 5 now has a user ticket field, *authorization-data*, which provides arbitrary and additive restrictions interpreted by the end application [18]. The user-generated *proxy ticket* is tied to a specific client and host service as it is a regular Kerberos ticket with added restrictions. Unfortunately, such a system creates unreasonable requirements on the user/system generating the delegated credentials as the future clients may not be known ahead of time. Additionally, which data the job requires access to may not be known at job submission time.

Ceph provides a protocol, Maat [13], which allows the creation of credentials that may be delegated to other clients. This system is the most similar to the approach adopted by Chirp but differs in a few significant ways. For both systems, a delegated credential is a private/public key pair that proves authenticity. For each file the user wishes to delegate access to, Ceph provides a capability associating the hash of the public key with the rights to the file and a lifetime token. In contrast, Chirp manages ticket lifetime and access control on the exported filesystem as it does with regular Access Control Lists; clients with the ticket do not need a signed credential from the Chirp server indicating the rights of the ticket. Jobs also only need to carry the ticket, or private key, to authenticate.

Ceph also requires that rights to files for a delegated credential be known during creation. Chirp allows the dynamic modification of the ACL mask for a ticket using the

`ticket_modify` RPC.

In Ceph, delegated credentials associate rights with the hash of the public key. As a result, the private/public key pair temporarily has access rights to files independent of the user which created the credential. For Chirp, a ticket has a number of access right masks which are applied to the ticket creator's access rights. This feature is useful so ticket rights do not need to be simultaneously updated when changes to the ACL are made. Updating all of a user's tickets is an expensive operation: it requires iterating over all tickets on the Chirp server to locate all of the user's tickets.

VIII. Conclusions

Many established authentication systems exist which securely authenticate users; however, we have argued that these systems are not always suitable for grid authentication needs and that we require solutions that are agnostic of environment authentication systems while providing fine-grained authorization to user data. We have developed an authentication mechanism which fulfils these needs by providing mobile, limited-lifetime tickets with configurable access control limits. We also demonstrate how these tickets may be used in a complete environment.

Chirp is free software distributed under the GNU General Public License and can be downloaded at: http://www.cse.nd.edu/~ccl/software.

References

[1] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and implementation of the Sun network filesystem," in *USENIX Summer Technical Conference*, 1985, pp. 119–130.

[2] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West, "Scale and performance in a distributed file system," *ACM Trans. on Comp. Sys.*, vol. 6, no. 1, pp. 51–81, February 1988.

[3] D. Thain, C. Moretti, and J. Hemmes, "Chirp: A Practical Global Filesystem for Cluster and Grid Computing," *Journal of Grid Computing*, vol. 7, no. 1, pp. 51–72, 2009.

[4] D. Thain and M. Livny, "Parrot: An Application Environment for Data-Intensive Computing," *Scalable Computing: Practice and Experience*, vol. 6, no. 3, pp. 9–18, 2005.

[5] "Filesystem in user space," http://sourceforge.net/projects/fuse.

[6] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the grid," in *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and T. Hey, Eds. John Wiley, 2003.

[7] T. Ylönen, "SSH - Secure Login Connections Over the Internet," in *Proceedings of the 6th USENIX Security Symposium*. San Jose, CA: USENIX, July 1996, pp. 37–42.

[8] R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, 1978.

[9] D. Thain and M. Livny, "Parrot: Transparent user-level middleware for data-intensive computing," University of Wisconsin, Computer Sciences Department, Tech. Rep. 1493, December 2003.

[10] Hadoop, http://hadoop.apache.org/, 2007.

[11] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *USENIX Operating Systems Design and Implementation*, 2006.

[12] J. Steiner, C. Neuman, and J. I. Schiller, "Kerberos: An authentication service for open network systems," in *Proceedings of the USENIX Winter Technical Conference*, 1988, pp. 191–200.

[13] A. Leung and E. Miller, "Scaling security for big, parallel file systems," in *Proceedings of the 5th USENIX conference on File and Storage Technologies*. USENIX Association, 2007, pp. 14–14.

[14] R. Oldfield, L. Ward, R. Riesen, A. Maccabe, P. Widener, and T. Kordenbrock, "Lightweight i/o for scientific applications," in *Cluster Computing, 2006 IEEE International Conference on*. IEEE, 2006, pp. 1–11.

[15] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," in *ACM Conference on Computer and Communications Security*, San Francisco, CA, November 1998, pp. 83–92.

[16] R. Weber, "Information technology: Scsi object-based storage device commands (osd)," *Technical Council Proposal Document T*, vol. 10, p. 92, 2004.

[17] P. Carns, W. Ligon III, R. Ross, and R. Thakur, "Pvfs: A parallel file system for linux clusters," in *Proceedings of the 4th annual Linux Showcase & Conference-Volume 4*. USENIX Association, 2000, pp. 28–28.

[18] B. Neuman, "Proxy-based authorization and accounting for distributed systems," in *Distributed Computing Systems, 1993., Proceedings the 13th International Conference on*. IEEE, 1993, pp. 283–291.