

IMPROVING SECURITY IN THE INTERMEZZO FILE SYSTEM

December 9th, 2001

By
Jacob Gorm Hansen & Asger Kahl Henriksen
University of Copenhagen, Department of Computer Science

Abstract

By adding Kerberos style authentication of users and servers to the Intermezzo filesystem, it becomes feasible to use the filesystem across open untrusted networks, such as the Internet.

We have implemented a set of security enhancements, and measured the performance impact relative to a non-secured system. We found the impact of signing meta-data-operations in the local file cache negligible, but that network transfer throughput when using over-the-wire encryption was largely decreased.

.

Contents

Preface	ii
Motivation	ii
Acknowledgements	iii
I Security	1
Scope of this work	1
Classification of machines	2
II GSSAPI and Intermezzo	4
Overview of Intermezzo	4
Kerberos and the GSSAPI	4
Kerberizing Intermezzo	5
III Implementation	9
Modifications to the Intermezzo code	9
Current limitations	11
IV Findings	13
Testing security	13
Benchmarks	14
Conclusion Future work	16
A Test results	17
B Bibliography	18

Preface

Motivation

The precursor to this work is our earlier attempt to build a network of workstations (NOW) system by combining the Andrew File System (AFS) (Satyanarayanan [1990]) with the MOSIX (Barak and La'adan [1998]) process migration system. The goal was to build a system that would allow the distribution of diverse computational tasks across a general purpose local area network. To attain this goal, the following components were needed:

- A distributed file system providing a unified name-space, and an efficient and predictable caching strategy, in this case AFS .
- A load balancing and process migration system, in this case MOSIX, combined with specialised tools such as MPMake for efficient distribution of compile jobs.
- A security system protecting against both wire-tapping and forgery of traffic, in order to let the system run over an untrusted network.

The benchmarks performed showed AFS's lack of write caching to be a performance bottleneck when distributing compilations, and though the system would have been usable for other tasks than compilation, we felt that a more modern and lean file system than AFS would be in place. After researching the alternatives (NFS was inadequate due to its happy-go-lucky caching strategy which is too dangerous for distributed compilations (McClure and Wheeler [2000])), we found Intermezzo to be not quite ready for what we wanted it to do, but an interesting design and rapidly developing implementation.

Contacting the main author of Intermezzo, Mr. Peter Braam (of AFS and CODA fame), for suggestions of how we could help improve the system, led to this work on integration Intermezzo and Kerberos, via the GSSAPI.

It was our initial hope that this work would be of real value to the usability of Intermezzo, while acquainting ourselves with the inner details of this promising distributed file system.

The work we have performed has made us more intimate with the Lento cache manager (written in Perl), as well as some (though far from all) the details of implementing a file system in the Linux kernel.

Acknowledgements

We would like to thank Mr. Peter Braam for making the Intermezzo source code accessible under an open license, and for taking the time to comment on some of our uneducated guesses about how to best implement security in the system. And our advisor at DIKU, Niels Elgaard, for, well, his advice.

This crisp L^AT_EX 2_ε style was provided to us by Peter Andreasen, who modified it from the original by Christian Tønsberg.

Security

Self-preservation is the first law of nature

SAMUEL BUTLER

Scope of this work

Security is not limited to encryption. To secure a system, code must be audited thoroughly, especially code reachable from the network, and running with root privileges as the Intermezzo file server does, so that an error in the implementation will not open the whole operating system to compromise. The code must also withstand unexpected input, calling patterns, and so on, without crashing, which would result in denial of service to users.

Our intention with this work is not to do that. What we are aiming to do is mutually authenticate users towards services, and services towards other services, and protect messages between these parties against forgery and eavesdropping.

Further, since Intermezzo supports disconnected operation, updates must be signed so that their genuity can later be proven.

In order for the system to be complete, preventing release of data to unauthorised users, by not releasing it from the server, and optionally also by filtering modification logs before they reach a client, would be necessary. However, at its present stage, Intermezzo only does active replication of files, not data on demand, which makes anything else than releasing all data, and relying on client filesystem to enforce access control, impractical.

Discussions about the adequacy of the security features of the Linux kernel, and that of other operating systems, are plenty. A number of proposed changes address a number of faults, whether real or perceived.

Some systems, such as AFS, invent their own security models, residing on top of the host operating system. This has some advantages when the system has to work on a number of different platforms, but also makes the system more complex, to implement, install, and use. AFS for example, has its own ACL system, includes code for time synchronisation, and cannot be installed without first installing Kerberos.

Our purpose is not to change the Linux security model, but rather to extend it across the network. We do not introduce any changes to the operating system, and, like in NFS, we assume

that uids are synchronised between hosts. To us, the value of Intermezzo lies in its close relation to the underlying kernel and file systems, and the lack of redundancy. Linux is consistent, in that it never gives you a false impression of security, keeping things wide open when they cannot be closed. We would prefer for both to stay that way.

Classification of machines

As is done in (Satyanarayanan [1989]), it is practical to make some initial assumptions about the types of machines of which the Intermezzo system consists, to be able to better describe and understand the various threats to the overall security of the system.

In an Internet-connected Intermezzo system, every host on the Internet may be considered a possible participant, as one goal of Intermezzo is to be usable over both local and wide area networks.

Clearly, trying to control security for all hosts on the Internet is impractical, so a way of obtaining a smaller subset which can be trusted will be needed. In Kerberos, trust in a host is signified by creating a host-principal instance for the network address of the host, in the form `host/workstation@DOMAIN`, and handing the secret key for this principal to the workstation. Doing this, and demanding of all hosts connecting that they are able to present such credentials, effectively limits the set of hosts which needs to be considered.

However, the set of trusted hosts may still be large (possibly thousands of computers), and will need to be divided into several subsets for us to be able to focus our resources, and to contain the various types of compromise to affect only a limited number of hosts or users. The set may be further subdivided by looking at what type software is run, how their operation affects that of other hosts, and how they are used by human users.

Apart from the basic operating system, the different types of software interesting to us are:

- Kerberos services such as the Key Distribution Center and the Ticket Granting Service.
- Intermezzo configured as a file server.
- Intermezzo configured as a client.
- Other user or system software.

Of these types, the first two are able to greatly affect client systems relying on the correctness of their operation, meaning that compromise of one or more server systems will lead to compromise of their clients.

Compromising hosts running the latter two types of software, will not automatically lead to compromise of other hosts, but may enable the intruder to steal the identities of any users on the host, install trojans, and use the host as a base for future attacks against the rest of the network.

We are thus able to divide the set of trusted hosts into the following security-equivalent subsets:

- Kerberos servers. Compromising a Kerberos server means possibly compromising the entire network.
- Intermezzo file servers. Compromising a file server means compromising all hosts trusting the integrity of its data.
- Shared or public workstations, running user software. Since a shared workstation is typically used by multiple users, compromising it must only result in obtaining privileges contained within the union of all privileges of its users. The amount of protection afforded to such hosts should be proportional to the amount and importance of its users.
- Private workstations, among them laptops. Private workstations cannot be trusted to enforce any access control, or not to run rogue software.

Should an attacker compromise a private workstation, the damage should be limited to the unauthorised release or modification of the owner's data.

For members of the first three subsets, it must be expected that their operating systems and super-users can be trusted, and that the machines are kept secure, both by physical and technical means.

CHAPTER II

GSSAPI and Intermezzo

Computers make it easier to do a lot of things, but most of the things they make it easier to do don't need to be done.

ANDY ROONEY

Overview of Intermezzo

Intermezzo is an Open Source distributed filesystem, aiming at giving local filesystem performance on cached data. It consists of a user space cache manager (Lento) written in Perl, and a kernel module (Presto) acting as the filesystem device driver. It is currently available for Linux only.

Intermezzo works as a filter layer above a local journaling file system. It functions in coalition with a user space cache manager, whose primary task is to move data back and forth over the network.

Because Intermezzo resides atop a journaling file system, and uses its journal to keep track of all meta-data-operations, it is able to inform other computers serving the same files about changes, and supports disconnected operation, by keeping a log, called the Kernel Modification Log (KML), of all changes occurring while separated from the file server.

The KML also allows caching of write operations, and can be optimised so that redundant operations are removed before data is transmitted back to the file server.

Intermezzo replicates normal permission attributes along with files, and relies on the underlying file system enforcing them. Intermezzo also replicates extended attributes, enabling the use of POSIX Access Control Lists (ACLs) instead of the traditional User, Group, Others (UGO) scheme, as long as the appropriate patch to the Linux Kernel is applied.

Intermezzo currently only allows actively pushing data from the file server to the client. In time, it is going to support clients pulling data from the server on demand.

Kerberos and the GSSAPI

Kerberos (Miller et al. [1987]) was developed at MIT in the late eighties. It is a symmetric key system based on the Needham-Schroeder model, where knowledge of a shared secret is

proof of identity. Kerberos uses symmetric keys for encryption, originally based on the Data Encryption Standard (DES), but now also supporting other more modern types of encryption.

Along with Kerberos version 5, the Generic Security Service Application Program Interface (GSSAPI) was designed. The GSSAPI is an authentication layer, allowing the use of several underlying authentication systems, Kerberos being the most prominent. GSSAPI is not limited to symmetric-key systems, the Globus project (Globus [2000]) for example, uses GSSAPI on top of a public-key system built using SSL. In this project however, we shall focus entirely on the use of GSSAPI as an interface to Kerberos.

GSSAPI sessions typically operate by a client calling `init-sec-context`, presenting some credentials and obtaining a token which is sent to a server via a network link. The server uses the token from the client along with local credentials and calls `accept-sec-context` to obtain a new token which it passes to the client. The client can now establish the mutually authenticated context through a final call to `init-sec-context`. (this is greatly simplified, the reader is referred to (Linn [2000]) for more information).

Having a mutually authenticated context means that both client and server possess a key for signing and optionally encrypting messages, in order to prevent forgery and eavesdropping.

Kerberos principals

Every user, workstation, or service, has a Kerberos principal. Each principal has a name, which is public, and a key which is private. For user principals, the key is often a human readable password string, whereas for workstation and service principals it is often randomly generated. Users prove their identity to Kerberos by presenting the password, and services prove theirs by presenting their key, often stored in a protected area of the file system. Keys and passwords are not actually transferred across the network, and they are only needed for the initial authentication of user or service, not for encrypting actual communication, where newly generated session keys are used instead.

Kerberizing Intermezzo

The process of added Kerberos authentication and possibly message signing and encryption to a network protocol is often referred to as 'Kerberizing' it.

Kerberizing a connection-oriented protocol such as telnet is relatively simple, due to its single-user nature. Kerberizing a file system, running in the kernel of the operating system and operating on behalf of multiple users, is less trivial.

If you wish to retain the single-user model, where the credentials used for a network operation are those of the requesting user, you need to track the credentials used along with every file system operation. This is difficult in a system supporting disconnected operation, where there may be a large time span between a file operation at the client and its re-integration at the server.

As described below, we have chosen a different strategy, where a user logging in to a file server is merely registering her presence, and all file system network communication takes place between service principals.

Authorising release of data

One purpose of the security system is to prevent release of data to unauthorised clients. Currently, Intermezzo only supports an active replication mode, in which all files and changes are pushed to all clients.

Pro-actively pushing data is desirable in some situations, like for a laptop computer that will wish to cache as much data as possible, before possibly finding itself disconnected. However, since workstations cannot be trusted to enforce correct access to cached files (the holder may possess the root password, or be able to mount the disks under his own operating system), Intermezzo must take care not to release more data than is necessary to keep only the currently authenticated users of the workstation happy.

To achieve this, the file server must maintain a set of logged-in users for each client workstation, and only release a file to the client computer, if one or more of the client's users has read-access to that file.

This solution is not without problems though, as possibly large amounts of data will have to be pushed when a user logs in, or the push and pull mode be combined in a way where accessible data and meta-data is pushed as before, whereas protected data is pulled by the client workstation if it is missing.

Currently, data on demand is not fully implemented in Intermezzo, so we have decided to rely on local access control preventing unauthorised read access. We allow all data to be replicated to clients, even though they may not have valid users present. Once data on demand in Intermezzo becomes stable, a better solution can be implemented.

Still, a computer containing cached data may be stolen or compromised, and the data will be accessible to the intruder. The only real solution to this problem is to encrypt the cache file system, which should be entirely possible by mounting via an encrypted loop-back device, but will not be discussed here in detail.

Trusting file system modifications

When a file has been modified at a workstation and needs to be written back to the file server, the file server needs to determine whether to accept the modifications into its own file system.

The client file system is expected to deny modification of files for which the user does not hold write permission. However, the client workstation may be running a malicious version of the Intermezzo software, or a user may have assumed superuser identity before changing the file, whether for honest or dishonest reasons.

Since Intermezzo is able to operate disconnected, a method of proving the validity of changes in the KML, even long after the user who performed them may have logged out, is needed.

We propose the following solution;

Upon login, the file server decides on a secret key to use for communication with the user on the particular client workstation. This key is securely transmitted via Kerberos to the client login program (called `imlog`), which hands it to the kernel for safekeeping and later use. On the server, this key is kept in a secret log, indexed by time of creation, the user principal name, and the address of the client workstation.

Every entry in the KML on the client is signed using this key, as proof that the modifications were performed by a logged in user. When changes are reintegrated back into the server file system, the server is able to correlate the timestamp of the changes, the identity of the user performing them, and the corresponding key from its own secret log, to verify the signature.

If the signature verifies correctly, the changes may be reintegrated into the server file system. If not, re-integration will have to be aborted, since the client system must be assumed compromised.

In Kerberos, session keys usually expire after a period of time, typically 24 hours, to limit the time an attacker has to breach them. In a disconnected system like Intermezzo, timeouts for KML changes will have to be much longer, since a laptop may reconnect after being off-line for months, and changes should not be lost. Setting this timeout will be a compromise between security and practicality of use.

A mechanism for re-signing a client KML with a more recent key could be provided to salvage a timed-out client.

Authentication of clients and services

Before data can be exchanged between a workstation and a server, both need to mutually authenticate, using their respective credentials. Like users, Kerberos services are registered as principals, and have their own key which they share only with Kerberos. These are typically kept in a file on each workstation (called a keytab).

In Intermezzo, the following Kerberos principals will be needed:

- A user principal for each user wishing to access the file system.
- A service principal for the Intermezzo service on every file server.
- A service principal for every host wishing to talk to the Kerberos services and act as an Intermezzo client.

Lento will have to be modified to authenticate with Kerberos at connection time, as will the accepting side of Lento running on the file server.

With GSSAPI, both user and service principals may initiate connections, and even though Intermezzo file servers are the ones accepting incoming connections, it is more convenient if only they possess a special service principal, since they are the ones to be trusted by users and by client services. Initiating parties have to run the Kerberos login command (typically called

kinit) to be able to obtain session keys, so the file server startup script will have to do that, typically reading its key from a keytab file.

When a new client tries to connect to an Intermezzo file server, the server initiates the negotiation of a new security context, using its special service principal (named `lento/hostname@REALM`), which it possesses because it has been allowed to act like a server. All that is required of a client, is that it is known to belong to the Kerberos realm, and so is able to present a host principal (named `host/hostname@REALM`) for its network address.

The client and server are now said to be in a mutually authenticated context, in which all further communication takes place.

Authentication of users

The individual users must present valid Kerberos credentials to the Intermezzo file server before being allowed write access to the local file cache. If Intermezzo is running in data on demand mode, the server has the option of verifying the ACL of a file against all known users on a given workstation before releasing data to the client.

Upon successful authentication a user is given a key with which to sign each change to the filesystem, in order to allow re-integration of changes, even after expiry of the Kerberos session.

We require the user principal names to match those in the local `/etc/passwd` file.

CHAPTER III

Implementation

Nothing endures but change.

HERACLITUS

Modifications to the Intermezzo code

The implementation process added a few new modules to Intermezzo, and required changing a few others.

Presto

Presto handles all updates to the local KML file. We modified Presto to sign all records with a key based on the users uid. Presto adds a MD5 signature, based on the log-entry, and the user's key, to the end of KML record. To obtain the key, we expose two interfaces in the `/proc/sys/intermezzo` area. `/proc/sys/intermezzo/key` is a two-way interface, that, when written to updates the key-table for the current user with the contents written, and when read from, returns the timestamp associated with the current key of the current user.

The reason the timestamp is returned from the kernel, is that it is important to know exactly when a new keys comes in use, so that server signature checks do not fail due to lack of timestamp precision.

`/proc/sys/intermezzo/usekey` is used to control whether or not signing should occur, so that Lento is able to initialise the KML and other system files upon startup, without being logged in first.

Whenever a user tries to modify the Intermezzo filesystem, Presto denies the user access if he does not yet have a key registered.

Imlog

This program handles the user login process. It requires the user to have run `kinit` (the Kerberos login program) prior to login. It tries to mutually authenticate the user with the server, and upon success securely receives a key, stores it through `/proc/sys/intermezzo/key`, reads the timestamp of the key from the same location, and securely communicates this back to the server.

A user can be logged out by setting an empty key in `/proc/sys/intermezzo/key`.

Lento

Lento needed to be modified in a number of places, specifically to handle signature checks, mutual authentication with an incoming client, and signing or encryption of over the wire traffic.

A persistent key storage was implemented to store a database of (client-hostname, user-id, timestamp, key) tuples. Every time a KML record has been received from a client, the server looks up the `fsuid` and `time` fields of the record, and looks up the users key in the key database. Whenever the server fails a signature check, it simply rewrites the offending KML-records to no-ops.

The server does not need to sign changes to its own filesystem, as clients simply trust updates received from a trusted server.

For the GSSAPI security to have any effect, the server and client must deny to process any RPCs not arriving via an authenticated context. This prevents attackers from bypassing the initial `SysId` handshake, but does not guard against a trusted client exploiting potential security weaknesses in the file server RPC interface.

Furthermore three remote procedure calls (RPCs) were added to Lento in order to support the new security features.

ServiceAuth

When a client connects to a server, it starts out by sending a system id (`SysId`) RPC call to the server. This allows the server to update the list of active clients, and forces the client to re-integrate any changes.

When the server receives a `SysId` RPC from a client wishing to connect, it immediately sends back a `ServiceAuth` RPC to the client, with the result of an `init-sec-context` as parameter. Response of the `SysId` RPC is halted pending successful completion of the authentication. `ServiceAuth` does `accept-sec-context` on the token received from the server, and returns the result in the ensuing `REP`. The server then finishes the context establishment and wakes up the `SysId` session-handler to finish the connection process. The established context henceforth forms the basis of all communication between the server and the client, by being used to either encrypt or sign individual packets.

In order to establish mutual authentication between client and server, the server needs to have Kerberos credentials corresponding to the service principal `lento/servername@REALM`.

UserAuth

`Imlog` calls this RPC on the server with the result of a `init-sec-context` as the body of the call. The server does `accept-sec-context` and returns the result as well as an encrypted key used for signing KML entries. `Imlog` finishes context establishment, unwraps the key and

stores it in `/proc/sys/intermezzo/key`. `Imlog` then reads the timestamp for the key from the same file, and calls `SetTimeKey` on the server to set the timestamp.

The key is a reasonably random bit-string generated by reading a number of bytes from system random generator. The current implementation reads 128 bytes, but this may be easily adjusted.

SetTimeKey

After successful `UserAuth` the server expects to get a `SetTimeKey` RPC call, echoing back the key along with precise time at which the key went into use at the client. Both values are stored in the key storage.

Current limitations

The current implementation has a number of limitations.

- The `/proc/sys/intermezzo/key` interface only handles one key per user. Ideally it should be able to handle one key per user per server, since a client-Lento could access more than one server. It would be easy to extend the data-structure with an extra field in order to store the servers name.
- The keys cache in the Presto module is presently a double-linked list. It would be more efficient to implement this as a hash or tree structure, especially if the client serves a large number of users at the same time. This becomes even more of an issue when the interface allows multiple keys per user.
- Whenever the server fails a signature check, it simply rewrites the offending KML-records to `noops`. This holds a few drawbacks, since the KML parser in Lento currently begins back-fetches as soon as a `close` operation is detected. Since a valid close operation could occur earlier in the KML file than the invalid one, Lento would effectively begin a back-fetch of a file that is compromised. The entire KML file should be verified before any integration occurs, and any invalid records should result in a discard of the entire KML file, along with a warning to the operator that the client has been compromised.
- The implementation of the key storage relies on the users `uid`. Should an attacker obtain a root shell on a running system, the false root can still `su` to another user and have changes to the filesystem signed as that user. This can be alleviated by introducing PAG's (Braam [1998]). In our current implementation we decided not to use PAGs since they are not (yet) supported by the Linux kernel, but it would be trivial to use PAG numbers as subscripts rather than user ids, since we hold no semantic value to the indexes in the key storage.

- Signing the user's key with a key negotiated between server and client Lento, before returning it to imlog, and in turn Presto, would allow us to verify that the user was indeed issued by the server. An extra interface could be made in `/proc/sys/intermezzo`, where this key was placed, and Presto could make a signature check an incoming key, to see if it was a valid key. As it is, any user can write his key directly to Presto, and provoke a "Signature failed" error upon re-integration.
- The signature algorithm used is a simple MD5 digest of the user key and KML changes. Our knowledge of the MD5 algorithm, and its possible limitations when a large amount of clear-text is available to an attacker, is not comprehensive. It may be that the algorithm makes deriving user keys from signatures possible. This algorithm was chosen due to it not requiring the inclusion of reversible encryption code in the kernel module, which is impractical for political reasons. Changing to another algorithm should be simple.

CHAPTER IV

Findings

Some people drink from the fountain of knowledge, others just gargle.

ROBERT ANTHONY

Testing security

To assess the success of the implementation, a number of tests will have to be conducted. These tests fall into the main categories described below.

The tests only verify the correctness of our modifications, not the GSSAPI implementation, or other aspects of Intermezzo.

A table detailing all the tests, and their results, can be seen in appendix A.

User to service authentication

Using GSSAPI and Kerberos enabled tools, the user must be able to log into the system, and gain the expected access privileges, provided that correct credentials are presented. Conversely, the Intermezzo file server service must authenticate itself towards the user, using its own credentials.

Attempts to log in using false or missing credentials should result in an error message and denied access. Attempts to log into a server which does not authenticate itself should fail with a warning message.

Service to service authentication

Client workstations and the the Intermezzo file server service should authenticate mutually, presenting their correct service credentials.

In case either part fails to present correct credentials, the other side should halt the connection process, refusing to proceed any further.

Access to data

On client workstations, the file system should deny insertion of changes to users whose key is not stored in the kernel.

Upon re-integration of changes from a client workstation, the Intermezzo file server service should deny the replay of changes whose signatures cannot be verified by looking up the key for the (client-hostname, user-id, timestamp) tuple in the server's key database, and calculating the signature. Furthermore, an error message should be logged at the server.

Normal file system access controls are assumed enforced by the underlying file system, and will not be tested here.

Remote procedure calls

Only a the smallest possible number of remote procedure calls between client and server systems should be accepted in any direction, without first having negotiated a mutually authenticated context.

The calls that should be accepted are those needed to start a new connection, and to log a user into the system. Thus, all calls except `Ping`, `SysId`, `ServiceAuth`, and `UserAuth` should result in an error, if not taking place within a mutually authenticated context.

Expiration of authenticated contexts

In Kerberos, session keys expire after a while, typically 24 hours. However, the keys used by our modified Intermezzo kernel module exists independently of Kerberos keys, and should not expire. The `imlog` program only runs for a few seconds, so here key expiration will not be an issue. However, the Intermezzo file server and clients may run for a long time, and should take care to re-authenticate at latest when their common context expires.

This is a feature we would need to implement for the system to become fully usable, but for which we have not found time.

Benchmarks

Rather than benchmarking Intermezzo, we want to measure the impact on the system by our additions.

Specifically we investigate how the MD5 signing performed in the kernel, the signed or encrypted communication between client and server, impact performance.

File system performance with/without signing

We ran a benchmark called `bonnie++` (Coker [2001]), and measured for meta-data operations. Throughput on the filesystem is less important since we only sign meta-data.

The benchmark was run 5 times for each setup, creating and then deleting 8192 files. The best and worst times have been disregarded, and the remaining three were averaged.

The tests were run on a client-Lento in disconnected mode, since we wanted to measure raw filesystem performance.

We ran the test in two modes, one where each meta-data operation was signed with the MD5 signature, and one without. The results are tabulated in table IV.1.

		Sequential Order	Random order	
Test	Create	Delete	Create	Delete
With signing	233	1782	244	141
Without signing	230	1860	259	118
Difference	1%	-5%	-6%	19%

Table IV.1: Benchmarking filesystem performance. All values are measured in operations per second.

Conclusion on implementation tests

Predictably the signing incurred a performance hit, but our measurements show that it is in the order of 5-10% on meta-data-only operations. When averaged across operations reading or writing file data, we should see no performance penalty at all.

These results is important because Intermezzo was originally designed to exhibit local filesystem performance when operating on cached files (Braam and Nelson [1999]), so the added meta-data-signing should not prevent this.

Network performance in secure contexts

The Lento client and server establishes a context in which to communicate securely. GSSAPI offers two modes of security; signing and encryption of packages.

We wanted to test the impact on network performance resulting from securing Intermezzo file transfers when either signing or encrypting all traffic.

The time taken for a client Lento to re-integrating changes to a 40MB file back to the server was measured. The test was run both with GSSAPI encryption, GSSAPI signing, as clear-text and, for scale, simply copied with `scp`, the file transfer tool from Secure Shell (SSH).

Results are in table IV.2.

Benchmark conclusion

It is clear that encrypting or signing every packet between the server and a client, imposes a performance penalty, in the range 60% to 80%.

Why even clear-text Intermezzo is so much slower than `scp`, is not yet clear to us, though as can be seen the choice of encryption algorithm matters quite a lot. It may also have to do with the Perl implementation of Lento, or with the relatively small buffers used when reading or writing to and from disk and sockets. We expect this to improve in the future, when the Intermezzo implementation is rewritten in a non-interpreted language.

Conclusion Future work

Though long underway, Kerberos is gaining momentum as a centralised authentication service. With Microsoft touting it as the system powering its Passport service, and its competitors joined in the Liberty Alliance likely to adopt it as well, it finally seems to go into actual use.

Thin distributed file systems such as Intermezzo show great promise, due to their ease of implementation and good performance on cached data.

The integration of the above two systems makes using Intermezzo over untrusted networks easier, and more manageable, while not introducing fundamental changes to its network protocol or operation.

Intermezzo still lacks some features, such as data and demand, and better cache manager performance, needed to make it the ideal distributed file system for Linux, but it is rapidly progressing, and we are looking forward to the day when it may replace `scp`, `rsync`, `unison`, and other tools used for manually synchronising data today.

As Intermezzo is headed towards tight integration into the kernel (Braam [2001]), and towards using HTTP as the file transfer protocol, some of the points considered in this paper become moot. The main points about security models, and prevention schemes will still need to be considered though.

The user authentication and KML signing code should be applicable with little or no change, when Lento is replaced by Intersync, the next generation cache manager.

Test	Encrypted (3DES)	Signed	Clear	scp (3DES)	scp (Blowfish)
Total time	6:21	5:32	3:21	1:45	0:29

Table IV.2: Benchmarking network performance. All values are measured in seconds.

APPENDIX A

Test results

Test	Expected result	Result as expected
Successful kinit as correct user, followed by imlog	Welcome message, write access	Yes
Successful kinit as other user followed by imlog	Error message, no write access	Yes
Unsuccessful kinit as other user followed by imlog	Error message, no write access	Yes
Imlog into server without credentials	Error message, warning, no write access	Yes
Both client and server possess valid credentials	System running	Yes
Only client possesses valid credentials	Server running, client connection refused	Yes
Only server possesses valid credentials	Server running, client refusing to connect	Yes
Write to file system with valid key in kernel	Write successful at both client and server	Yes
Write to file system with fake key in kernel	Write successful at client, fails at server	Yes
Write to file system without any key in kernel	Permission denied at client	Yes
Client to server connection surviving expiration of context	No error	N/A

Bibliography

- Barak, A. and O. La'adan. The MOSIX multicomputer operating system for high performance cluster computing, 1998.
- Braam, Peter J. Process authentication groups (pags).
<http://www.uwsg.iu.edu/hypermail/linux/kernel/9802.2/0287.html>, 1998.
- Braam, Peter J. Intermezzo: File synchronization with Intersync. Technical report, Cluster Filesystems Inc, 2001.
- Braam, Peter J and Phillip A Nelson. Removing bottlenecks in distributed filesystems: Coda & Intermezzo as examples. *Proceedings of the 5th Annual Linux Expo*, pages 131–139, 1999.
- Coker, Russell. bonnie++.
<http://www.coker.com.au/bonnie++/>, 2001.
- Globus, . The globus project.
<http://www.globus.org/>, 2000.
- Linn, J. Generic security service application program interface.
<http://www.ietf.org/rfc/rfc2743.txt>, 2000.
- McClure, Steve and Richard Wheeler. MOSIX: How linux clusters solve real world problems, 2000.
- Miller, S. P., B. C. Neuman, J. I. Schiller and J. H. Saltzer. Kerberos authentication and authorization system. Technical report, Massachusetts Institute of Technology, 1987.
- Satyanarayanan, M. Integrating security in a large distributed system. *ACM Transactions on Computer Systems*, 07(3):247–280, 1989.
- Satyanarayanan, M. Scalable, secure, and highly available distributed file access. *IEEE Computer*, 23(5), 1990.