Introducing Rxk5

Marcus Watts
Primary author of Rxk5 and K5ssl design
and
implementation

Matt Benjamin responsible for most of
integration of
Rxk5 with OpenAFS

What is OpenAFS?

Andrew File System
Distributed Filesystem with Unique Properties

Open Source Descendant of Transarc AFS

What is OpenAFS?

i

Strengths of AFS

Volume Management
Scalability
Security (ACLs, Encryption)
Client Cache
Replication

Some Limitations of AFS

Sync-On-Close
Whole-File Locking
Kerberos IV
Legacy Encryption
Complex Administration
Windows Issues

Some Protocol Details

Rx Relative of ONC RPC (XDR)
Support for Streaming
Extensible RX Security Class
Layered on UDP
Ubik builds on Rx, adds High Availability

Some Server Details

Quorate Clustered Database Server
(Protection Server, Vlserver)
Arbitrary Number of Network Fileservers
(Fileserver/Volserver)
Vlserver Implements Transparent
Fileserver Redirection

Volume Concept

Filesevers Host Volumes (Virtual Disk)
Volumes Have Backup Clones, Replica Clones

Backup Volumes (Presently) May Be Accessed With RW Replica On-Line

Cache Manager

Kernel-Mode Filesystem on Unix*
Cache Manager Implements Block Cache
File Integrity Protected Via Callback
(Ancestor of Oplock)
Read Only Volumes Use 1 Callback
Etc

OpenAFS Developments 1.4.0/1

Pthreads (tviced, tbutc)
Vastly-Improved Windows Client
Client for MacOS X
Roaming Client Detection
LFS
Linux 2.6
Many Bugfixes/Improvements

AFS Developments Planned 1.6 and Later

Windows IFS Driver

(EricJW@umich.edu)

Enhanced Encryption (Rxgk, Rxk5 [us])

Database Server Improvements

Rxtcp

AFS Developments We'd Like To See

Server-Side Byte-Range Locking
Arbitrary Volume Snapshots*
Enhanced Replication
Journaled Fileserver
New Management Tools (Check Back)

What Is Rxk5?

Native Kerberos 5 security implementation for AFS (and related applications that use Rx RPC protocol)

Rxk5 Design Philosophy

More fully integrate Kerberos 5 into Rx and OpenAFS, providing strong encryption, up-to-date protocol design, in least amount of code

History

Of Rx...

- AFS
  - 3.0: bcrypt, rxkad
  - 3.1: rxkad
- DCE was AFS 4, not Rx

AFS with Kerberos 5

- IBM (idea)
- Rxkad2b
- Rxgk (GSSAPI, not finished)
- Rxk5

History

Of Rxk5...

- kth
- uniqname

An outgrowth of collaboration between various people, heavily on Marcus' work with AFS and Kerberos, Uniqname infrastructure at University of Michigan

History

Of Rxk5...

In 2004, we decided to collaborate on development projects with OpenAFS

Originally thought we would follow the CITI idea to use RPCSEC_GSS, like NFSv4, Changed our minds along with everyone else

Requirements

- Improve Encryption
- Suitable for use on small, standalone servers
- Small code size
- Must work with un-modified OpenAFS libraries

Improve Encryption

- Limit use of Kerberos session key
- Strong protection against replays
- Time/use limit on keys, no one key used too long
- Enctype and Level negotiation

Packet Flow

Pretty much what any authenticating Rx service does:

1. Server advertises service, creates UDP port, calls rx_NewService, passes in SecurityObjects for each supported security mechanism (index)

Packet Flow

2. Client decides to establish connection to server, decides which security mechanism it will use, and creates a Security Object for it (associating security object w/connection est. master secret for it)

(no data has passed)

Packet Flow

3. Client starts an RPC, waits for results, flushes data--this sends 1 data packet to the server, encrypted under a per-packet key derived from the master secret

Packet Flow

4. Server receives data packet, discovers it has no connection, creates a connection structure, and decides that the connection is not yet authenticated, saves the data packet, and sends a challenge

Packet Flow

5. Client receives challenge, and sends back a response, containing k5 ticket w/session key, along with information to derive the master secret

Packet Flow

6. Server receives the response, uses it to authenticate the connection, decrypts the saved data packet, does its processing, and sends back results, encrypted

Rxk5 Specifics

Challenge/Response could not just use Kerberos 5 authenticator, instead uses one based on Rx particulars:

* CID
* Epoch
* Security Index
* Service ID

Challenge uses random nonce (replay protection).

Per-Packet Keys

Derived from a per-connection master key, which is randomly generated by the client

Challenge response protocol is used to transfer this key to the server, which uses it to generate identical per-packet keys

Re-Keying

Not done, premise is that per-packet keys gives most of the advantages, with less complexity...

Key Derivation

Straight Kerberos derivation is used generate the client and serverdata keys. To encrypt a packet, another stage of derivation is used, based on MD5 hash of the packet header fields that can be reliably extracted from the Rx security object:

* Epoch
* CID
* Call Number
* Sequence
* Security Index

Consequences

Each data packet encrypted with a unique key. If you can break MD5, you might be able to get that side's data key, and you can reverse Kerberos 5 key derivation, you can get the master key for the connection, which does not expose the Kerberos 5 session key, which is only used in the challenge/response

AFS Integration

a. General Strategy

User Mode
Use local preferred Kerberos 5 implementation, already integrated into
OpenAFS 1.4. Heimdal pre-.8 snapshots, MIT 1.5, and standalone (using K5SSL in progress)

Kernel Mode

Only performs authentication on client-side connections, needs K5 encryption and basic data structures

Portability requires portable encryption and support libraries. We use K5SSL, an implementation of (mostly) just those parts of Kerberos 5 needed for Rxk5

K5SSL

Partial Kerberos V Implementation With Interesting Properties

Fast Portable Crypto Implementations (AES, DES3, Arcfour, ++ :)

Stripped Down (Doesn't Parse Keytabs, Implement a KDC, etc)

Emulates OpenSSL EVP Interface in User and Kernel Mode

K5 Principal Name Mapping

Defined at OpenAFS Hackathon 2006

Uses afs-k5/<cellname>@REALM as the service principal for Rxk5 connections.

This principal also used for inter-server communications, programs on AFS server hosts look for the key of thisprincipal in <afssysconfdir>/server/afs.keytab.

Code exists to forge
K5 service tickets (-k5 -localauth).

Pioctl/Token Interface

Implements 3 new pioctls*:

PGetK5Tokens
PSetK5Tokens
PGetCapabilities

Capabilities interface intended to permit Cache Manager to answerqueries about features it supports, in this case, supported or preferred Kerberos 5 encryption types

(Moving to CGet/Set*)

Platform Support

* i386, x86_64, UM Linux (client, server, cache manager)
* Solaris 8 (builds, some tools work)
* AIX 4.3 (builds, some tools work)
* i386 OpenBSD 3.9 (builds)

Left to do:

* MacOS X
* Windows (being worked on)

What Happens Next

Rxk5 is ported to current OpenAFS 1.5(.8) release candidate

Next Steps:

* Make OpenAFS build with Rxk5 on more platforms
* Finalize command line details
* Cosmetics
* Wider Testing
* Incorporation in OpenAFS CVS, hopefully within 1.5 unstable release cycle (looks on track)
* General availability in the next even-numbered release