Open Software Foundation **M. Burati (HP)**
Request For Comments: 93.0
February 1996

DCE 1.2 SECURITY SCALABILITY AND PERFORMANCE — FUNCTIONAL SPECIFICATION

1. INTRODUCTION

For DCE 1.2, HP has agreed to provide 6 engineering months (EM) of effort toward improving the scalability and performance of the DCE Security component. That effort includes looking into a subset of the changes listed below to determine what was possible to complete within the given time frame, and what should be looked at in a future release. The 6EM agreed to includes the time devoted to investigate each of these areas and others relating to customer feedback, the writing of this functional spec, and the coding, building and testing of any fixes made as a result of this work.

The main driving force for providing scalability enhancements to DCE 1.2 was to begin to remove the barriers to DCE acceptance that currently exist for customers who would like to migrate large (on the order of many tens of thousands, up to a million) user bases to DCE. Given the limited timeframe that DCE 1.2 must be completed and shipped in, and given the large number of other projects that will be a more visible improvement to the DCE functionality base, we needed to weed out those items that will not provide the scalability improvements that we really need to address, and concentrate on the issues that are currently barriers to acceptance.

1.1. Changes Since Last Publication

Relative to previous drafts, this document now includes further breakdown of the size and components of each piece of work, and further distinction of what is scheduled to appear in the 1.2 code base.

2. TARGET

The target audience of this technology is the same as that of DCE Security in general, although those that are attempting to deploy large cells of machines and/or users will benefit the most from these changes and from future resolution of the remaining issues.

3. GOALS AND NON-GOALS

> *NOTE: Nothing in this document should be construed as a commitment to provide any of the described functionality unless otherwise stated. This specification exists solely for the purposes of documenting the investigative work completed to determine the scalability needs of the DCE security component, and to document the improvements that we are able to implement during the DCE 1.2 timeframe.*

The *main goal* of this effort is to improve the performance and/or scalability limits of DCE Security, without affecting the publicly visible interfaces in any way. No change that threatens the security of DCE will be considered for implementation.

It is a goal to attempt to evaluate the usefulness/risk/work-required of each of the following areas (the results of these evaluations will be described below):

(a) Fine grain checkpointing.

(b) Replace back-end security database.

(c) Efficient ACL storage.

(d) Client binding rewrite

(e) More efficient memory usage.

It is *not* a goal of this effort to address any of the following previously suggested scalability topics:

(a) Batch propagations.

(b) Batch wire RGY interface.

(c) ERA triggers and search capability.

(d) User credentials carrying machine attributes.

(e) Non-RPC interface to `secd`.

All of these areas are beyond the scope of this limited project. We make no attempt to evaluate the importance or usefulness of any of them, and expect that they may be raised as future issues post DCE 1.2.

4. TERMINOLOGY

N/A.

5. REQUIREMENTS

There should be no publicly visible functional changes as a result of this project.

6. FUNCTIONAL DESCRIPTIONS

The following descriptions are given solely to provide direction to those investigating each area. The actual changes required for each selected area will be documented in implementation specifications as those projects are staffed and funded. Projects that are expected to appear in DCE 1.2 are given first, then longer-term projects are discussed.

6.1. Short Term Projects (Committed for DCE 1.2)

6.1.1. Memory leak cleanup

Because server size is one of the more limiting factors of security scalability, we undertook an effort to fix all DCE 1.1 memory leaks that we could find with a well-known memory analyzer. This was a multi-week effort.

6.1.2. Better cleanup of database memory

The security server backend database had legacy code (from a now obsolete operating system with poor malloc/free performance) that attempted to reuse deleted database entries, instead of freeing the memory. This (mis)feature results in the loss of large amounts of memory over the lifetime of a long-running security server with a large registry database. Given that malloc/free perform better on most platforms today, and noting that the reuse of memory has been inadequate in the current implementation because of the type of keys used to index the data storage, this was deemed worth fixing in the 1.2 timeframe.

The scope of work for this effort included determining the algorithm for deletion of nodes from a 2-3 Balanced Tree (it was left as an exercise to the reader in the reference text used for the rest of the algorithms). Before we got around to start implementing it from scratch, it was found that this same database code had been used for the `glbd` (Global Location Broker Daemon) submitted by HP/Apollo as part of DCE 1.0, and that the owner of that component had worked out the algorithm and included it in the reference source. The rest of the work included a rewrite of some of the code to work with the data structures in use by the security server database, a code review/inspection and a test effort.

6.1.3. Security salvage tool rewrite

For a long time, it had been known that the salvager tool (`sec_salvage_db`) used up an inordinate amount of memory, for even relative small (100's of accounts) registry databases. During OSF tests, it was determined that it was impossible to salvage a database of more than 10,000 accounts with any of the available test machines. Work here verified that even with 100's of megabytes of swap, the way that the salvager reconstructed the database in memory prevented us from salvaging a database that large. Given that many DCE customers may start to work with large databases in the near future, it was deemed necessary to fix this for DCE 1.2.

A combination of code inspection and redesign, along with running memory leakage analyzers over the tool, helped us come up with a new salvager that works on large databases (many thousands of accounts) even on minimally configured DCE server machines (e.g., 64MB memory, 128MB swap).

6.1.4. Security client binding improvement

As cells become much larger, and the number of servers used to support those cells becomes larger, it also becomes painfully slow to wait for rebinding to occur. The default RPC timeout used for calls to the DCE registry in all previous releases has been 30 seconds. It has been determined that this timeout is too long, and that many users feel that the system is down if there is no response in a matter of seconds. See the following list for details.

(a) *Rebind performance.*

Changes were made to the binding code submitted to DCE 1.2.1, to rebind on failure once for each server in the list, using 4 second timeouts. If there is no response from any of the servers, using a 4 second timeout, the binding code will automatically retry each server again with a 30 second timeout. The maximum number of 30 second tries will be 5, as with previous releases. We found with trial uses here and at customer sites, that this greatly improved the response time of DCE as a whole, in a busy/changing network environment.

(b) *Update rebind.*

Another improvement was to allow update bindings to rebind on error. Prior to DCE 1.2.1, update bindings would not rebind on error (because there could only be one master). In an environment where the master server might be temporarily unavailable (e.g., 60 seconds) or the replica that you've contacted to find the master might be down, this caused annoying errors on client machines.

(c) *Rebind on bad state.*

Another problem that could occur in a large cell with many replicas is that you could attempt an operation to a security replica that was being reinitialized for some reason. In prior releases, this would just return a ''Bad State'' error. As of DCE 1.2.1, it now attempts to rebind to another server after receiving this error.

(d) *Stale binding cache.*

There has always been a problem administering multiple security servers with tools like `sec_admin`. For example:

(i) Stop a security server (`secd`).

(ii) Attempt to bind to any `secd` in the cell for the next command.

The bind in step (ii) could end up retrieving the binding obtained in step (i) from the binding cache, even though the binding is no longer useful.

To fix this problem with the binding cache, DCE 1.2.1 now ensures that the `requested_site_name` in a registry binding lookup matches the `requested_site_name` on a cache entry, before returning it. With this additional check, the binding to a specific server obtained in step (ii) can no longer be returned from the request for a binding to ''any `secd` in the cell'' in step (i).

6.1.5. Configuration file management

Part of the problem with managing any large cell of DCE client machines, is the management overhead involved with the many copies of configuration files.

In a large cell environment, both the `pe_site` file (security server binding list) and the `krb.conf` file (Kerberos configuration file) can become out of date on every client, as security servers are added/removed/moved within a cell.

To solve this problem, we have added functionality to `dced`, to automatically update the information in these files on a regular basis.

6.1.6. GSSAPI performance

There was a modification request opened against GSSAPI after DCE 1.1 to notify us that the `gssapi.c` module's misuse of serviceability debugging statements was causing a fairly large performance hit. We investigated the request and made the necessary fixes (a little more work than the request described) to eliminate the performance hit.

6.1.7. General security performance work

Performance of clients plays a big part in scalability of DCE cells. If clients do not perform adequately to begin with, then any decrease in cell speed as a result of increased server size will be more likely to cause problems. Given that the perception of overall performance can be more important than absolute performance numbers, we went looking for things that appeared to be taking longer than they should:

(a) *Name translation performance.*

In DCE 1.1, there was a limitation in `sec_id_parse_*()` calls, where they would never get a cache hit if the input global name was cell relative. This was fixed, along with some other performance improvements to the code (e.g., make calls to registry binding from the `sec_id` layer more efficient).

(b) *CDS security context lookups.*

The CDS client code calls `sec_login_export_context()` twice for every usage (once with a 0 length, so it can determine the actual length needed). At first, this appeared to be unnecessary overhead, but it turns out to be necessary given the current CDS client implementation. The quick fix is to make the simple case of `sec_login_export_context()` (when called with a 0 length buffer) a fast path that does hardly any work. This fix improved CDS lookups by upwards of 10% in hand testing.

(c) *Ticket cache storage.*

New Kerberos tickets were always appended to the end of the credential file cache, allowing it to grow without bound. This limitation required overhead on every lookup to scan over the expired credentials. This caused fairly large credential cache files for users if they stayed logged in for days at a time, and for machines that use the same credential cache for as long as they remain up.

New tickets now overwrite any existing ticket for the same client and server as long as the tickets are the same size. If the existing ticket has not expired, then the new ticket overwrites the old one only if the `authdata` matches. Otherwise the new tickets are appended as previously.

(d) *Ticket cache lookups.*

Retrieval of a ticket from the credential file cache required a new lookup and other overhead each time an expired ticket was encountered, even if it was known that only unexpired tickets were desired. To remove this overhead, the logic for skipping over expired tickets has been moved down the code chain to a lower-level ticket retrieval routine.

6.1.8. Cryptographic performance

We made a small change to the implementation of the MD5 algorithm to save a few instructions per iteration, and we replaced the DES algorithm with a newer one that saves a few instructions per iteration. Hand testing of the algorithms themselves shows that these two changes gave us the following performance improvements: improves MD5 inner loop ~200%, DES inner loop ~50%. It's likely that these changes will speed up authenticated RPC by a few percent, though no performance testing is planned or required for 1.2.

6.1.9. utc_gettime() performance

`utc_gettime()` was slower than it needed to be. The changes to improve performance include using an exponential backoff for mapping the `shmid` on failure and more efficient lookups (at most every 10 seconds) to avoid the need to lock/unlock the shared memory segment as often.

With this fix, CDS speeds up ~10-20% if `dtsd` isn't running. The fix results in a small performance improvement (reduced syscalls) if `dtsd` is running. While this change isn't to security itself, it can speed up security binding indirectly.

6.1.10. NSI performance

NSI wasn't setting the CDS `MaybeMore` bit. This bit is a hint which is used when a client knows that it will be fetching more than one attribute on a given name. The fix to `nsutil.c` causes this bit to be set more often, leading to better cold CDS cache performance.

6.2. Other Potential Projects (Not Committed for DCE 1.2)

6.2.1. Fine grain checkpointing

The problem of security server availability during checkpoints has already been determined to exist in actual simulation tests run by the University of Michigan CITI program (see [Carter]).

This should be considered one of the the highest priority scalability items listed here. Unfortunately, the amount of work necessary to scope, specify, implement, build and test finer grain database use cannot be adequately staffed within a subset of the allotted 6 months.

This work would entail a good size effort at reorganizing the existing backend registry database code (all of which is organized around reading and writing entire database trees from/to disk at one time), and writing/testing database conversion code to migrate from existing security databases.

6.2.2. Configurable checkpoints

For the short term, DCE 1.2 does include the configurable checkpoint mechanism for the security server, where you can specify the interval or exact times that checkpoints should occur. This mechanism should allow administrators to minimize the affect that checkpointing has on security server availability.

6.2.3. More efficient ACL storage

Ever since DCE 1.0, we have intended to provide internal `secd` functionality that would allow ''sparse ACLs'' (reuse of existing ACLs via refcount and copy-on-write). We must consider this a high priority issue, given the enormous overhead that not fixing this earlier has caused with large registry databases.

It has been determined at Transarc that the ACL relation of the security database can use up to one-third of the size of the entire database. Given that most of the ACLs in the security database are redundant duplicates, we should be able to gain an enormous Virtual Memory, Disk Space, and Checkpoint Time savings by providing this functionality. Out of the larger, non-completed projects, I would consider this second in priority only to more efficient checkpointing within the scalability project.

A related and necessary subproject, not included as part of previous descriptions of this work, is providing a ''Salvage'' mechanism (via Checkpoints, or manual, or other means) that would go through the entire database and compress duplicate ACL entries back into a single refcounted ACL entry. This functionality will be necessary to reclaim the redundant space from existing security databases that are just migrating forward to the new software, and to periodically reclaim redundant space used when multiple ACLs are changed and end up being the same in the end but occupying separate storage.

6.2.4. Client binding rewrite

It has been determined that the overhead that a client endures in order to bind to the registry is undesirable. The current client binding code has been evolving through many years of DCE changes and now includes inefficiencies based on changes that were the right thing to do at the time, to fix a problem or add new functionality.

The fact that all DCE clients on a particular machine have to go through a separate CDS clerk to contact the CDS database to get identical information seems rather inefficient, even wasteful. We believe that we should be able to improve performance significantly by having a central daemon (say, `dced`) per machine keep track of the current bindings to the registry. An additional improvement should be possible by keeping track of which replicas are currently available. This would be a very large win for DCE acceptability in large cells, but not quite as important as the checkpointing, and maybe only as important as the sparse ACL work.

There is no way that this work could fit into a subset of the allotted six months, but smaller fixes (listed above) were made as a result of initial investigation into the problems. The more general problem of binding by ''server affinity'' was determined to be a general problem with DCE itself and should be addressed for all RPC binding lookups, not just security.

6.2.5. Replace back end database for security server

Initial work will entail estimating how long it would take to do the actual replacement. Ongoing work during the larger project's ''initial scoping'' will include investigating whether there are any existing databases that we can use in the timeframe given to this project.

6.2.6. Propagation model changes

It takes a large number of RPCs and a lot of server overhead, to propagate changes to replicas in a rapidly changing large-scale environment. In addition, it takes too long (hours) to initialize a replica from a single master security server cell, with hundreds of thousands of accounts.

Although the investigation of the solution to this problem still needs to be done at some point, we can attest to the magnitude of the problem at this time. It is not something that is solvable by merely speeding up a few operations. We really need a bulk initialization protocol for security replicas.

7. DATA STRUCTURES

N/A.

8. USER INTERFACES

N/A.

9. API'S

N/A.

10. REMOTE INTERFACES

N/A.

11. MANAGEMENT INTERFACES

N/A.

12. RESTRICTIONS AND LIMITATIONS

The work submitted to DCE 1.2 will be limited to that which does not adversely affect the quality of the system as a whole, and to that which does not affect compatibility in any way.

13. OTHER COMPONENT DEPENDENCIES

N/A.

14. COMPATIBILITY

N/A.

15. STANDARDS

N/A.

16. OPEN ISSUES

One of the largest problems we have with addressing scalability and performance issues is the lack of a good, generic scalability and performance test. It's one thing to say that you've made a fix to DES to speed up the inner loop by X%, and yet another to find that it increases a fixed size single RPC by Y%, but that doesn't tell us what we're doing for DCE as a whole. Customers need to see response time improvements, and while RPC performance tests will let us know if we're making RPCs any faster, we need to know what's making DCE in general faster or slower as changes to the source base are made (both for verifying supposed performance/scalability fixes and for verifying that we aren't regressing during a release development cycle).

IBM has offered to DCE 1.2, and is expected to deliver to the ‘‘nosupport tree’’, a generic scalability test that they have developed to measure DCE’s ability to scale, using a simulated customer application. A test like this is vital to the DCE performance effort and should be utilized where possible to measure DCE scalability during (at least before and after) release development cycles to see what progress (or not) is being made.

REFERENCES

[Carter] M. Carter, ‘‘Adding 50,200 Users to a DCE Registry.’’, February 1994.[1]

AUTHORS’ ADDRESSES

Michael Burati
Hewlett Packard
300 Apollo Drive
Chelmsford, MA 01824
USA

Internet email: burati@apollo.hp.com
Telephone: +1-508-436-4351

1. gopher://gopher.citi.umich.edu/11/public/techreports/ASCII/citi-tr-94-1.ascii or gopher://gopher.citi.umich.edu/11/public/techreports/PS.Z/citi-tr-94-1.ps.