Distributed GSM - Feature #4306

GSUP proxy cache: store data persistently

12/04/2019 01:58 PM - neels

Status:	Stalled	**Start date:**	12/05/2019
Priority:	Low	**Due date:**	12/05/2019
Assignee:		**% Done:**	0%
Category:			
Target version:			
Spec Reference:			

Description

when we store auth tuples (#4305), even a local power outage should retain those tuples in the proxy.

options:
- store in the HLR DB
- just dump to a local .csv file to recover on startup

Related issues:

Follows Distributed GSM - Feature #4305: GSUP proxy: cache auth tuples to re-...	**Stalled**	**12/04/2019**

History

#1 - 12/04/2019 04:09 PM - neels

#2 - 12/10/2019 04:04 PM - neels

Thoughts about data persistence

What data is stored in the proxy?

- IMSI (key)
- MSISDN (for mslookup)
- last CS LU (for mslookup)
- last CS LU VLR's IPA name (which MSC is the phone attached at)
- auth tuples
- auth tuple state? (which ones used how often?)
- LU "faked" locally or solicited by home HLR?

Auth tuples: see #4305

What do we need to persist?

The scenarios are:
(A) home HLR unreachable
(B) proxy HLR restarted
(C) both

(A) home HLR unreachable.

- A subscriber is currently attached at a proxy HLR, and the link to the home HLR goes down or unresponsive.
- We've asked for auth tuples before. For incoming SendAuthInfo-Request from a local VLR (MSC or SGSN), we use the cached auth tuples and don't even attempt asking the home HLR.
- Once the VLR has authenticated, it will follow up with an UpdateLocation-Request. We attempt to forward, but the home HLR doesn't respond (timeout?) or the GSUP client is unable to connect.
- We send an UpdateLocation-Result (and flag in the proxy data that we have "faked" a LU without the home HLR knowing? may even be unnecessary).

Maybe it would make sense to actually never forward UpdateLocation to the home HLR, and always keep that entirely in the proxy HLR?
Except, when the home HLR has disabled the subscriber (nam_cs/nam_ps), the proxy HLR won't know about that.
IIUC the SendAuthInfo-Request is serviced regardless of nam_cs/nam_ps, so the proxy would still get tuples
and continue to accept LU, even though the home HLR has disabled the subscriber.
Also, if the MSISDN or anything else changes, the proxy needs to obtain an update as soon as possible.
So, it does make sense to at least attempt to contact the home HLR for each periodic LU.

(B) proxy restarts

- A subscriber is currently attached at a proxy HLR, and the proxy HLR experiences power outage / program restart.
- After the outage, the subscriber does a Complete-Layer-3 with auth+ciph
- We still have usable tuples in the persistent cache, we use those and don't even do SendAuthInfo-Request to the home HLR.
- Upon UpdateLocation-Request, we reach the home HLR and continue as usual.

(C) proxy restarts + home HLR unreachable

- A subscriber is currently attached at a proxy HLR, and the proxy HLR experiences power outage / program restart.
- After the outage, the subscriber does a Complete-Layer-3 with auth+ciph
- We still have usable tuples in the persistent cache, we use those and don't even do SendAuthInfo-Request to the home HLR.
- Upon UpdateLocation-Request, we **cannot** reach the home HLR and continue as in (A)

#3 - 12/10/2019 05:09 PM - neels

thoughts about persistence technology

Which is easier, storing in a CSV file or in the SQLite database?

SQLite

- we already have SQLite in osmo-hlr
- add separate table for proxy cache (key=IMSI); another table for N auth tuples per IMSI
- future: proxy cache structural changes will require DB upgrade path
- store:
  - UPDATE last cache entry for IMSI
  - INSERT INTO to add auth tuples
  - garbage collection in auth tuples table: DELETE from db table after N re-uses
  - DB writes for every action
- read:
  - select proxy cache entry for IMSI
  - select N auth tuple results, possible to select by previous usage count, etc.
  - DB reads for every action -- could also keep in RAM and only write to SQLite, and read only if a new IMSI shows up
- scale: scalability handled by SQLite -- we can choose to keep all or "no" records in RAM

CSV:

- need to manage another file location (new config item / default to current dir)
- store:
  - only append to CSV file.
  - write last cache state, one per line
  - write N tuples in separate lines? (I)
  - write all N remaining tuples with usage count in one line along with cache state? (CSV within a CSV column, one with tab sep and one with comma?) (II)
  - filesys write for every action
  - no protection against corrupted writes
- read:
  - read only on program startup.
  - read and parse all previous lines and keep only the last cache entry item per IMSI.
  - guard against parse errors; mangle strings and so forth
  - in running program, no filesys reads, only RAM state used
- garbage collection:
  - Remove the file manually?
  - regularly keep only unique IMSI entries by cron? make sure to not write at the same time as osmo-hlr attempts to append an entry.
  - On program startup, osmo-hlr could read all entries, keep only the last, and then overwrite the file.
- scale: always require all proxy data in RAM (also entries inactive for a long time?)

(I) N tuples in separate lines

```
IMSI   MSISDN   last-LU ...  USED RAND SRES KC IK CK AUTN RES
123456 123      2019-12-08
123456                       2 abcd123 abc123 abc123...
123456                       0 def987 def987 def987...
987654 987      2019-12-09
987654                       ...
123456 123      2019-12-10
123456                       3 abcd123 abc123 abc123...
```

drawback: need to combine lines / detect identical auth tuples
advantage: less data written on each action, easier to read for humans

(II) all in one line

```
IMSI   MSISDN   last-LU ...  TUPLES
123456 123      2019-12-08   2,abcd123,abc123,abc123...;0,def987,def987,def987...;...
987654 987      2019-12-09   ...
123456 123      2019-12-10   3,abcd123,abc123,abc123...;0,def987,def987,def987...;...
```

drawback: huge line width (think 20+ tuples); "nested CSV" parsing; more I/O for each action (a lot more data written than actually changes)
advantage: a single final line determines entire IMSI state

conclusion

CSV	SQLite
pros	**pros**
no file reads except on startup	already have SQLite tools
	better scalability
	DB version upgrade path allows keeping proxy entries across osmo-hlr upgrades
cons	**cons**
need to re-invent: string parsing, validation...	DB reads on each operation (but could also read only on program startup?)
need to guard against partial writes?	DB version upgrade needed for changes to proxy data model
large-ish filesys writes for each operation	
configure/manage another file location	

It is hard to tell which solution would benchmark as more efficient.

At first glance, it seems simpler to not hit the database all the time; but we do that for normal HLR operation anyway.

I would personally favor writing a SQL DB API for a proxy cache instead of adding CSV parsing and string mangling to osmo-hlr.
Especially since we already have all the SQLite tooling in osmo-hlr, it seems odd to not use that for an obviously similar dataset.
Storing and reading from SQLite is already solved and known, CSV parsing would be a new wheel invented with possible surprises / debugging needed on a basic level.

As a gut feel, we should benefit from the SQLite implementation in terms of protection against interrupted writes
and similar basic problems we might have to re-invent protection against for CSV.

My choice would be to go for SQLite.

#4 - 12/10/2019 08:50 PM - laforge

On Tue, Dec 10, 2019 at 04:04:26PM +0000, neels [REDMINE] wrote:

> So, it does make sense to at least attempt to contact the home HLR for each periodic LU.

I would also state it makes sense. Not doing so would break fundamentally how GSM traditionally
works, and people might be placing external/additional logic around the assumption that every
LU (at least during available back-haul) will be seen at the HLR.

#5 - 12/10/2019 08:59 PM - laforge

Hi,

On Tue, Dec 10, 2019 at 05:09:51PM +0000, neels [REDMINE] wrote:

> As a gut feel, we should benefit from the SQLite implementation in terms of protection against interrupted writes
> and similar basic problems we might have to re-invent protection against for CSV.
>
> My choice would be to go for SQLite.

In theory that's correct, but I'm a bit worried that we might not be
using SQLite properly. Whatever method is chosen, we should try to think
of how we can simulate/test those scenarios that we're trying to protect
against. Just assuming that a third party tool will do a good job at X
without any hard evidence or proof has often turned into a big
disappointment later.

What I liked about the CSV (or any kind of "log file", whether binary or
textual) approach is the simplicity and ultra-low comlexity when writing

the data. I had assumed we don't ever do any updates to the data, but simply always append the most recent complete record. A database will put significantly more complexity (and hence latency) as it will update existing records.

I'm not saying I'm against SQLite, I'm just saying for "append only" (and possibly "logrotate" on a external signal) use cases, I would presume that writing to a file would be low overhead and low complexity.

At least to me, that is the use case here: append-only, no updates, and read-only at startup.

One could also simply append each record in binary as some "C struct" with some kind of checksum (CRC32) at the end. On restart, we read all records where the CRC matches, and we skip/drop/stop once it doesn't match.

Just some thoughts. Your call, as you'll do the actual implementation.

#6 - 12/12/2019 09:26 PM - neels

On Tue, Dec 10, 2019 at 08:59:33PM +0000, laforge [REDMINE] wrote:

> In theory that's correct, but I'm a bit worried that we might not be using SQLite properly.

I remember the Subversion lib_wc needing extra file locks to ensure no concurrent writes...

> What I liked about the CSV (or any kind of "log file", whether binary or textual) approach is the simplicity and ultra-low comlexity when writing

That also appeals to me very much, indeed.

But it seems to me like it sounds very simple and desirable, but has tails attached that end up being not so simple. If there were no hlr.db already I would probably not consider SQLite...

Anyway, I'll reflect on it again a bit.
First off I'll simply implement the behavior in RAM, and as a next step will handle persistence, so there's still a lot of time to reflect.

- I do expect that we would need to store how often a particular tuple is used. So we would dump the full record with all, what, 20 tuples on each use of any one tuple?
- String mangling (CSV escaping? error handling?)
- validation hash?
- logrotate
- when we always use the DB, we are sure that ongoing use acts identically after a restart (because it is used==tested all the time). If we read only on startup, bugs in there are only noticed after a restart, less likely that anyone would even notice that kind of error at all, if not paying close attention to SQN/SYNC logging...

Ends up a five-tailed fox behind the CSV rabbit

#7 - 04/28/2020 02:39 PM - neels

- Status changed from New to Stalled

Since the auth tuple cache #4305 itself is stalled, there is no use in making the HLR proxy cache any data persistently.

#8 - 04/28/2020 02:39 PM - neels

- Due date set to 12/05/2019

- Start date changed from 12/04/2019 to 12/05/2019

- Follows Feature #4305: GSUP proxy: cache auth tuples to re-use / fall back to simpler auth methods added

#9 - 05/20/2020 09:18 PM - neels

- Priority changed from High to Low