Replay Caching is a Pain

- Kerberos AP exchange requires replay caching
  - Replays → bad
- rcaches usually hard to get right, and ***slow***
  - Regular disks → 120 fsync()s per-disk/second
  - Clustering → ouch!
- Want to do something about this?
- Options:

  1) fast rcache designs

  2) rcache avoidance

  3) ??

Fast Replay Caching (1)

- Don't do synchronous writes at all
- Or, if you care about DoS attacks
  - Define 'cutoff' (say, 3 seconds)
  - Async write when Authenticator time-stamp < (now + cutoff), else sync
  - **And** reject when time-stamp < (boot_time + cutoff)
  - Memory speed most of the time w/ minimal outage on crash (none if all clients do kdc_timesync/NTP and time to boot < cutoff)
    - Assumption: time never goes backward!

Fast Replay Caching (2)

- Use an Adaptive Bloom filter or similar data structure for fast rcache check
  - Positive → check actual replay cache entries, **or**
  - Accept non-zero false positive rate and size filter for acceptable rate → rcache is just the filter
  - *Adaptive* Bloom filter → trivially self-expunging!
  - Filter-only → harder to do on disk, better to have IPC svc
- Or use memcached
- Expunge matters when rcache can handle hundreds of thousands of ops/s!

Replay Cache Avoidance (1) Implicit 3rd leg

- If all clients of a service principal must use per-message tokens and the client's first such token effectively acts as a “third leg” for 1 round-trip mechs, then rcache can be avoided.
  - E.g., NFS, SSHv2
  - If configured in the mech → no API changes needed
    - Problem: leap of faith that there won't be apps that only authenticate (think of kerberized r-cmd)
  - A req_flag could be used to solve leap-of-faith issue
  - rcache still needed for PROT_READY per-msg tokens
  - “3rd leg” must confirm something (e.g., acceptor sub-key)

Replay Cache Avoidance (2) Explicit 3rd leg

- Client could say “I can do an additional AP exchange leg” in its AP-REQ
  - MUST ensure that such AP-REQs are rejected in any other contexts
    - Use GSS flag or authz-data element for this; upgrade SW
  - Needed: critical way to signal in AP-REP whether the server accepts this and expects that additional leg
  - GSS-only? Or raw krb5 too? Easier if GSS-only!
- GSS details: new req_flag for initiator
  - New ret_flag not needed
  - If flag present → use an extended AP-REP PDU

Replay Cache Avoidance (2)
Other uses for extra legs extension

- We could use a negotiation of “extra round-trips OK” for other purposes
  - Key rollover! (get fresh Ticket, retry w/o re-connecting)
  - Negotiation of authz-data elements?
  - U2U TGT request using bogus AP-REQ with new APOptions flag?
    - Reply would be TGT
- Other uses for extended AP-REP
  - Allow svc to issue a replacement Ticket w/ a single authz-data element referencing cached elements on svc side
    - A way to deal with those huge PACs (and soon also huge PADs :)
- ...

Replay Caching for Clusters?

- Filesystem-based rcaches are no good for clusters – clustered filesystems are too slow
- New protocol?
- Memcached?
- Split brain → bad, must avoid
- *Much* easier (better?) to avoid need for rcache!

Consensus Questions

- Avoidance by implied 3rd leg only?
- Avoidance by negotiation of 3rd leg?
  - With a fast rcache it's still better to avoid that explicit 3rd leg!
    - But fast rcache for clusters is extra hard
- Fast rcache is really just implementation details – worth documenting some techniques in an FYI RFC?
  - And if we need a protocol for clusters, do we want a Standards-Track protocol? Or shall we assume homogeneous clusters? Distributed rcache service might be generally useful!
- I'm of two minds about all this myself... I prefer implicit 3rd leg (with req_flag) and fast rcaches, but explicit 3rd leg is simplest for clusters -Nico

Exporting Partially-Established GSS Security Contexts (1)

- Why?
  - Implementation choices
    - Useful for acceptor application implementations where process must or might restart between security context tokens
  - Protocol design issues
    - Stateless acceptor apps not susceptible to replay attacks (e.g., KDCs)
- Why not?
    - Nothing makes this fundamentally infeasible <rehash-arguments src='KITTEN WG mailing list'/>

Exporting Partially-Established GSS Security Contexts (2)

- Specification is trivial: just allow it
  - Possibly define a new major status code
- Might want a new utility function to output encrypted “state cookie” containing exported security context tokens
  - Should contain a time-stamp, validity period
  - Can't protect against replays...
  - ...but one could use an rcache to *minimize* acceptor-side state size and still get replay protection