Problems with the GSS-APIv2u1 C Bindings

- Where to start...
  - Memory management headaches
  - Insufficiency of minor_status for communicating complex errors
  - Extensibility + software layering issues
    - See further slides
- What to do?
  - Make do?
  - Fix?
  - GSS-API version 3?

Memory Management Hell

- gss_buffer_desc is part of the ABI, often allocate on the stack (automatic)
  - Has no way to record how to release the actual buffer
- gss_release_buffer(), in a mechglue environment – what to do?
- This problem is **urgent**
  - Likely solution: require all providers to use the same allocator as the mechglue
    - This requires good run-time linker fu, but we think we can make it work on all major OSes

Minor Status Issues

- Want to have detailed error messages about complex situations
- minor_status is an integer, not smaller than 32-bit, and, really, not larger than 64
- What to do?
  - So far MIT and Heimdal allocate minor_status values dynamically, mapping them to detailed info
    - Oops: when to release? When gss_display_status() is called?! LRU?

Extensibility/Layering Issues

- Layering: imagine sshd using GSS and PAM with a pam_ldap using libsasl using gss
  - GSS used in two “apps” in same process
  - Very common
  - “don't do that” is not a very satisfactory answer
    - Yes, some of this can be avoided by using IPC to add isolation
- Want GSS functions like...
  - Set mechglue config
  - Set mechanism config
  - Set policy (cipher suites allowed, …)

Extensibility/Layering Issues

- Imagine a gss_acquire_cred() extension that can handle initial cred acquisition (interaction)
  - New function to replace gss_acq._cred()
    vs.
  - GSS_S_INTERACT → call gss_get_prompt() and gss_set_answer(), say – where to stash the prompt/answer?
    - If we had a call context, then we could stash them there

Solutions?

- There's at least **four** solutions that some of us have considered
  - “PGSS-API”
    - Change minor_status arg in ABI backwards-compat way to a call context
      - Change the header file
      - Don't change the header file
    - Spec a GSS-API version 3
    - Run-time linker fun (fun!)
    - More types, more functions (gss cred opts, sec ctx opts, …)
      - we've been going down this path (think of gss_set_allowable_enctypes())

PGSS-API

- Variant 1:

  s/OM_uint32 *minor_status/gss_call_ctx_t call_ctx/

- Variant 2:

  gss_alloc_call_ctx() →

  outputs a call context object **and** an OM_uint32 * to be passed to gss functions

  gss_release_call_ctx() →

  releases the call context and associated OM_uint32

- Impact on mechglue/providers: must map input OM_uint32 * to call context (yes, it can be fast)

  Credit for initial PGSS use-case and solution: David Leonard @ Quest

GSS-APIv3

- An opportunity to fix all sorts of issues
- New function/type/constant prefix? gss3? Gs3?
- But will any apps be changed to use it?
- v2u1→v3 shim should be feasible and desirable
  - Only apps need this: providers should get converted along with mechglue since we don't have an open SPI at the moment

Run-Time Linker Fun! (1)

- Apps would dlopen()/LoadLibrary() **distinct** copies the mechglue, and the mechglue would load distinct copies of the providers
  - Literal file copies – dlopen() knows about links!
  - Or new dlopen() flag to request a distinct load
- GSS functions with global state would affect only the global state of the loaded object, of which there may be many "local" copies, therefore global state → local state!
  - Magical!
  - We think we can make this work on major OSes
  - GSS function pointer set is equivalent to caller context handle

Run-Time Linker Fun! (2)

- How to make this manageable?!
  - We could use a struct to hold gss fn *s
  - We could use a function from a “gss loader” library or else a macro to load these
- Something like this, perhaps:

```
gss_fn_set_v3u0_t gss = GSS_LOAD(); /* or */
gss_fn_set_v3u0_t gss = GSS_LOAD_OBJ(“/.../foo.so”);

gss->set_config(...);

gss->init_sec_context(...);
```

Run-Time Linker Fun! (3)

- GSS_LOAD*() macros might look like:

```
do { \
  void *hdl = dlopen(...); \
  gss_load_fn_t fn; \
  if (hdl != NULL && (fn = dlsym(hdl, …) != NULL) \
    return fn(hdl, …); \
while (0);
```

- Or use GCC statement-expressions :)

Run-Time Linker Fun! (4)

- But! Whereas objects from a PGSS or GSSv3 provider/mechglue might be useable with distinct call contexts, objects from a distinct provider/glue **cannot** be safely passed to others!
  - **Huge** gotcha?
    - Mostly not for layered software case, but it does mean that GSS objects can't be passed over non-GSS APIs (think of APIs for, say, RPCSEC_GSS).

Questions

- Sooner or later an implementor or three will badly want to do *something* about this
  - We should specify a Standards-Track solution
  - Which?!
- Nico dislikes the more-types-more-functions approach to addressing every sub-problem here
- Margaret dislikes PGSS variants
  - Sam dislikes PGSS variant with header changes, thinks C++ will be impacted (not clear yet)
- Nico loves rtld games, but... this one?! Not sure.

Questions?