Client-OSS Connection DLD

Zhao Qiang

Mar 07, 2005

1 Functional specification:

- What will modified functions do?
- What is their prototype?
- How are parameters affected? What is returned?
- You may use Z if you wish.

1.1 lmc

Lustre use the lmc to generate config.xml for mds/ost/client, need add following secure option for lmc:

1.1.1 secure option for client

These options define the secure flavor for connecting from client to mds/oss.

```
--mds_sec       /* client<=>mds */
--oss_sec       /* client<=>oss */
```

1.1.2 secure option for mds

These options define the secure flavor for connecting from mds to mds/oss.

```
--inter_mds_sec /* mds<=>mds */
--mds_oss_sec   /* mds<=>oss */
```

This option define which secure flavor from remote to this mds is **denied**.

```
--deny_sec      /* mds<=[client|mds] */
```

1.1.3 config for oss

This option define which secure flavor from remote to this oss is denied.

```
--deny_sec      /* oss<=[client|mds] */
```

1.1.4 config xml sample:

Following is security section of sample config file generated by lmc:

```
<node name="client" ...>
  <sec>
    <mds_sec>krb5i</mds_sec>
    <oss_sec>krb5p</oss_sec>
  </sec>
</node>
<mds ...>
  <sec>
    <inter_mds_sec>krb5i</inter_mds_sec>
    <mds_oss_sec>krb5p</mds_oss_sec>
    <deny_sec>null</deny_sec>
  </sec>
</mds>
<ost ...>
  <sec>
    <deny_sec>krb5i</deny_sec>
  </sec>
</ost>
```

Note:
Default sec should be null if no special secure flavor option assigned.

1.2 lconf

lconf should provide folllowing function to support the secure option in config xml:

- Write the sec option to config log
- To load the secure module
- Parse the secure option to lctl
- To execute lgssd/lsvcgssd

1.3 variable to store secure flavor

1.3.1 secure flavor deny list type

```
/* mds and oss need deny list to refuse certain type connection from remote */
typedef struct {
    struct list_head list;
    ptlrpcs_flavor_t flavor;
}deny_sec_t;
```

1.3.2 secure flavor variable on mds

```
/* the secure flavor used from this mds to mds/oss */
struct mds_obd{
    ...
    char *mds_mds_sec;              /* mds<=>mds */
    char *mds_oss_sec;              /* mds<=>oss */
    struct list_head deny_list;     /* which secure flavor from remote to this mds is d
    ...
}
```

1.3.3 secure flavor variable on oss

```
/* oss only need deny list, because it accept connection from other nodes passively */
struct ost_obd{
    ...
    struct list_head deny_list;    /* which secure flavor from remote to this oss is de
    ...
}
```

1.3.4 secure flavor variable on client

```
/* client[osc/mdc] use which secure flavor to request service */
struct client_obd {
    ...
    __u32 cl_sec_flavor;
    __u32 cl_sec_subflavor;
    ...
}
```

1.4 jt_obd_set_security

1.4.1 Prototype

```
int jt_obd_set_security(int argc, char *argv[]);
```

1.4.2 Parameters

```
input: argc
input: argv
```

1.4.3 Return

```
0                   /* Success */
CMD_HELP            /* command format error */
ERR_PARAM           /* parameter invalidate */
rc                  /* return from lctl */
```

1.4.4 Description

Pass –mds_sec, –oss_sec option to mdc/osc module;
Pass –inter_mds_sec, –mds_oss_sec, –deny_sec option to mds module;
Pass –deny_sec option to oss module;
Before execute this command, you should assign the current device first.

1.4.5 Examples

lctl set_security [key] [value]

1.5 ioctl handle

In order to process the secure setting from ioctl, we need add corresponding entry in the ".o_iocontrol" method of the obd.

1.5.1 mds side:

```
mds_process_config(struct obd_device *obd, obd_count len, void *buf):
1. store the secure option from ioctl to mds_obd
```

1.5.2 oss side:

```
ost_process_config(struct obd_device *obd, obd_count len, void *buf):
1. store the secure option from ioctl to ost_obd
```

1.5.3 client side:

```
client_process_config(struct obd_device *obd, obd_count len, void *buf):
1. call mdc/osc_set_info to set the secure option
```

1.6 lustre's security on client side

1.6.1 secure option priority

1. secure option from cmdline
2. config log from MDS
3. default value (null)

1.6.2 obd_set_info & *_set_info

1. add the "sec" key process to set the flavor of the client obd.

```
mdc_set_info(struct obd_export *exp, obd_count keylen,
             void *key, obd_count vallen, void *val)
osc_set_info(struct obd_export *exp, obd_count keylen,
             void *key, obd_count vallen, void *val)
lmv_set_info(struct obd_export *exp, obd_count keylen,
             void *key, obd_count vallen, void *val)
lov_set_info(struct obd_export *exp, obd_count keylen,
             void *key, obd_count vallen, void *val)
```

1.6.3 gss_init_upcall_msg

When client prepare request, it need upcall to obtain its credential according the service it requests.

We need modify the struct gss_upcall_msg and gss_init_upcall_msg() to accomplish this issue.

Add service type to upcall msg struct:

```
struct gss_upcall_msg{
    ...
    char     gum_obdname[64];
    char     gum_srvname[64];
    uid_t    gum_uid;
    __u32    gum_ip;
}
```

Add the service type to upcall msg in gss_init_upcall_msg():

```
static void gss_init_upcall_msg(struct gss_upcall_msg *msg,
                                struct gss_sec *gsec, char *obdname,
                                uid_t uid, __u32 dest_ip)
```

1.7 lustre on server side

1.7.1 service type

unpack the secdata from client's request and store the service type to rsi

```
gss_svcsec_accept(struct ptlrpc_request *req, enum ptlrpcs_error *res)
```

Need modify struct rsi to store the service type as upcall msg.

```
struct rsi {
    ...
    rawobj_t in_srv_type;
    ...
};
```

Server need pass the service type from client's request to upcall.

```
static void rsi_request(struct cache_detail *cd,
                        struct cache_head *h, char **bpp, int *blen)
```

1.7.2 deny secure

When process the connection request from client, mds/oss need check its deny list to decide accept the connection or not. So we need modify target_handle_connnect to implement the checking before call obd_connect().

```
int target_handle_connect(struct ptlrpc_request *req)
```

1.8 GSS daemon

1.8.1 lgssd(client)

Get service type from upcall msg
1. delete the macro-defination GSSD_SERVER_NAME
2. modify handle_krb5_upcall():
use the service_name read from upcall msg to construct_servicename then do following steps.

```
void handle_krb5_upcall(struct clnt_info *clp)
```

1.8.2 lsvcgssd(server)

1. delete the MACRO define GSSD_SERVER_NAME
2. main():
use service type from command option to call gssd_acquire_cred(srv_name) to get credentian for the service

```
int gssd_acquire_cred(srv_cred_t *p_srv_cred)
```

3. handle_nullreq():
use service type from proc cache to call gss_accept_sec_context(creds[srv_type]) to check the context from client request

```
void handle_nullreq(FILE *f)
OM_uint32 KRB5_CALLCONV
gss_accept_sec_context (minor_status,
                        context_handle,
                        verifier_cred_handle,
                        input_token_buffer,
                        input_chan_bindings,
                        src_name,
                        mech_type,
                        output_token,
                        ret_flags,
                        time_rec,
                        delegated_cred_handle)
```

1.9 keytab

1. Use only one pricipal for one type service on different host
2. Divide the principal of lustre service into two pricipal which related to different service(lustre_mds & lustre_oss):

```
lustre/host@REALM ==>
    lustre_mds@REALM
    lustre_oss@REALM
```

2 Use cases:

- How will the specified functions be called by other code?
- Do the api's match?
- Design Unit, Integration and System tests for your code through use cases
- Provide performance and scalability use cases.

2.1 How will the specified functions be called by other code?

2.1.1 int jt_obd_sec_secure(int argc, char *argv[])

>From lctl's command:

```
command_t cmdlist[] = {
...
    {"set_security", jt_lcfg_set_security, 0,
     "usage: set_security key value\n"
     "       secure key on mdc: [mds_sec]\n"
     "       secure key on osc: [oss_sec]\n"
     "       secure key on mds: [inter_mds_sec|mds_oss_sec|deny_sec]\n"
     "       secure key on oss: [deny_sec]\n"
     "       value: [null|krb5i|krb5p]\n"}
...
}
```

2.1.2 mdc/osc/mds/ost_process_config

```
obd_class_ioctl
    \class_handle_ioctl
         \class_handle_ioctl
              \obd_process_config
                   \mdc_process_config
                   \osc_process_config
                   \mds_process_config
                   \ost_process_config
```

2.1.3 mdc/osc_set_info

```
obd_set_info
    \mdc_set_info()
    \osc_set_info()
1) lustre:
lustre_read_super
    \lustre_fill_super
        \lustre_process_log
            \obd_set_info("sec")
        \lustre_common_fill_super
            \obd_set_info("sec")
2) lustre_lite:
ll_read_super
    \ll_fill_super
        \lustre_common_fill_super
```

```
            \obd_set_info("sec")
3) llmount
main
    \build_data
        \parse_options
            \obd_set_info("sec")
```

2.1.4 lmv/lov_set_info()

```
obd_set_info
    \lmv_set_info
    \lov_set_info
1) class_process_config
    \class_setup
        \obd_setup
            \mds_setup
                \mds_lmv_connect()
                    \obd_set_info("sec")
                \mds_postsetup
                    \mds_lov_connect()
                        \obd_set_info("sec")
2) target_handle_connect
    \obd_connect_post
        \mds_connect_post
            \mds_lmv_connect
                \obd_set_info("sec")
```

2.1.5 gss_init_upcall_msg()

```
client_connect_import
    \ptlrpc_connect_import
         \ptlrpc_prep_req
              \ptlrpcs_cred_refresh
                   \gss_init_upcall_msg
```

2.1.6 rsi_request()

```
ptlrpc_server_handle_request
    \svcsec_accept
         \gss_svcsec_accept
              \gssd_upcall(&rsikey, ...)
                   \cache_check
                        \check_make_upcall
                             \cache_request
                                  \rsi_request
```

2.1.7 target_handle_connect()

```
ptlrpc_server_handle_request
    \mds/ost_handle
        \target_handle_connect
```

2.1.8 lgssd & lsvcgssd

Run by lconf or test scripts.

- lgssd:

```
gssd_run
    \scan_poll_results
        \handle_krb5_upcall
            \read
            \construct_servicename
```

- lsvcgssd:

```
main
    \gssd_acqurie_cred
    \gssd_run
        \handle_nullreq
            \readline
            \gss_accept_sec_context
```

2.2 Test case

2.2.1 Secure option

- Secure option while mount lustre at client

```
Note:
1. lmc --mds_sec krb5p --oss_sec krb5i config.xml.
2. lconf --node ost1 config.xml
3. lconf --node mds1 config.xml
4. mount -t lustre server:/mds1/client /mnt/lustre
Note: Now connections to MDS's are privacy protected, connections to OSS's are integrit
5. umount lustre.
4. mount -t lustre -o mds_sec=krb5i,oss_sec=krb5p server:/mds1/client /mnt/lustre
Note: Now connections to MDS's are integrity protected, connections to OSS's are privac
5. umount lustre.
6. mount -t lustre -o mds_sec=krb5p,oss_sec=krb5p server:/mds1/client /mnt/lustre
Note: Now connections to all MDS's and OSS's are privacy protected.
```

- Startup MDS

```
1. lmc --inter_mds_sec krb5p --mds_oss_sec krb5p config.xml
2. MDS: lconf --node mds config.xml.
Note: Now connections between MDS's and MDS's to OSS's are privacy protected.
3. MDS: lconf --node mds --cleanup config.xml
4. MDS: lconf --node mds --inter_mds_sec=krb5i --mds_oss_sec=krb5p config.xml.
Note: Now connections between MDS's are integrity protected, while MDS's to OSS's are p
```

2.2.2 Deny secure option

```
1. lconf --node ost1 --deny_sec=null config.xml
2. MDS: lconf --node mds1 --mds_oss_sec=null config.xml
/* setup will fail because OST reject connection from MDS's */
3. MDS: lconf --node mds1 --deny_sec=null --mds_oss_sec=krb5i config.xml
/* setup should succeed */
4. Client: mount -t lustre -o mds_sec=null server:/mds1/client /mnt/lustre
/* mount fail because either MDS's or OSS's will reject null secure connection */
5. Client: mount -t lustre -o oss_sec=null server:/mds1/client /mnt/lustre
/* mount fail because either MDS's or OSS's will reject null secure connection */
6. Client: mount -t lustre -o mds_sec=krb5i,oss_sec=krb5i server:/mds1/client /mnt/lust
/* mount should succeed */
```

2.3 Performance case

There 4 type connects in lustre:
1) Client <=> MDS
2) Client <=> OSS
3) MDS <=> MDS
4) MDS <=> OSS
Each type connection have 3 secure flavor option:
A) NULL
B) KRB5I
C) KRB5P
So, there are 3*3*3*3 = 81 situation to test the performance.
Iterate different secure flavor to setup lustre and run bonnie to compare the performance.

2.4 Scalability case

N/A

3 Logic specification:

- Provide pseudo code for critical elements and API usage.

- What do the internals of the changes look like?
- How do they call other methods? Do the API's match?

3.1 lmc

N/A

3.2 lconf

3.2.1 Write the sec option to config log:

```
lconf: write_conf();
```

3.2.2 To load secure module

```
N/A
```

3.2.3 Parse the sec option in config xml to lctl:

```
switch obd_device:
    /* client */
    mds_sec:
             jt_lcfg_set_security("mds_sec", flavor);
    oss_sec:
             jt_lcfg_set_security("oss_sec", flavor);
    /* mds */
    inter_mds_sec:
             jt_lcfg_set_security("mds_mds_sec", flavor);
    mds_oss_sec:
             jt_lcfg_set_security("mds_oss_sec", flavor);
    mds_deny_sec:
             jt_lcfg_set_security("deny_sec", flavor);
    /* oss */
    oss_deny_sec:
             jt_lcfg_set_security("deny_sec", flavor);
```

3.2.4 To execute lgssd/lsvcgssd:

```
lconf: run()
Note:
    Add service_type argument to lsvcgssd to indicate which service's cred to be loaded
```

3.3 ioctl command

3.3.1 int jt_lcfg_set_security(int argc, char *argv[]);

```
{
    struct lustre_cfg lcfg;
    LCFG_INIT(lcfg, LCFG_SET_SECURITY, lcfg_devname);
    lcfg.lcfg_inllen1 = strlen(argv[1]) + 1;
    lcfg.lcfg_inlbuf1 = argv[1];
    lcfg.lcfg_inllen2 = strlen(argv[2]) + 1;
    lcfg.lcfg_inlbuf2 = argv[2];
    rc = lcfg_ioctl(argv[0], OBD_DEV_ID, &lcfg);

    return 0;
}
```

3.3.2 client_process_config():

```
struct obd_ops osc_obd_ops = {
    ...
    .o_process_config = client_process_config,
    ...
};
struct obd_ops mdc_obd_ops = {
    ...
    .o_process_config = client_process_config,
    ...
};
client_process_config()
{
  switch (lcfg->lcfg_command {
    case LCFG_SET_SECURITY:
        if (!strcmp(lcfg->lcfg_inlbuf1, "mds_sec"))
            rc = obd_set_info(obd->obd_self_export, strlen("mds_sec"), "mds_sec",
                              strlen(lcfg->lcfg_inlbuf2), lcfg->lcfg_inlbuf2);
        else if (!strcmp(lcfg->lcfg_inlbuf1, "oss_sec"))
            rc = obd_set_info(obd->obd_self_export, strlen("oss_sec"), "oss_sec",
                              strlen(lcfg->lcfg_inlbuf2), lcfg->lcfg_inlbuf2);
        } else {
            CERROR("Unrecognized key\n");
            rc = -EINVAL;
        }
        break;
  }
}
```

3.3.3 mds_process_config

```
static struct obd_ops mds_obd_ops = {
    ...
    .o_process_config  = mds_process_config,
    ...
};
struct mds_obd *mds = &obd->u.mds;
...
switch(lcfg->lcfg_command) {
    case LCFG_SET_SECURITY:
        if (!strcmp(lcfg->lcfg_inlbuf1, "mds_mds_sec"))
            rc = set_security(lcfg->lcfg_inlbuf2, &mds->mds_mds_sec);
        else if (!strcmp(lcfg->lcfg_inlbuf1, "mds_ost_sec"))
            rc = set_security(lcfg->lcfg_inlbuf2, &mds->mds_ost_sec);
        else if (!strcmp(lcfg->lcfg_inlbuf1, "deny_sec")){
            rc = add_deny_security(lcfg->lcfg_inlbuf2, &mds->deny_list);
        } else {
            CERROR("Unrecognized key\n");
            rc = -EINVAL;
        }
        break;
}
```

3.3.4 ost_process_config:

```
static struct obd_ops ost_obd_ops = {
    ...
    .o_process_config = ost_process_config,
    ...
};
struct ost_obd *ost = &obd->u.ost;
...
switch(lcfg->lcfg_command) {
    case LCFG_SET_SECURITY:
        if (!strcmp(lcfg->lcfg_inlbuf1, "deny_sec")){
            rc = add_deny_security(lcfg->lcfg_inlbuf2, &ost->deny_list);
        } else {
            CERROR("Unrecognized key\n");
            rc = -EINVAL;
        }
        break;
}
```

3.4 client side: (keyword: "sec")

3.4.1 mdc/osc_set_info()

```
int mdc_set_info(struct obd_export *exp, obd_count keylen,
                 void *key, obd_count vallen, void *val)
{
...
     else if (keylen == strlen("mds_sec") && memcmp(key, "mds_sec", keylen) == 0) {
         struct client_obd *cli = &exp->exp_obd->u.cli;

         if (vallen == strlen("null") && memcmp(val, "null", vallen) == 0) {
             cli->cl_sec_flavor = PTLRPC_SEC_NULL;
             cli->cl_sec_subflavor = 0;
             RETURN(0);
         }
         if (vallen == strlen("krb5i") && memcmp(val, "krb5i", vallen) == 0) {
             cli->cl_sec_flavor = PTLRPC_SEC_GSS;
             cli->cl_sec_subflavor = PTLRPC_SEC_GSS_KRB5I;
             RETURN(0);
         }
         if (vallen == strlen("krb5p") && memcmp(val, "krb5p", vallen) == 0) {
             cli->cl_sec_flavor = PTLRPC_SEC_GSS;
             cli->cl_sec_subflavor = PTLRPC_SEC_GSS_KRB5P;
             RETURN(0);
         }
         CERROR("unrecognized security type %s\n", (char*) val);
         rc = -EINVAL;
     }
...
}
```

3.4.2 interactive with gssd on client

- Add service type to upcall msg;

```
struct gss_upcall_msg{
    ...
    char     gum_obdname[64];
    char     gum_srvname[64];
    uid_t    gum_uid;
    __u32    gum_ip;
}
static void gss_init_upcall_msg(struct gss_upcall_msg *gmsg, struct gss_sec *gsec, char
{
    ...
```

```
    if (!strcmp(obdtype, "mdc"))
        strncpy(gmsg->gum_srvname, "lustre_mds", sizeof(gmsg->gum_srvname));
    else if (!strcmp(obdtype, "osc"))
        strncpy(gmsg->gum_srvname, "lustre_oss", sizeof(gmsg->gum_srvname));
    ...
    rpcmsg->len = sizeof(gmsg->gum_uid) + sizeof(gmsg->gum_ip) + sizeof(gmsg->gum_srvna
    ...
}
```

- Add service type argument while generate init secure request buf

```
gss_send_secinit_rpc()
{
    ...
    if (strcmp(obd->obd_type->typ_name, "mdc"){
        srv_type = LUSTRE_MDS;
    }else if (strcmp(obd->obd_type->typ_name, "osc")){
        srv_type = LUSTRE_OSS;
    }else{
        CERROR("...");
        return (-EINVAL);
    }
    ...
    reqlen = secinit_compose_request(reqbuf, reqbuf_size, srv_type, (char *)inbuf[1]);
    ...
}
static int secinit_compose_request(char *buf, int bufsize, int srv_type, char __user *t
{
    ...
    *p++ = cpu_to_le32(8 * 4 + token_size);
    ...
    *p++ = cpu_to_le32(PTLRPC_GSS_SRV_NONE);
    *p++ = cpu_to_le32(srv_type);
    ...
}
```

3.5 gssd:

add service_type argument to updata, and modify construct_servicename() to use srv_name from upcall msg.

```
service_typehandle_krb5_upcall(struct clnt_info *clp)
{
    struct {
        uid_t        uid;
        unsigned int ip;
```

```
        char         srv_name[MAX_LEN_SRV_NAME];
    } updata;
    ...
    if (construct_servicename(clp, updata.ip, updata.srv_name))
    ...
    lgd = lgssd_create_default(clp->dirname, clp->servicename, &sec);
    ...
}
static int construct_servicename(struct clnt_info *clp, unsigned int ip, char *service)
{
    ...
    clp->servicename = malloc(strlen(service) + 1 + strlen(ent->h_name) + 1);
    sprintf(clp->servicename,"%s@%s", service, ent->h_name);
    ...
}
```

3.6 server side:(keyword: cl_sec_subflavor)

3.6.1 target_handle_connect()

check client's secure flavor according to client's secure flavor and target obd's deny list

```
target_handle_connect()
{
    ...
dont_check_exports:
            if (!strcmp(target->obd_type->typ_name, "mds")){
                rc = check_deny_list(&target->u.mds.deny_list, &req->rq_cred->pc_sec.ps
            }else if (!strcmp(target->obd_type->typ_name, "oss")){
                rc = check_deny_list(&target->u.ost.deny_list, &req->rq_cred->pc_sec.ps
            }
            if (rc != 0)
                GOTO(out, rc);
            rc = obd_connect(&conn, target, &cluuid,connect_flags);
    ...
}
int check_deny_list(struct list_head *head, ptlrpcs_flavor_t *p_flavor)
{
    int rc = 0;
    deny_sec_t *p_deny_sec;
    while (!list_empty(head)) {
        p_deny_sec = list_entry(head->next, deny_sec_t, list);
        if ((p_deny_sec.flavor.flavor == p_flavor->flavor)
           && (p_deny_sec.flavor.subflavor == p_flavor->subflavor)){
            rc = 1, break;
```

```
        }
    }
    return rc;
}
```

3.6.2 set the secure flavor while connect from mds to mds/ost

```
mds_lmv_connect(
{
...
   if (mds->mds_mds_sec) {
       rc = obd_set_info(mds->mds_lmv_exp, strlen("sec"), "sec",
                         strlen(mds->mds_mds_sec), mds->mds_mds_sec);
       if (rc)
           GOTO(err_reg, rc);
    }
...
}
mds_lov_connect(struct obd_device *obd, char *lov_name)
{
...
    if (mds->mds_ost_sec) {
        rc = obd_set_info(mds->mds_lov_obd->obd_self_export, strlen("sec"),
                          "sec", strlen(mds->mds_ost_sec), mds->mds_ost_sec);
        if (rc) {
            mds->mds_lov_obd = ERR_PTR(rc);
            RETURN(rc);
        }
     }
...
}
```

3.6.3 interactive with lsvcgssd on server

- pass the srv_type from client to rsi

```
struct rsi {
    ...
    rawobj_t    in_srv_type;
    ...
};
gss_svcsec_accept(struct ptlrpc_request *req, enum ptlrpcs_error *res)
{
    ...
    srv_type = le32_to_cpu(*secdata++);
    seclen -= 6*4;
    ...
```

```
    rawobj_dup(&rsikey->int_srv_type, &srv_type);
    rsip = gssd_upcall(rsikey, &my_chandle);
    ...
}
```

- rsi_request() /* add service_type */

```
rsi_request(struct cache_detail *cd, struct cache_head *h, char **bpp, int *blen)
{
    ...
    qword_addhex(bpp, blen,  rsii->in_srv_type, rsii->in_srv_type.len);
    ...
}
```

3.7 lsvcgssd

- give service type argument to run lsvcgssd in lconf, call gssd_acquire_cred to get all service cred

```
#define NR_MAX_SERVICE 8
typedef struct{
    int srv_id;
    char srv_name[MAX_LEN_SRV_NAME];
}srv_id_name_t;
enum lustre_service_type{
    0,
    LUSTRE_MDS,
    LUSTRE_OSS,
};
srv_id_name_t srv_id_name[]={
    {0, ""},
    {LUSTRE_MDS, "lustre_mds"},
    {LUSTRE_OSS, "lustre_oss"},
    {NR_MAX_SERVICE, "max_service"}
};
typedef struct{
    int            srv_id;
    char           *srv_name;
    gss_cred_it_t  gssd_cred;
}srv_cred_t;
main(){
    srv_cred_t srv[NR_MAX_SERVICE];
    int i = 0;
    while ((opt = getopt(argc, argv, "s:") )!= -1){
        switch (opt) {
                case 's':
```

```
                        srv[i++].srv_name = optarg;
                        srv[i++].srv_id = srv_name2id(optarg);
                        break;
                ...
        }
    }
    ...
    while (i>0){
        i--;
        gssd_acquire_cred(srv[i]);
    }
    ...
}
int gssd_acquire_cred(srv_cred_t *p_srv_cred){
    ...
    name.value = (void *)p_srv_cred->srv_name;
    name.length = strlen(p_srv_cred->srv_name);
    ...
    maj_stat = gss_acquire_cred(..., &p_srv_cred->gssd_cred, NULL,NULL);
    ...
}
```

- according to the service name passed from lustre module to check the context

```
void handle_nullreq(FILE *f){
    ...
    srv_type.length = qword_get(&cp, srv_type.value, sizeof(srv_type_buf));
    srv_type = srv_name2id(srv_type_buf);
    ...
    /* find the gssd_cred whose srv_id eq service_type from upcall */
    for (i = 0; i<NR_MAX_SERVICE; i++){
        if (srv[i].srv_id == srv_type){
            break;
        }
    }
    maj_stat = gss_accept_sec_context(... , srv[i].gssd_cred, ...);
    ...
}
```

4 State:

- What shared state is affected?

  None

- In detail, how is this protected with locks?

None

- Recovery: How is recovery addressed?

No effect on recovery, it's transparent to all layer above ptlrpc.

5 Environment:

- What network api's are used?

None

- What disk and configuration state is changed? Discuss compatibility with older versions.

krb5 principle database

- What documentation changes are necessary?

Lctl manual