

new/usr/src/cmd/gss/gssd/gssd\_clnt\_stubs.c

1

```
*****
67754 Thu May 7 01:13:42 2009
new/usr/src/cmd/gss/gssd/gssd_clnt_stubs.c
6817447 libgss and various mechs are hiding both the real minor_status and the e
6405422 Solaris acceptors fail in AD-KDC environments when using non-"host" serv
6824434 Unable to accept context establishment initiated by Windows 2000 clients
6787343 kclient's site lookups fail in certain network environments
6692646 kclient should output errors to stderr
6525327 kinit failed when arcfour-hmac-md5-exp was used for the principal's key
6745582 SUNWkdcu missing package dependencies after kclientv2 integration
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  * Common Development and Distribution License, Version 1.0 only
8  * (the "License"). You may not use this file except in compliance
9  * with the License.
10 *
11 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
12 * or http://www.opensolaris.org/os/licensing.
13 * See the License for the specific language governing permissions
14 * and limitations under the License.
15 *
16 * When distributing Covered Code, include this CDDL HEADER in each
17 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
18 * If applicable, add the following below this CDDL HEADER, with the
19 * fields enclosed by brackets "[]" replaced with your own identifying
20 * information: Portions Copyright [yyyy] [name of copyright owner]
21 *
22 * CDDL HEADER END
23 */
24
25 /*
26
27 #pragma ident "%Z%M% %I% %E% SMI"
28
29 */
30
31 #include <stdio.h>
32 #include <stdlib.h>
33 #include <mechglueP.h>
34 #include "gssd.h"
35 #include <rpc/rpc.h>
36
37 #ifdef _KERNEL
38 #define MALLOC(n) kmem_alloc((n), KM_SLEEP)
39 #define FREE(x, n) kmem_free((x), (n))
40 #define memcpy(dst, src, n) bcopy((src), (dst), (n))
41 #define clnt_pcreateerror(srv) printf("Cannot connect to server on %s\n", srv)
42
43 #ifdef DEBUG
44 #ifndef _SYS_CMN_ERR_H
45 #define _SYS_CMN_ERR_H
46 #define CE_NOTE 1
47 #endif
48 #include <sys/types.h>
49 #include <sys/devops.h>
```

new/usr/src/cmd/gss/gssd/gssd\_clnt\_stubs.c

2

```
50 #include <sys/open.h>
51 #include <sys/stat.h>
52 #include <sys/conf.h>
53 #include <sys/ddi.h>
54 #include <sys/sunddi.h>
55 #include <sys/uio.h>
56 #endif /* DEBUG */
57
58 #else /* !_KERNEL */
59 #define MALLOC(n) malloc(n)
60 #define FREE(x, n) free(x)
61 #endif /* _KERNEL */
62 #define DEFAULT_MINOR_STAT ((OM_uint32) ~0)
63
64 CLIENT *clnt, *getgssd_handle();
65 char *server = "localhost";
66
67 OM_uint32
68 kgss_acquire_cred_wrapped(minor_status,
69                             desired_name,
70                             time_req,
71                             desired_mechs,
72                             cred_usage,
73                             output_cred_handle,
74                             actual_mechs,
75                             time_rec,
76                             uid,
77                             gssd_cred_verifier)
78 OM_uint32 *minor_status;
79 gss_name_t desired_name;
80 OM_uint32 time_req;
81 gss_OID_set desired_mechs;
82 int cred_usage;
83 gssd_cred_id_t *output_cred_handle;
84 gss_OID_set *actual_mechs;
85 OM_uint32 *time_rec;
86 uid_t uid;
87 OM_uint32 *gssd_cred_verifier;
88 {
89     OM_uint32 minor_status_temp;
90     gss_buffer_desc external_name;
91     gss_OID name_type;
92     int i;
93
94     gss_acquire_cred_arg arg;
95     gss_acquire_cred_res res;
96
97     /* get the client handle to GSSD */
98
99     if ((clnt = getgssd_handle()) == NULL) {
100         clnt_pcreateerror(server);
101         return (GSS_S_FAILURE);
102     }
103
104     /* convert the desired name from internal to external format */
105
106     if (gss_display_name(&minor_status_temp, desired_name, &external_name,
107                         &name_type) != GSS_S_COMPLETE) {
108
109         *minor_status = (OM_uint32) minor_status_temp;
110         gss_release_buffer(&minor_status_temp, &external_name);
111         return ((OM_uint32) GSS_S_FAILURE);
112     }
113
114     /* copy the procedure arguments into the rpc arg parameter */
```

```

117     arg.uid = (OM_uint32)uid;
118
119     arg.desired_name.GSS_BUFFER_T_len = (uint_t)external_name.length;
120     arg.desired_name.GSS_BUFFER_T_val = (char *)external_name.value;
121
122     arg.name_type.GSS_OID_len =
123         name_type == GSS_C_NULL_OID ?
124         0 : (uint_t)name_type->length;
125
126     arg.name_type.GSS_OID_val =
127         name_type == GSS_C_NULL_OID ?
128         (char *)NULL : (char *)name_type->elements;
129
130     arg.time_req = time_req;
131
132     if (desired_mechs != GSS_C_NULL_OID_SET) {
133         arg.desired_mechs.GSS_OID_SET_len =
134             (uint_t)desired_mechs->count;
135         arg.desired_mechs.GSS_OID_SET_val = (GSS_OID *)
136             MALLOC(sizeof (GSS_OID) * desired_mechs->count);
137
138         for (i = 0; i < desired_mechs->count; i++) {
139             arg.desired_mechs.GSS_OID_SET_val[i].GSS_OID_len =
140                 (uint_t)desired_mechs->elements[i].length;
141             arg.desired_mechs.GSS_OID_SET_val[i].GSS_OID_val =
142                 (char *)
143                 MALLOC(desired_mechs->elements[i].length);
144             memcpy(arg.desired_mechs.GSS_OID_SET_val[i].GSS_OID_val,
145                 desired_mechs->elements[i].elements,
146                 desired_mechs->elements[i].length);
147         }
148     } else
149         arg.desired_mechs.GSS_OID_SET_len = 0;
150
151     arg.cred_usage = cred_usage;
152
153     /* call the remote procedure */
154
155     memset(&res, 0, sizeof (res));
156     if (gss_acquire_cred_l(&arg, &res, clnt) != RPC_SUCCESS) {
157
158         /*
159          * if the RPC call times out, null out all return arguments,
160          * set minor_status to its maximum value, and return GSS_S_FAILURE
161          */
162
163         if (minor_status != NULL)
164             *minor_status = DEFAULT_MINOR_STAT;
165         *minor_status = 0xffffffff;
166         if (output_cred_handle != NULL)
167             *output_cred_handle = NULL;
168         if (actual_mechs != NULL)
169             *actual_mechs = NULL;
170         if (time_rec != NULL)
171             *time_rec = 0;
172
173         return (GSS_S_FAILURE);
174     }
175
176     /* free the allocated memory for the flattened name and desire_mechs */
177
178     gss_release_buffer(&minor_status_temp, &external_name);
179     for (i = 0; i < desired_mechs->count; i++)
180         FREE(arg.desired_mechs.GSS_OID_SET_val[i].GSS_OID_val,
181             arg.desired_mechs.GSS_OID_SET_val[i].GSS_OID_len);

```

```

181     FREE(arg.desired_mechs.GSS_OID_SET_val,
182         arg.desired_mechs.GSS_OID_SET_len * sizeof (GSS_OID));
183
184     /* copy the rpc results into the return arguments */
185
186     if (minor_status != NULL)
187         *minor_status = res.minor_status;
188
189     if (output_cred_handle != NULL) {
190         *output_cred_handle =
191             /*LINTED*/
192             *((gssd_cred_id_t *)res.output_cred_handle.GSS_CRED_ID_T_val);
193         *gssd_cred_verifier = res.gssd_cred_verifier;
194     }
195
196     if (res.status == GSS_S_COMPLETE &&
197         res.actual_mechs.GSS_OID_SET_len != 0 &&
198         actual_mechs != NULL) {
199         *actual_mechs = (gss_OID_set) MALLOC(sizeof (gss_OID_set_desc));
200         (*actual_mechs)->count =
201             (int)res.actual_mechs.GSS_OID_SET_len;
202         (*actual_mechs)->elements = (gss_OID)
203             MALLOC(sizeof (gss_OID_desc) * (*actual_mechs)->count);
204
205         for (i = 0; i < (*actual_mechs)->count; i++) {
206             (*actual_mechs)->elements[i].length = (OM_uint32)
207                 res.actual_mechs.GSS_OID_SET_val[i].GSS_OID_len;
208             (*actual_mechs)->elements[i].elements =
209                 (void *) MALLOC((*actual_mechs)->elements[i].length);
210             memcpy((*actual_mechs)->elements[i].elements,
211                 res.actual_mechs.GSS_OID_SET_val[i].GSS_OID_val,
212                 (*actual_mechs)->elements[i].length);
213         }
214     } else {
215         if (res.status == GSS_S_COMPLETE && actual_mechs != NULL)
216             (*actual_mechs)->count = 0;
217     }
218
219     if (time_rec != NULL)
220         *time_rec = res.time_rec;
221
222     /*
223      * free the memory allocated for the results and return with the status
224      * received in the rpc call
225      */
226
227     clnt_freeres(clnt, xdr_gss_acquire_cred_res, (caddr_t)&res);
228     return (res.status);
229 }

```

unchanged\_portion\_omitted

```

468 OM_uint32
469 kgss_release_cred_wrapped(minor_status,
470     cred_handle,
471     uid,
472     gssd_cred_verifier)
473 OM_uint32 *minor_status;
474 gssd_cred_id_t *cred_handle;
475 uid_t uid;
476 OM_uint32 gssd_cred_verifier;
477 {
478
479     gss_release_cred_arg arg;
480     gss_release_cred_res res;

```

```

483  /* get the client handle to GSSD */
484  if ((clnt = getgssd_handle()) == NULL) {
485      clnt_pcreateerror(server);
486      return (GSS_S_FAILURE);
487  }

489  /* copy the procedure arguments into the rpc arg parameter */

491  arg.uid = (OM_uint32) uid;
492  arg.gssd_cred_verifier = gssd_cred_verifier;

494  if (cred_handle != NULL) {
495      arg.cred_handle.GSS_CRED_ID_T_len =
496          (uint_t)sizeof (gssd_cred_id_t);
497      arg.cred_handle.GSS_CRED_ID_T_val = (char *)cred_handle;
498  } else
499      arg.cred_handle.GSS_CRED_ID_T_len = 0;

501  /* call the remote procedure */

503  memset(&res, 0, sizeof (res));
504  if (gss_release_cred_l(&arg, &res, clnt) != RPC_SUCCESS) {

506      /*
507       * if the RPC call times out, null out all return arguments,
508       * set minor_status to its max value, and return GSS_S_FAILURE
509       */

511      if (minor_status != NULL)
512          *minor_status = DEFAULT_MINOR_STAT;
514      if (cred_handle != NULL)
515          *cred_handle = NULL;

516      return (GSS_S_FAILURE);
517  }

519  /* if the release succeeded, null out the cred_handle */
520  if (res.status == GSS_S_COMPLETE && cred_handle != NULL)
521      *cred_handle = NULL;

523  /* copy the rpc results into the return arguments */
524  if (minor_status != NULL)
525      *minor_status = res.minor_status;

527  /* return with status returned in rpc call */
528  return (res.status);
529 }

```

unchanged portion omitted

```

556 OM_uint32
557 kgss_init_sec_context_wrapped(minor_status,
558                               claimant_cred_handle,
559                               gssd_cred_verifier,
560                               context_handle,
561                               gssd_context_verifier,
562                               target_name,
563                               mech_type,
564                               req_flags,
565                               time_req,
566                               input_chan_bindings,
567                               input_token,
568                               actual_mech_type,
569                               output_token,
570                               ret_flags,
571                               time_rec,

```

```

572      uid)
573      OM_uint32 *minor_status;
574      gssd_cred_id_t claimant_cred_handle;
575      OM_uint32 gssd_cred_verifier;
576      OM_uint32 *context_handle;
577      OM_uint32 *gssd_context_verifier;
578      gss_name_t target_name;
579      gss_OID mech_type;
580      int req_flags;
581      OM_uint32 time_req;
582      gss_channel_bindings_t input_chan_bindings;
583      gss_buffer_t input_token;
584      gss_OID *actual_mech_type;
585      gss_buffer_t output_token;
586      int *ret_flags;
587      OM_uint32 *time_rec;
588      uid_t uid;
589  {
590      OM_uint32 minor_status_temp;
591      gss_buffer_desc external_name;
592      gss_OID name_type;
593      gss_init_sec_context_arg arg;
594      gss_init_sec_context_res res;

596      /* get the client handle to GSSD */

598      if ((clnt = getgssd_handle()) == NULL) {
599          clnt_pcreateerror(server);
600          return (GSS_S_FAILURE);
601      }

603      /* convert the target name from internal to external format */

605      if (gss_display_name(&minor_status_temp, target_name,
606                          &external_name, &name_type) != GSS_S_COMPLETE) {

608          *minor_status = (OM_uint32) minor_status_temp;
609          return ((OM_uint32) GSS_S_FAILURE);
610      }

613 /* copy the procedure arguments into the rpc arg parameter */

615      arg.uid = (OM_uint32) uid;

617      arg.context_handle.GSS_CTX_ID_T_len =
618          *context_handle == (OM_uint32) GSS_C_NO_CONTEXT ? 0 :
619          (uint_t)sizeof (OM_uint32);
620      arg.context_handle.GSS_CTX_ID_T_val = (char *)context_handle;
621      arg.gssd_context_verifier = *gssd_context_verifier;

623      arg.claimant_cred_handle.GSS_CRED_ID_T_len =
624          claimant_cred_handle == (gssd_cred_id_t)GSS_C_NO_CREDENTIAL ?
625          0 : (uint_t)sizeof (gssd_cred_id_t);
626      arg.claimant_cred_handle.GSS_CRED_ID_T_val =
627          (char *)&claimant_cred_handle;
628      arg.gssd_cred_verifier = gssd_cred_verifier;

630      arg.target_name.GSS_BUFFER_T_len = (uint_t)external_name.length;
631      arg.target_name.GSS_BUFFER_T_val = (char *)external_name.value;

633      arg.name_type.GSS_OID_len =
634          name_type == GSS_C_NULL_OID ?
635          0 : (uint_t)name_type->length;

637      arg.name_type.GSS_OID_val =

```

```

638     name_type == GSS_C_NULL_OID ?
639     (char *)NULL : (char *)name_type->elements;

641     arg.mech_type.GSS_OID_len = (uint_t)(mech_type != GSS_C_NULL_OID ?
642     mech_type->length : 0);
643     arg.mech_type.GSS_OID_val = (char *) (mech_type != GSS_C_NULL_OID ?
644     mech_type->elements : 0);

646     arg.req_flags = req_flags;

648     arg.time_req = time_req;

650     if (input_chan_bindings != GSS_C_NO_CHANNEL_BINDINGS) {
651         arg.input_chan_bindings.present = YES;
652         arg.input_chan_bindings.initiator_addrtype =
653             input_chan_bindings->initiator_addrtype;
654         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_len =
655             (uint_t)input_chan_bindings->initiator_address.length;
656         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_val =
657             (void *) input_chan_bindings->initiator_address.value;
658         arg.input_chan_bindings.acceptor_addrtype =
659             input_chan_bindings->acceptor_addrtype;
660         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_len =
661             (uint_t)input_chan_bindings->acceptor_address.length;
662         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_val =
663             (void *) input_chan_bindings->acceptor_address.value;
664         arg.input_chan_bindings.application_data.GSS_BUFFER_T_len =
665             (uint_t)input_chan_bindings->application_data.length;
666         arg.input_chan_bindings.application_data.GSS_BUFFER_T_val =
667             (void *) input_chan_bindings->application_data.value;
668     } else {
669         arg.input_chan_bindings.present = NO;
670         arg.input_chan_bindings.initiator_addrtype = 0;
671         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_len = 0;
672         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_val = 0;
673         arg.input_chan_bindings.acceptor_addrtype = 0;
674         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_len = 0;
675         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_val = 0;
676         arg.input_chan_bindings.application_data.GSS_BUFFER_T_len = 0;
677         arg.input_chan_bindings.application_data.GSS_BUFFER_T_val = 0;
678     }

680     arg.input_token.GSS_BUFFER_T_len = (uint_t)
681     (input_token != GSS_C_NO_BUFFER ? input_token->length : 0);
682     arg.input_token.GSS_BUFFER_T_val = (char *)
683     (input_token != GSS_C_NO_BUFFER ? input_token->value : 0);

685     /* initialize the output parameters to empty values */
686     if (minor_status != NULL)
687         *minor_status = DEFAULT_MINOR_STAT;
688     *minor_status = 0xffffffff;
689     if (actual_mech_type != NULL)
690         *actual_mech_type = NULL;
691     if (output_token != NULL)
692         output_token->length = 0;
693     if (ret_flags != NULL)
694         *ret_flags = 0;
695     if (time_rec != NULL)
696         *time_rec = 0;

697     /* call the remote procedure */
698     memset(&res, 0, sizeof(res));
699     if (gss_init_sec_context_l(&arg, &res, clnt) != RPC_SUCCESS) {

701         /* free the allocated memory for the flattened name */
702         gss_release_buffer(&minor_status_temp, &external_name);

```

```

704         return (GSS_S_FAILURE);
705     }

707     /*
708     * We could return from a GSS error here and need to return both the
709     * minor_status and output_token, back to the caller if applicable.
710     */
711     if (minor_status != NULL)
712         *minor_status = res.minor_status;

714     if (output_token != NULL && res.output_token.GSS_BUFFER_T_val != NULL) {
715         output_token->length =
716             (size_t)res.output_token.GSS_BUFFER_T_len;
717         output_token->value =
718             (void *)res.output_token.GSS_BUFFER_T_val;
719         res.output_token.GSS_BUFFER_T_val = NULL;
720         res.output_token.GSS_BUFFER_T_len = 0;
721     }

723     /* free the allocated memory for the flattened name */
724     gss_release_buffer(&minor_status_temp, &external_name);

726     /* if the call was successful, copy out the results */
727     if (res.status == (OM_uint32) GSS_S_COMPLETE ||
728         res.status == (OM_uint32) GSS_S_CONTINUE_NEEDED) {
729         /*
730         * copy the rpc results into the return argument
731         * on CONTINUE_NEEDED only ctx handle is ready.
732         * copy the rpc results into the return arguments
733         * on CONTINUE_NEEDED only the output token, minor
734         * code and ctxt handle are ready.
735         */
736         if (minor_status != NULL)
737             *minor_status = res.minor_status;
738         /*LINTED*/
739         *context_handle = *((OM_uint32 *)
740             res.context_handle.GSS_CTX_ID_T_val);

742         /*LINTED*/
743         *context_handle = *((OM_uint32 *)
744             res.context_handle.GSS_CTX_ID_T_val);
745         *gssd_context_verifier = res.gssd_context_verifier;

747         if (output_token != NULL) {
748             output_token->length =
749                 (size_t)res.output_token.GSS_BUFFER_T_len;
750             output_token->value =
751                 (void *)res.output_token.GSS_BUFFER_T_val;
752             res.output_token.GSS_BUFFER_T_val = NULL;
753             res.output_token.GSS_BUFFER_T_len = 0;
754         }

756         /* the rest of the parameters is only ready on COMPLETE */
757         if (res.status == GSS_S_COMPLETE) {
758             if (actual_mech_type != NULL) {
759                 *actual_mech_type = (gss_OID)
760                     MALLOC(sizeof(gss_OID_desc));
761                 (*actual_mech_type)->length = (OM_UINT32)
762                     res.actual_mech_type.GSS_OID_len;
763                 (*actual_mech_type)->elements = (void *)
764                     MALLOC((*actual_mech_type)->length);
765                 memcpy((*actual_mech_type)->elements, (void *)
766                     res.actual_mech_type.GSS_OID_val,
767                     (*actual_mech_type)->length);
768             }
769         }

```

```

754         if (ret_flags != NULL)
755             *ret_flags = res.ret_flags;

757         if (time_rec != NULL)
758             *time_rec = res.time_rec;
759     }
760 }

763 /*
764  * free the memory allocated for the results and return with the
765  * status received in the rpc call.
766  */

768     clnt_freeres(clnt, xdr_gss_init_sec_context_res, (caddr_t)&res);
769     return (res.status);
770 }

unchanged portion omitted
828 OM_uint32
829 kgss_accept_sec_context_wrapped(minor_status,
830     context_handle,
831     gssd_context_verifier,
832     verifier_cred_handle,
833     gssd_cred_verifier,
834     input_token,
835     input_chan_bindings,
836     src_name,
837     mech_type,
838     output_token,
839     ret_flags,
840     time_rec,
841     delegated_cred_handle,
842     uid)
843     OM_uint32 *minor_status;
844     gssd_ctx_id_t *context_handle;
845     OM_uint32 *gssd_context_verifier;
846     gssd_cred_id_t verifier_cred_handle;
847     OM_uint32 gssd_cred_verifier;
848     gss_buffer_t input_token;
849     gss_channel_bindings_t input_chan_bindings;
850     gss_buffer_t src_name;
851     gss_OID *mech_type;
852     gss_buffer_t output_token;
853     int *ret_flags;
854     OM_uint32 *time_rec;
855     gss_cred_id_t *delegated_cred_handle;
856     uid_t uid;
857 {
858     gss_accept_sec_context_arg arg;
859     gss_accept_sec_context_res res;
860     struct kgss_cred *kcred;

862     /* get the client handle to GSSD */
863     if ((clnt = getgssd_handle()) == NULL) {
864         clnt_pcreateerror(server);
865         return (GSS_S_FAILURE);
866     }

868     /* copy the procedure arguments into the rpc arg parameter */
869     arg.uid = (OM_uint32) uid;

871     arg.context_handle.GSS_CTX_ID_T_val =
872         *context_handle == (gssd_ctx_id_t)GSS_C_NO_CONTEXT ?
873         0 : (uint_t)sizeof (gssd_ctx_id_t);

```

```

874     arg.context_handle.GSS_CTX_ID_T_val = (char *)context_handle;
875     arg.gssd_context_verifier =
876         *context_handle == (OM_uint32) GSS_C_NO_CONTEXT ?
877         0 : *gssd_context_verifier;

879     arg.verifier_cred_handle.GSS_CRED_ID_T_len =
880         verifier_cred_handle ==
881         (gssd_cred_id_t)GSS_C_NO_CREDENTIAL ?
882         0 : (uint_t)sizeof (gssd_cred_id_t);
883     arg.verifier_cred_handle.GSS_CRED_ID_T_val =
884         (char *)&verifier_cred_handle;
885     arg.gssd_cred_verifier = gssd_cred_verifier;

887     arg.input_token_buffer.GSS_BUFFER_T_len =
888         (uint_t)(input_token != GSS_C_NO_BUFFER ?
889             input_token->length : 0);
890     arg.input_token_buffer.GSS_BUFFER_T_val =
891         (char *)(input_token != GSS_C_NO_BUFFER ?
892             input_token->value : 0);

894     if (input_chan_bindings != GSS_C_NO_CHANNEL_BINDINGS) {
895         arg.input_chan_bindings.present = YES;
896         arg.input_chan_bindings.initiator_addrtype =
897             input_chan_bindings->initiator_addrtype;
898         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_len =
899             (uint_t)input_chan_bindings->initiator_address.length;
900         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_val =
901             (void *) input_chan_bindings->initiator_address.value;
902         arg.input_chan_bindings.acceptor_addrtype =
903             input_chan_bindings->acceptor_addrtype;
904         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_len =
905             (uint_t)input_chan_bindings->acceptor_address.length;
906         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_val =
907             (void *) input_chan_bindings->acceptor_address.value;
908         arg.input_chan_bindings.application_data.GSS_BUFFER_T_len =
909             (uint_t)input_chan_bindings->application_data.length;
910         arg.input_chan_bindings.application_data.GSS_BUFFER_T_val =
911             (void *) input_chan_bindings->application_data.value;
912     } else {
913         arg.input_chan_bindings.present = NO;
914         arg.input_chan_bindings.initiator_addrtype = 0;
915         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_len = 0;
916         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_val = 0;
917         arg.input_chan_bindings.acceptor_addrtype = 0;
918         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_len = 0;
919         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_val = 0;
920         arg.input_chan_bindings.application_data.GSS_BUFFER_T_len = 0;
921         arg.input_chan_bindings.application_data.GSS_BUFFER_T_val = 0;
922     }

924     /* set the output parameters to empty values... */
925     if (minor_status != NULL)
926         *minor_status = DEFAULT_MINOR_STAT;
927     *minor_status = 0xffffffff;
928     if (src_name != NULL) {
929         src_name->length = 0;
930         src_name->value = NULL;
931     }
932     if (mech_type != NULL)
933         *mech_type = NULL;
934     if (output_token != NULL)
935         output_token->length = 0;
936     if (ret_flags != NULL)
937         *ret_flags = 0;
938     if (time_rec != NULL)
939         *time_rec = 0;

```

```

939     if (delegated_cred_handle != NULL)
940         *delegated_cred_handle = NULL;

942     /* call the remote procedure */
943     memset(&res, 0, sizeof (res));
944     if (gss_accept_sec_context_l(&arg, &res, clnt) != RPC_SUCCESS) {
945         return (GSS_S_FAILURE);
946     }

951     if (res.status == (OM_uint32) GSS_S_COMPLETE ||
952         res.status == (OM_uint32) GSS_S_CONTINUE_NEEDED) {
948     /*
949     * We could return from a GSS error here and need to return both the
950     * minor_status and output_token, back to the caller if applicable.
954     * when gss returns CONTINUE_NEEDED we can only
955     * use the context, minor, and output token
956     * parameters.
951     */
952     if (minor_status != NULL)
953         *minor_status = res.minor_status;
954     /*LINTED*/
955     *context_handle = *((gssd_ctx_id_t *)
956         res.context_handle.GSS_CTX_ID_T_val);
957     *gssd_context_verifier = res.gssd_context_verifier;

955     if (output_token != NULL && res.output_token.GSS_BUFFER_T_val != NULL) {
963         if (output_token != NULL) {
956             output_token->length =
957                 res.output_token.GSS_BUFFER_T_len;
958             output_token->value =
959                 (void *) res.output_token.GSS_BUFFER_T_val;
960             res.output_token.GSS_BUFFER_T_val = 0;
961             res.output_token.GSS_BUFFER_T_len = 0;
962         }

964     if (res.status == (OM_uint32) GSS_S_COMPLETE ||
965         res.status == (OM_uint32) GSS_S_CONTINUE_NEEDED) {
966     /*
967     * when gss returns CONTINUE_NEEDED we can only
968     * use the context parameter.
969     */
970     /*LINTED*/
971     *context_handle = *((gssd_ctx_id_t *)
972         res.context_handle.GSS_CTX_ID_T_val);
973     *gssd_context_verifier = res.gssd_context_verifier;
972     if (minor_status != NULL)
973         *minor_status = res.minor_status;

975     /* the other parameters are ready on for COMPLETE */
976     if (res.status == GSS_S_COMPLETE)
977     {

978     /*
979     * The src_name is in external format.
980     */
981     if (src_name != NULL) {
982         src_name->length = res.src_name.GSS_BUFFER_T_len;
983         src_name->value = res.src_name.GSS_BUFFER_T_val;
984         res.src_name.GSS_BUFFER_T_val = NULL;
985         res.src_name.GSS_BUFFER_T_len = 0;
986     }
987     /*
988     * move mech type returned to mech_type
989     * for gss_import_name_for_mech()
990     */

```

```

991     if (mech_type != NULL) {
992         *mech_type =
993             (gss_OID) MALLOC(sizeof (gss_OID_desc));
994         (*mech_type)->length =
995             (OM_UINT32) res.mech_type.GSS_OID_len;
996         (*mech_type)->elements =
997             (void *) MALLOC((*mech_type)->length);
998         memcpy((*mech_type)->elements,
999             res.mech_type.GSS_OID_val,
1000             (*mech_type)->length);
1001     }

1003     if (ret_flags != NULL)
1004         *ret_flags = res.ret_flags;

1006     if (time_rec != NULL)
1007         *time_rec = res.time_rec;

1009     if ((delegated_cred_handle != NULL) &&
1010         (res.delegated_cred_handle.GSS_CRED_ID_T_len
1011             != 0)) {
1012         kcred = KGSS_CRED_ALLOC();
1013         /*LINTED*/
1014         kcred->gssd_cred = *((gssd_cred_id_t *)
1015             res.delegated_cred_handle.GSS_CRED_ID_T_val);
1016         kcred->gssd_cred_verifier =
1017             res.gssd_context_verifier;
1018         *delegated_cred_handle = (gss_cred_id_t)kcred;
1019     }
1020     } /* res.status == GSS_S_COMPLETE */
1021     } /* res.status == GSS_S_COMPLETE or GSS_S_CONTINUE_NEEDED */

1024     /*
1025     * free the memory allocated for the results and return with the status
1026     * received in the rpc call
1027     */

1029     clnt_freeres(clnt, xdr_gss_accept_sec_context_res, (caddr_t)&res);
1030     return (res.status);
1031 }

    unchanged portion omitted

1085 OM_uint32
1086 kgss_process_context_token(minor_status,
1087     context_handle,
1088     token_buffer,
1089     uid)
1090     OM_uint32 *minor_status;
1091     gssd_ctx_id_t context_handle;
1092     gss_buffer_t token_buffer;
1093     uid_t uid;
1094 {
1095     OM_uint32 gssd_context_verifier;

1097     gssd_process_context_token_arg arg;
1098     gssd_process_context_token_res res;

1100     gssd_context_verifier = KGSS_CTX_TO_GSSD_CTXV(context_handle);

1102     /* get the client handle to GSSD */

1104     if ((clnt = getgssd_handle()) == NULL) {
1105         clnt_pcreateerror(server);
1106         return (GSS_S_FAILURE);
1107     }

```

```

1109      /* copy the procedure arguments into the rpc arg parameter */
1110      arg.uid = (OM_uint32) uid;

1112      arg.context_handle.GSS_CTX_ID_T_len = (uint_t)sizeof (gss_ctx_id_t);
1113      arg.context_handle.GSS_CTX_ID_T_val = (char *)&context_handle;
1114      arg.gssd_context_verifier = gssd_context_verifier;
1115      arg.token_buffer.GSS_BUFFER_T_len = (uint_t)token_buffer;
1116      arg.token_buffer.GSS_BUFFER_T_val = (char *)token_buffer->value;

1118      /* call the remote procedure */

1120      memset(&res, 0, sizeof (res));
1121      if (gss_process_context_token_1(&arg, &res, clnt) != RPC_SUCCESS) {

1123      /*
1124      * if the RPC call times out, null out all return arguments,
1125      * set minor_status to its maximum value, and return GSS_S_FAILURE
1126      */

1128          if (minor_status != NULL)
1129              *minor_status = DEFAULT_MINOR_STAT;
1130              *minor_status = 0xffffffff;

1131          return (GSS_S_FAILURE);
1132      }

1134      /* copy the rpc results into the return arguments */

1136      if (minor_status != NULL)
1137          *minor_status = res.minor_status;

1139      /* return with status returned in rpc call */

1141      return (res.status);
1142 }

1144 OM_uint32
1145 kgss_delete_sec_context_wrapped(minor_status,
1146                                context_handle,
1147                                gssd_context_verifier,
1148                                output_token)
1149     OM_uint32 *minor_status;
1150     gssd_ctx_id_t *context_handle;
1151     OM_uint32 gssd_context_verifier;
1152     gss_buffer_t output_token;
1153 {
1154     gss_delete_sec_context_arg arg;
1155     gss_delete_sec_context_res res;

1158     /* get the client handle to GSSD */
1159     if ((clnt = getgssd_handle()) == NULL) {
1160         clnt_pcreateerror(server);
1161         return (GSS_S_FAILURE);
1162     }

1164     /* copy the procedure arguments into the rpc arg parameter */

1166     arg.context_handle.GSS_CTX_ID_T_len =
1167         *context_handle == (OM_uint32) GSS_C_NO_CONTEXT ? 0 :
1168         (uint_t)sizeof (OM_uint32);
1169     arg.context_handle.GSS_CTX_ID_T_val = (char *)context_handle;

1171     arg.gssd_context_verifier = gssd_context_verifier;

```

```

1173      /* call the remote procedure */

1175      memset(&res, 0, sizeof (res));
1176      if (gss_delete_sec_context_1(&arg, &res, clnt) != RPC_SUCCESS) {

1178          /*
1179          * if the RPC call times out, null out all return arguments,
1180          * set minor_status to its max value, and return GSS_S_FAILURE
1181          */

1183          if (minor_status != NULL)
1184              *minor_status = DEFAULT_MINOR_STAT;
1185              *minor_status = 0xffffffff;
1186          if (context_handle != NULL)
1187              *context_handle = NULL;
1188          if (output_token != NULL)
1189              output_token->length = 0;

1190          return (GSS_S_FAILURE);
1191      }

1193      /* copy the rpc results into the return arguments */

1195      if (minor_status != NULL)
1196          *minor_status = res.minor_status;

1198      if (res.context_handle.GSS_CTX_ID_T_len == 0)
1199          *context_handle = NULL;
1200      else
1201          /*LINTED*/
1202          *context_handle = *((gssd_ctx_id_t *)
1203                          res.context_handle.GSS_CTX_ID_T_val);

1205      if (output_token != NULL && res.output_token.GSS_BUFFER_T_val != NULL) {
1206          if (output_token != NULL) {
1207              output_token->length = res.output_token.GSS_BUFFER_T_len;
1208              output_token->value = res.output_token.GSS_BUFFER_T_val;
1209              res.output_token.GSS_BUFFER_T_len = 0;
1210              res.output_token.GSS_BUFFER_T_val = NULL;
1211          }

1212          /*
1213          * free the memory allocated for the results and return with the status
1214          * received in the rpc call
1215          */

1217          clnt_freeres(clnt, xdr_gss_delete_sec_context_res, (caddr_t)&res);
1218          return (res.status);
1219      }

1220      unchanged_portion_omitted

1264     OM_uint32
1265     kgss_sign_wrapped(minor_status,
1266                      context_handle,
1267                      qop_req,
1268                      message_buffer,
1269                      msg_token,
1270                      gssd_context_verifier)
1271         OM_uint32 *minor_status;
1272         gssd_ctx_id_t context_handle;
1273         OM_uint32 gssd_context_verifier;
1274         int qop_req;
1275         gss_buffer_t message_buffer;
1276         gss_buffer_t msg_token;
1277 {

```

```

1279     gss_sign_arg arg;
1280     gss_sign_res res;

1282     /* get the client handle to GSSD */

1284     if ((clnt = getgssd_handle()) == NULL) {
1285         clnt_pcreateerror(server);
1286         return (GSS_S_FAILURE);
1287     }

1289     /* copy the procedure arguments into the rpc arg parameter */

1292     arg.context_handle.GSS_CTX_ID_T_len = (uint_t)sizeof (gssd_ctx_id_t);
1293     arg.context_handle.GSS_CTX_ID_T_val = (char *)&context_handle;
1294     arg.gssd_context_verifier = gssd_context_verifier;

1296     arg.qop_req = qop_req;
1297     arg.message_buffer.GSS_BUFFER_T_len = (uint_t)message_buffer->length;
1298     arg.message_buffer.GSS_BUFFER_T_val = (char *)message_buffer->value;

1300     /* call the remote procedure */

1302     memset(&res, 0, sizeof (res));
1303     if (gss_sign_1(&arg, &res, clnt) != RPC_SUCCESS) {

1305         /*
1306          * if the RPC call times out, null out all return arguments,
1307          * set minor_status to its maximum value, and return GSS_S_FAILURE
1308          */

1310         if (minor_status != NULL)
1311             *minor_status = DEFAULT_MINOR_STAT;
1312             *minor_status = 0xffffffff;
1312         if (msg_token != NULL)
1313             msg_token->length = 0;

1315         return (GSS_S_FAILURE);
1316     }

1318     /* copy the rpc results into the return arguments */

1320     if (minor_status != NULL)
1321         *minor_status = res.minor_status;

1323     if (msg_token != NULL) {
1324         msg_token->length = res.msg_token.GSS_BUFFER_T_len;
1325         msg_token->value = (void *) MALLOC(msg_token->length);
1326         memcpy(msg_token->value, res.msg_token.GSS_BUFFER_T_val,
1327             msg_token->length);
1328     }

1330     /*
1331      * free the memory allocated for the results and return with the status
1332      * received in the rpc call
1333      */

1335     clnt_freeres(clnt, xdr_gss_sign_res, (caddr_t)&res);
1336     return (res.status);
1337 }

```

unchanged portion omitted

```

1355 OM_uint32
1356 kgss_verify_wrapped(
1357     minor_status,
1358     context_handle,

```

```

1359     message_buffer,
1360     token_buffer,
1361     qop_state,
1362     gssd_context_verifier)
1363     OM_uint32 *minor_status;
1364     gssd_ctx_id_t context_handle;
1365     OM_uint32 gssd_context_verifier;
1366     gss_buffer_t message_buffer;
1367     gss_buffer_t token_buffer;
1368     int *qop_state;
1369 {
1370     gss_verify_arg arg;
1371     gss_verify_res res;

1373     /* get the client handle to GSSD */

1375     if ((clnt = getgssd_handle()) == NULL) {
1376         clnt_pcreateerror(server);
1377         return (GSS_S_FAILURE);
1378     }

1380     /* copy the procedure arguments into the rpc arg parameter */

1382     arg.context_handle.GSS_CTX_ID_T_len = (uint_t)sizeof (gssd_ctx_id_t);
1383     arg.context_handle.GSS_CTX_ID_T_val = (char *)&context_handle;

1385     arg.gssd_context_verifier = gssd_context_verifier;

1387     arg.message_buffer.GSS_BUFFER_T_len = (uint_t)message_buffer->length;
1388     arg.message_buffer.GSS_BUFFER_T_val = (char *)message_buffer->value;

1390     arg.token_buffer.GSS_BUFFER_T_len = (uint_t)token_buffer->length;
1391     arg.token_buffer.GSS_BUFFER_T_val = (char *)token_buffer->value;

1393     /* call the remote procedure */

1395     memset(&res, 0, sizeof (res));
1396     if (gss_verify_1(&arg, &res, clnt) != RPC_SUCCESS) {

1398         /*
1399          * if the RPC call times out, null out all return arguments,
1400          * set minor_status to its maximum value, and return GSS_S_FAILURE
1401          */

1403         if (minor_status != NULL)
1404             *minor_status = DEFAULT_MINOR_STAT;
1405             *minor_status = 0xffffffff;
1405         if (qop_state != NULL)
1406             *qop_state = 0;

1408         return (GSS_S_FAILURE);
1409     }

1411     /* copy the rpc results into the return arguments */

1413     if (minor_status != NULL)
1414         *minor_status = res.minor_status;

1416     if (qop_state != NULL)
1417         *qop_state = res.qop_state;

1419     /* return with status returned in rpc call */

1421     return (res.status);
1422 }

```

unchanged portion omitted

```

1440 /* EXPORT DELETE START */

1442 OM_uint32
1443 kgss_seal_wrapped(
1444     minor_status,
1445     context_handle,
1446     conf_req_flag,
1447     qop_req,
1448     input_message_buffer,
1449     conf_state,
1450     output_message_buffer,
1451     gssd_context_verifier)

1453     OM_uint32 *minor_status;
1454     gssd_ctx_id_t context_handle;
1455     OM_uint32 gssd_context_verifier;
1456     int conf_req_flag;
1457     int qop_req;
1458     gss_buffer_t input_message_buffer;
1459     int *conf_state;
1460     gss_buffer_t output_message_buffer;
1461 {
1462     gss_seal_arg arg;
1463     gss_seal_res res;

1465     /* get the client handle to GSSD */

1467     if ((clnt = getgssd_handle()) == NULL) {
1468         clnt_pcreateerror(server);
1469         return (GSS_S_FAILURE);
1470     }

1472     /* copy the procedure arguments into the rpc arg parameter */

1475     arg.context_handle.GSS_CTX_ID_T_len = (uint_t)sizeof (gssd_ctx_id_t);
1476     arg.context_handle.GSS_CTX_ID_T_val = (char *)&context_handle;
1477     arg.gssd_context_verifier = gssd_context_verifier;

1479     arg.conf_req_flag = conf_req_flag;

1481     arg.qop_req = qop_req;

1483     arg.input_message_buffer.GSS_BUFFER_T_len =
1484         (uint_t)input_message_buffer->length;

1486     arg.input_message_buffer.GSS_BUFFER_T_val =
1487         (char *)input_message_buffer->value;

1489     /* call the remote procedure */

1491     memset(&res, 0, sizeof (res));
1492     if (gss_seal_1(&arg, &res, clnt) != RPC_SUCCESS) {

1494     /*
1495     * if the RPC call times out, null out all return arguments,
1496     * set minor_status to its maximum value, and return GSS_S_FAILURE
1497     */

1499         if (minor_status != NULL)
1500             *minor_status = DEFAULT_MINOR_STAT;
1501         *minor_status = 0xffffffff;
1501         if (conf_state != NULL)
1502             *conf_state = 0;

```

```

1503         if (output_message_buffer != NULL)
1504             output_message_buffer->length = 0;

1506         return (GSS_S_FAILURE);
1507     }

1509     /* copy the rpc results into the return arguments */

1511     if (minor_status != NULL)
1512         *minor_status = res.minor_status;

1514     if (conf_state != NULL)
1515         *conf_state = res.conf_state;

1517     if (output_message_buffer != NULL) {
1518         output_message_buffer->length =
1519             res.output_message_buffer.GSS_BUFFER_T_len;

1521         output_message_buffer->value =
1522             (void *) MALLOC(output_message_buffer->length);
1523         memcpy(output_message_buffer->value,
1524             res.output_message_buffer.GSS_BUFFER_T_val,
1525             output_message_buffer->length);
1526     }

1528     /*
1529     * free the memory allocated for the results and return with the status
1530     * received in the rpc call
1531     */

1533     clnt_freeres(clnt, xdr_gss_seal_res, (caddr_t)&res);
1534     return (res.status);
1535 }

_____unchanged_portion_omitted_____

1556 OM_uint32
1557 kgss_unseal_wrapped(minor_status,
1558     context_handle,
1559     input_message_buffer,
1560     output_message_buffer,
1561     conf_state,
1562     qop_state,
1563     gssd_context_verifier)
1564     OM_uint32 *minor_status;
1565     gssd_ctx_id_t context_handle;
1566     OM_uint32 gssd_context_verifier;
1567     gss_buffer_t input_message_buffer;
1568     gss_buffer_t output_message_buffer;
1569     int *conf_state;
1570     int *qop_state;
1571 {
1572     gss_unseal_arg arg;
1573     gss_unseal_res res;

1575     /* get the client handle to GSSD */

1577     if ((clnt = getgssd_handle()) == NULL) {
1578         clnt_pcreateerror(server);
1579         return (GSS_S_FAILURE);
1580     }

1582     /* copy the procedure arguments into the rpc arg parameter */

1585     arg.context_handle.GSS_CTX_ID_T_len = (uint_t)sizeof (gssd_ctx_id_t);
1586     arg.context_handle.GSS_CTX_ID_T_val = (char *)&context_handle;

```

```

1587     arg.gssd_context_verifier = gssd_context_verifier;
1589     arg.input_message_buffer.GSS_BUFFER_T_len =
1590         (uint_t)input_message_buffer->length;
1592     arg.input_message_buffer.GSS_BUFFER_T_val =
1593         (char *)input_message_buffer->value;
1595 /* call the remote procedure */
1597     memset(&res, 0, sizeof (res));
1598     if (gss_unseal_l(&arg, &res, clnt) != RPC_SUCCESS) {
1600         /*
1601          * if the RPC call times out, null out all return arguments,
1602          * set minor_status to its maximum value, and return GSS_S_FAILURE
1603          */
1605         if (minor_status != NULL)
1606             *minor_status = DEFAULT_MINOR_STAT;
1607         *minor_status = 0xffffffff;
1608         if (output_message_buffer != NULL)
1609             output_message_buffer->length = 0;
1610         if (conf_state != NULL)
1611             *conf_state = 0;
1612         if (qop_state != NULL)
1613             *qop_state = 0;
1614     }
1615     return (GSS_S_FAILURE);
1617 /* copy the rpc results into the return arguments */
1619     if (minor_status != NULL)
1620         *minor_status = res.minor_status;
1622     if (output_message_buffer != NULL) {
1623         output_message_buffer->length =
1624             res.output_message_buffer.GSS_BUFFER_T_len;
1626         output_message_buffer->value =
1627             (void *) MALLOC(output_message_buffer->length);
1628         memcpy(output_message_buffer->value,
1629             res.output_message_buffer.GSS_BUFFER_T_val,
1630             output_message_buffer->length);
1631     }
1633     if (conf_state != NULL)
1634         *conf_state = res.conf_state;
1636     if (qop_state != NULL)
1637         *qop_state = res.qop_state;
1639     /*
1640     * free the memory allocated for the results and return with the status
1641     * received in the rpc call
1642     */
1644     clnt_freeres(clnt, xdr_gss_unseal_res, (caddr_t)&res);
1645     return (res.status);
1646 }
1647
1648 unchanged_portion_omitted
1655 /* EXPORT DELETE END */
1667 OM_uint32

```

```

1668 kgss_display_status(minor_status,
1669                     status_value,
1670                     status_type,
1671                     mech_type,
1672                     message_context,
1673                     status_string,
1674                     uid)
1675     OM_uint32 *minor_status;
1676     OM_uint32 status_value;
1677     int status_type;
1678     gss_OID mech_type;
1679     int *message_context;
1680     gss_buffer_t status_string;
1681     uid_t uid;
1682 {
1683     gss_display_status_arg arg;
1684     gss_display_status_res res;
1686     /* get the client handle to GSSD */
1688     if ((clnt = getgssd_handle()) == NULL) {
1689         clnt_pcreateerror(server);
1690         return (GSS_S_FAILURE);
1691     }
1693     /* copy the procedure arguments into the rpc arg parameter */
1695     arg.uid = (OM_uint32) uid;
1697     arg.status_value = status_value;
1698     arg.status_type = status_type;
1700     arg.mech_type.GSS_OID_len = (uint_t)(mech_type != GSS_C_NULL_OID ?
1701                                         mech_type->length : 0);
1702     arg.mech_type.GSS_OID_val = (char *) (mech_type != GSS_C_NULL_OID ?
1703                                         mech_type->elements : 0);
1705     arg.message_context = *message_context;
1707     /* call the remote procedure */
1709     if (message_context != NULL)
1710         *message_context = 0;
1711     if (status_string != NULL) {
1712         status_string->length = 0;
1713         status_string->value = NULL;
1714     }
1716     memset(&res, 0, sizeof (res));
1717     if (gss_display_status_l(&arg, &res, clnt) != RPC_SUCCESS) {
1719         /*
1720         * if the RPC call times out, null out all return arguments,
1721         * set minor_status to its maximum value, and return GSS_S_FAILURE
1722         */
1724         if (minor_status != NULL)
1725             *minor_status = DEFAULT_MINOR_STAT;
1726         *minor_status = 0xffffffff;
1727     }
1728     return (GSS_S_FAILURE);
1730     if (minor_status != NULL)
1731         *minor_status = res.minor_status;

```

```

1733      /* now process the results and pass them back to the caller */
1735      if (res.status == GSS_S_COMPLETE) {
1736          if (minor_status != NULL)
1737              *minor_status = res.minor_status;
1738          if (message_context != NULL)
1739              *message_context = res.message_context;
1740          if (status_string != NULL) {
1741              status_string->length =
1742                  (size_t)res.status_string.GSS_BUFFER_T_len;
1743              status_string->value =
1744                  (void *)MALLOC(status_string->length);
1745              memcpy(status_string->value,
1746                     res.status_string.GSS_BUFFER_T_val,
1747                     status_string->length);
1748          }
1749      }
1750      clnt_freeres(clnt, xdr_gss_display_status_res, (caddr_t)&res);
1751      return (res.status);
1752  }
1753  /*ARGSUSED*/
1754  OM_uint32
1755  kgss_indicate_mechs(minor_status,
1756                     mech_set,
1757                     uid)
1758  OM_uint32 *minor_status;
1759  gss_OID_set *mech_set;
1760  uid_t uid;
1761  {
1762      void *arg;
1763      gss_indicate_mechs_res res;
1764      int i;
1765
1766      /* get the client handle to GSSD */
1767
1768      if ((clnt = getgssd_handle()) == NULL) {
1769          clnt_pcreateerror(server);
1770          return (GSS_S_FAILURE);
1771      }
1772
1773      memset(&res, 0, sizeof (res));
1774      if (gss_indicate_mechs_1(&arg, &res, clnt) != RPC_SUCCESS) {
1775
1776          /*
1777           * if the RPC call times out, null out all return arguments,
1778           * set minor_status to its maximum value, and return GSS_S_FAILURE
1779           */
1780
1781          if (minor_status != NULL)
1782              *minor_status = DEFAULT_MINOR_STAT;
1783          *minor_status = 0xffffffff;
1784          if (mech_set != NULL)
1785              *mech_set = NULL;
1786
1787          return (GSS_S_FAILURE);
1788      }
1789
1790      /* copy the rpc results into the return arguments */
1791
1792      if (minor_status != NULL)
1793          *minor_status = res.minor_status;
1794
1795      if (mech_set != NULL) {
1796          *mech_set = (gss_OID_set) MALLOC(sizeof (gss_OID_set_desc));

```

```

1796          (*mech_set)->count = res.mech_set.GSS_OID_SET_len;
1797          (*mech_set)->elements = (void *)
1798              MALLOC ((*mech_set)->count * sizeof (gss_OID_desc));
1799          for (i = 0; i < (*mech_set)->count; i++) {
1800              (*mech_set)->elements[i].length =
1801                  res.mech_set.GSS_OID_SET_val[i].GSS_OID_len;
1802              (*mech_set)->elements[i].elements = (void *)
1803                  MALLOC ((*mech_set)->elements[i].length);
1804              memcpy ((*mech_set)->elements[i].elements,
1805                     res.mech_set.GSS_OID_SET_val[i].GSS_OID_val,
1806                     (*mech_set)->elements[i].length);
1807          }
1808      }
1809
1810      /*
1811       * free the memory allocated for the results and return with the status
1812       * received in the rpc call
1813       */
1814
1815      clnt_freeres(clnt, xdr_gss_indicate_mechs_res, (caddr_t)&res);
1816      return (res.status);
1817  }
1818
1819  OM_uint32
1820  kgss_inquire_cred_wrapped(minor_status,
1821                            cred_handle,
1822                            gssd_cred_verifier,
1823                            name,
1824                            lifetime,
1825                            cred_usage,
1826                            mechanisms,
1827                            uid)
1828  OM_uint32 *minor_status;
1829  gssd_cred_id_t cred_handle;
1830  OM_uint32 gssd_cred_verifier;
1831  gss_name_t *name;
1832  OM_uint32 *lifetime;
1833  int *cred_usage;
1834  gss_OID_set *mechanisms;
1835  uid_t uid;
1836  {
1837      OM_uint32 minor_status_temp;
1838      gss_buffer_desc external_name;
1839      gss_OID name_type;
1840      int i;
1841
1842      gss_inquire_cred_arg arg;
1843      gss_inquire_cred_res res;
1844
1845      /* get the client handle to GSSD */
1846
1847      if ((clnt = getgssd_handle()) == NULL) {
1848          clnt_pcreateerror(server);
1849          return (GSS_S_FAILURE);
1850      }
1851
1852      /* copy the procedure arguments into the rpc arg parameter */
1853
1854      arg.uid = (OM_uint32) uid;
1855
1856      arg.cred_handle.GSS_CRED_ID_T_len =
1857          cred_handle == (gssd_cred_id_t)GSS_C_NO_CREDENTIAL ?
1858          0 : (uint_t)sizeof (gssd_cred_id_t);
1859      arg.cred_handle.GSS_CRED_ID_T_val = (char *)&cred_handle;

```

```

1862     arg.gssd_cred_verifier = gssd_cred_verifier;
1864     /* call the remote procedure */
1866     memset(&res, 0, sizeof (res));
1867     if (gss_inquire_cred_l(&arg, &res, clnt) != RPC_SUCCESS) {
1869     /*
1870     * if the RPC call times out, null out all return arguments,
1871     * set minor_status to its maximum value, and return GSS_S_FAILURE
1872     */
1874         if (minor_status != NULL)
1875             *minor_status = DEFAULT_MINOR_STAT;
1876             *minor_status = 0xffffffff;
1877         if (name != NULL)
1878             *name = NULL;
1879         if (lifetime != NULL)
1880             *lifetime = 0;
1881         if (cred_usage != NULL)
1882             *cred_usage = 0;
1883         if (mechanisms != NULL)
1884             *mechanisms = NULL;
1885     }
1886     return (GSS_S_FAILURE);
1888     /* copy the rpc results into the return arguments */
1890     if (minor_status != NULL)
1891         *minor_status = res.minor_status;
1893     /* convert name from external to internal format */
1895     if (name != NULL) {
1896         external_name.length = res.name.GSS_BUFFER_T_len;
1897         external_name.value = res.name.GSS_BUFFER_T_val;
1899         /*
1900         * we have to allocate a name_type descriptor and
1901         * elements storage, since gss_import_name() only
1902         * stores a pointer to the name_type info in the
1903         * union_name struct
1904         */
1906         name_type = (gss_OID) MALLOC(sizeof (gss_OID_desc));
1908         name_type->length = res.name_type.GSS_OID_len;
1909         name_type->elements = (void *) MALLOC(name_type->length);
1910         memcpy(name_type->elements, res.name_type.GSS_OID_val,
1911             name_type->length);
1913         if (gss_import_name(&minor_status_temp, &external_name,
1914             name_type, name) != GSS_S_COMPLETE) {
1916             *minor_status = (OM_uint32) minor_status_temp;
1917             gss_release_buffer(&minor_status_temp, &external_name);
1919             clnt_freeres(clnt, xdr_gss_inquire_cred_res,
1920                 (caddr_t)&res);
1921             return ((OM_uint32) GSS_S_FAILURE);
1922         }
1923     }
1925     if (lifetime != NULL)
1926         *lifetime = res.lifetime;

```

```

1928     if (cred_usage != NULL)
1929         *cred_usage = res.cred_usage;
1931     if (mechanisms != NULL) {
1932         *mechanisms =
1933             (gss_OID_set) MALLOC(sizeof (gss_OID_set_desc));
1934         if (res.mechanisms.GSS_OID_SET_len != 0) {
1935             (*mechanisms)->count =
1936                 (int)res.mechanisms.GSS_OID_SET_len;
1937             (*mechanisms)->elements = (gss_OID)
1938                 MALLOC(sizeof (gss_OID) * (*mechanisms)->count);
1940             for (i = 0; i < (*mechanisms)->count; i++) {
1941                 (*mechanisms)->elements[i].length = (OM_uint32)
1942                     res.mechanisms.GSS_OID_SET_val[i].GSS_OID_len;
1943                 (*mechanisms)->elements[i].elements = (void *)
1944                     MALLOC((*mechanisms)->elements[i].length);
1945                 memcpy((*mechanisms)->elements[i].elements,
1946                     res.mechanisms.GSS_OID_SET_val[i].GSS_OID_val,
1947                     (*mechanisms)->elements[i].length);
1948             }
1949             } else
1950                 (*mechanisms)->count = 0;
1951     }
1953     /*
1954     * free the memory allocated for the results and return with the status
1955     * received in the rpc call
1956     */
1958     clnt_freeres(clnt, xdr_gss_inquire_cred_res, (caddr_t)&res);
1959     return (res.status);
1960 }

```

unchanged portion omitted

```

1992 OM_uint32
1993 kgss_inquire_cred_by_mech_wrapped(minor_status,
1994     cred_handle,
1995     gssd_cred_verifier,
1996     mech_type,
1997     uid)
1998     OM_uint32 *minor_status;
1999     gssd_cred_id_t cred_handle;
2000     OM_uint32 gssd_cred_verifier;
2001     gss_OID mech_type;
2002     uid_t uid;
2003 {
2004     OM_uint32 minor_status_temp;
2006     gss_inquire_cred_by_mech_arg arg;
2007     gss_inquire_cred_by_mech_res res;
2009     /* get the client handle to GSSD */
2011     if ((clnt = getgssd_handle()) == NULL) {
2012         clnt_pcreateerror(server);
2013         return (GSS_S_FAILURE);
2014     }
2017     /* copy the procedure arguments into the rpc arg parameter */
2019     arg.uid = (OM_uint32) uid;

```

```

2021     arg.cred_handle.GSS_CRED_ID_T_len =
2022         cred_handle == (gssd_cred_id_t)GSS_C_NO_CREDENTIAL ?
2023         0 : (uint_t)sizeof (gssd_cred_id_t);
2024     arg.cred_handle.GSS_CRED_ID_T_val = (char *)&cred_handle;
2025     arg.gssd_cred_verifier = gssd_cred_verifier;

2027     arg.mech_type.GSS_OID_len =
2028         (uint_t)(mech_type != GSS_C_NULL_OID ?
2029             mech_type->length : 0);
2030     arg.mech_type.GSS_OID_val =
2031         (char *)(mech_type != GSS_C_NULL_OID ?
2032             mech_type->elements : 0);
2033     /* call the remote procedure */

2035     memset(&res, 0, sizeof (res));
2036     if (gss_inquire_cred_by_mech_l(&arg, &res, clnt) != RPC_SUCCESS) {

2038     /*
2039     * if the RPC call times out, null out all return arguments,
2040     * set minor_status to its maximum value, and return GSS_S_FAILURE
2041     */

2043         if (minor_status != NULL)
2044             *minor_status = DEFAULT_MINOR_STAT;
2045         *minor_status = 0xffffffff;
2046     }

2048     /* copy the rpc results into the return arguments */

2050     if (minor_status != NULL)
2051         *minor_status = res.minor_status;

2053     /* convert name from external to internal format */

2055     /*
2056     * free the memory allocated for the results and return with the status
2057     * received in the rpc call
2058     */

2060     clnt_freeres(clnt, xdr_gss_inquire_cred_by_mech_res, (caddr_t)&res);
2061     return (res.status);
2062 }
    unchanged portion omitted

2291 OM_uint32
2292 kgss_export_sec_context_wrapped(minor_status,
2293     context_handle,
2294     output_token,
2295     gssd_context_verifier)
2296     OM_uint32 *minor_status;
2297     gssd_ctx_id_t *context_handle;
2298     gss_buffer_t output_token;
2299     OM_uint32 gssd_context_verifier;
2300 {
2301     CLIENT *clnt;
2302     gss_export_sec_context_arg arg;
2303     gss_export_sec_context_res res;

2306     /* get the client handle to GSSD */

2308     if ((clnt = getgssd_handle()) == NULL) {
2309         clnt_pcreateerror(server);
2310         return (GSS_S_FAILURE);
2311     }

```

```

2313     /* copy the procedure arguments into the rpc arg parameter */

2315     arg.context_handle.GSS_CTX_ID_T_len = (uint_t)sizeof (gssd_ctx_id_t);
2316     arg.context_handle.GSS_CTX_ID_T_val = (char *)context_handle;
2317     arg.gssd_context_verifier = gssd_context_verifier;

2319     /* call the remote procedure */

2321     memset(&res, 0, sizeof (res));
2322     if (gss_export_sec_context_l(&arg, &res, clnt) != RPC_SUCCESS) {

2324     /*
2325     * if the RPC call times out, null out all return arguments, set minor_status
2326     * to its maximum value, and return GSS_S_FAILURE
2327     */

2329         if (minor_status != NULL)
2330             *minor_status = DEFAULT_MINOR_STAT;
2331         if (context_handle != NULL)
2332             *context_handle = NULL;
2333         if (output_token != NULL)
2334             output_token->length = 0;

2336         return (GSS_S_FAILURE);
2337     }

2339     /* copy the rpc results into the return arguments */

2341     if (minor_status != NULL)
2342         *minor_status = res.minor_status;

2344     if (res.context_handle.GSS_CTX_ID_T_len == 0)
2345         *context_handle = NULL;
2346     else
2347         *context_handle =
2348             *((gssd_ctx_id_t *)res.context_handle.GSS_CTX_ID_T_val);

2350     if (output_token != NULL && res.output_token.GSS_BUFFER_T_val != NULL) {
2351         if (output_token != NULL) {
2352             output_token->length = res.output_token.GSS_BUFFER_T_len;
2353             output_token->value =
2354                 (void *) MALLOC(output_token->length);
2355             memcpy(output_token->value,
2356                 res.output_token.GSS_BUFFER_T_val,
2357                 output_token->length);
2358         }

2359     /*
2360     * free the memory allocated for the results and return with the status
2361     * received in the rpc call
2362     */

2364     clnt_freeres(clnt, xdr_gss_export_sec_context_res, (caddr_t)&res);
2365     return (res.status);

2367 }
    unchanged portion omitted

```

new/usr/src/cmd/krb5/kadmin/kclient/kclient.sh

1

```
*****
50214 Thu May  7 01:13:43 2009
new/usr/src/cmd/krb5/kadmin/kclient/kclient.sh
6817447 libgss and various mechs are hiding both the real minor_status and the e
6405422 Solaris acceptors fail in AD-KDC environments when using non-"host" serv
6824434 Unable to accept context establishment initiated by Windows 2000 clients
6787343 kclient's site lookups fail in certain network environments
6692646 kclient should output errors to stderr
6525327 kinit failed when arcfour-hmac-md5-exp was used for the principal's key
6745582 SUNWkdcu missing package dependencies after kclientv2 integration
*****
1 #!/bin/ksh93 -p
2 #
3 # CDDL HEADER START
4 #
5 # The contents of this file are subject to the terms of the
6 # Common Development and Distribution License (the "License").
7 # You may not use this file except in compliance with the License.
8 #
9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 # or http://www.opensolaris.org/os/licensing.
11 # See the License for the specific language governing permissions
12 # and limitations under the License.
13 #
14 # When distributing Covered Code, include this CDDL HEADER in each
15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 # If applicable, add the following below this CDDL HEADER, with the
17 # fields enclosed by brackets "[]" replaced with your own identifying
18 # information: Portions Copyright [yyyy] [name of copyright owner]
19 #
20 # CDDL HEADER END
21 #
22 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24 # Use is subject to license terms.
25 #
26 # This script is used to setup the Kerberos client by
27 # supplying information about the Kerberos realm and kdc.
28 #
29 # The kerberos configuration file (/etc/krb5/krb5.conf) would
30 # be generated and local host's keytab file setup. The script
31 # can also optionally setup the system to do kerberized nfs and
32 # bringover a master krb5.conf copy from a specified location.
33
34 function cleanup {
35     integer ret=$1
36
37     kdestroy -q > $TMP_FILE 2>&1
38     kdestroy -q 1> $TMP_FILE 2>&1
39     rm -r $TMPDIR > /dev/null 2>&1
40
41     exit $1
42     exit $ret
43 }
44
45 unchanged_portion_omitted
46
47 function error_message {
48
49     printf -- "-----\n" >&2
50     printf "$(gettext "Setup FAILED").\n\n" >&2
51     printf -- "-----\n"
52     printf "$(gettext "Setup FAILED").\n\n"
53 }
54
55 cleanup 1
56 }
```

new/usr/src/cmd/krb5/kadmin/kclient/kclient.sh

2

```
55 function check_bin {
56
57     typeset bin=$1
58
59     if [[ ! -x $bin ]]; then
60         printf "$(gettext "Could not access/execute %s").\n" $bin >&2
61         printf "$(gettext "Could not access/execute %s").\n" $bin
62         error_message
63     fi
64 }
65
66 unchanged_portion_omitted
67
68 function writeup_krb5_conf {
69     typeset dh
70
71     printf "\n$(gettext "Setting up %s").\n\n" $KRB5_CONFIG_FILE
72
73     exec 3>$KRB5_CONFIG
74     if [[ $? -ne 0 ]]; then
75         printf "\n$(gettext "Can not write to %s, exiting").\n" $KRB5_CO
76         printf "\n$(gettext "Can not write to %s, exiting").\n" $KRB5_CO
77         error_message
78     fi
79
80     printf "[libdefaults]\n" 1>&3
81     if [[ $no_keytab == yes ]]; then
82         printf "\tverify_ap_req_nofail = false\n" 1>&3
83     fi
84     if [[ $dns_lookup == yes ]]; then
85         printf "\tdns_lookup_realm = on\n" 1>&3
86         if [[ $dnsarg == dns_lookup_kdc ]]; then
87             printf "\tdefault_realm = $realm\n" 1>&3
88             printf "\tdomain_realm\n" 1>&3
89             if [[ -n $kdc_list ]]; then
90                 for kdc in $kdc_list; do
91                     printf "\tkdc = $realm\n" 1>&3
92                 done
93             fi
94             printf "\tFKDC = $realm\n" 1>&3
95             printf "\tclient_machine = $realm\n" 1>&3
96             if [[ -z $short_fqdn ]]; then
97                 printf "\t.$domain = $realm\n\n" 1>&3
98             else
99                 printf "\t.$short_fqdn = $realm\n\n" 1>&3
100            fi
101            if [[ -n $domain_list ]]; then
102                for dh in $domain_list; do
103                    printf "\tdh = $realm\n" 1>&3
104                done
105            fi
106        else
107            if [[ $dnsarg = dns_lookup_realm ]]; then
108                printf "\tdefault_realm = $realm\n" 1>&3
109                printf "\n[realms]\n" 1>&3
110                printf "\t$realm = {\n" 1>&3
111                if [[ -n $kdc_list ]]; then
112                    for kdc in $kdc_list; do
113                        printf "\tkdc = $kdc\n" 1>&3
114                    done
115                else
116                    printf "\tkdc = $KDC\n" 1>&3
117                fi
118                printf "\tadmin_server = $KDC\n" 1>&3
119                if [[ $non_solaris == yes ]]; then
120                    printf "\n\tkpasswd_protocol = SET_CHANGE\n" 1>&3
121                fi
122            fi
123        fi
124    }
125 }
```

```

243         printf "\t}\n\n" 1>&3
244     else
245         printf "\tdefault_realm = $realm\n\n" 1>&3
246     fi
247 fi
248 else
249     printf "\tdefault_realm = $realm\n\n" 1>&3
251     printf "[realms]\n" 1>&3
252     printf "\t$realm = {\n" 1>&3
253     if [[ -n $kdc_list ]]; then
254         for kdc in $kdc_list; do
255             printf "\t\tkdc = $kdc\n" 1>&3
256         done
257     else
258         printf "\t\tkdc = $KDC\n" 1>&3
259     fi
260     printf "\t\tadmin_server = $KDC\n" 1>&3
261     if [[ $non_solaris == yes ]]; then
262         printf "\n\t\tkpasswd_protocol = SET_CHANGE\n" 1>&3
263     fi
264     printf "\t}\n\n" 1>&3
266     printf "[domain_realm]\n" 1>&3
267     if [[ -n $fkdc_list ]]; then
268         for kdc in $fkdc_list; do
269             printf "\t\t$kdc = $realm\n" 1>&3
270         done
271     fi
272     printf "\t\t$FKDC = $realm\n" 1>&3
273     printf "\t\tclient_machine = $realm\n" 1>&3
274     if [[ -z $short_fqdn ]]; then
275         printf "\t\t.$domain = $realm\n\n" 1>&3
276     else
277         printf "\t\t.$short_fqdn = $realm\n\n" 1>&3
278     fi
279     if [[ -n $domain_list ]]; then
280         for dh in $domain_list; do
281             printf "\t\t$dh = $realm\n" 1>&3
282         done
283     fi
284 fi
286     printf "[logging]\n" 1>&3
287     printf "\tdefault = FILE:/var/krb5/kdc.log\n" 1>&3
288     printf "\tkdc = FILE:/var/krb5/kdc.log\n" 1>&3
289     printf "\tkdc_rotate = {\n\t\tperiod = 1d\n\t\tversions = 10\n\t}\n\n" 1
291     printf "[appdefaults]\n" 1>&3
292     printf "\tkinit = {\n\t\trenewable = true\n\t\tforwardable = true\n" 1>&3
293     if [[ $no_keytab == yes ]]; then
294         printf "\t\tno_addresses = true\n" 1>&3
295     fi
296     printf "\t}\n" 1>&3
297 }
    unchanged portion omitted
862 function write_ads_krb5conf {
863     printf "\n${gettext "Setting up %s"}.\n\n" $KRB5_CONFIG_FILE
865     exec 3>$KRB5_CONFIG
866     if [[ $? -ne 0 ]]; then
867         printf "\n${gettext "Can not write to %s, exiting"}.\n" $KRB5_CO
868         printf "\n${gettext "Can not write to %s, exiting"}.\n" $KRB5_CO
868         error_message
869     fi

```

```

871     printf "[libdefaults]\n" 1>&3
872     printf "\tdefault_realm = $realm\n" 1>&3
873     printf "\n[realms]\n" 1>&3
874     printf "\t$realm = {\n" 1>&3
875     for i in ${KDCs[@]}
876     do
877         [[ $i == +([0-9]) ]] && continue
878         printf "\t\tkdc = $i\n" 1>&3
879     done
880     # Defining the same as admin_server. This would cause auth failures
881     # if this was different.
882     printf "\n\t\tkpasswd_server = $KDC\n" 1>&3
883     printf "\n\t\tadmin_server = $KDC\n" 1>&3
884     printf "\n\t\tkpasswd_protocol = SET_CHANGE\n\t}\n" 1>&3
885     printf "\n[domain_realm]\n" 1>&3
886     printf "\t.$dom = $realm\n\n" 1>&3
887     printf "[logging]\n" 1>&3
888     printf "\tdefault = FILE:/var/krb5/kdc.log\n" 1>&3
889     printf "\tkdc = FILE:/var/krb5/kdc.log\n" 1>&3
890     printf "\tkdc_rotate = {\n\t\tperiod = 1d\n\t\tversions = 10\n\t}\n\n" 1
891     printf "[appdefaults]\n" 1>&3
892     printf "\tkinit = {\n\t\trenewable = true\n\t\tforwardable = true\n\t}\n
893 }
895 function getForestName {
896     ldapsearch -R -T -h $dc $ldap_args \
897         -b "" -s base "" schemaNamingContext | \
898         grep ^schemaNamingContext|read j schemaNamingContext
900     if [[ $? -ne 0 ]]; then
901         printf "${gettext "Can't find forest"}.\n" >&2
902         printf "${gettext "Can't find forest"}.\n"
903         error_message
904     fi
905     schemaNamingContext=${schemaNamingContext#CN=Schema,CN=Configuration,}
906     [[ -z $schemaNamingContext ]] && return 1
908     forest=
909     while [[ -n $schemaNamingContext ]]
910     do
911         schemaNamingContext=${schemaNamingContext#DC=}
912         forest=${forest}.${schemaNamingContext%*,*}
913         [[ "$schemaNamingContext" = *,* ]] || break
914         schemaNamingContext=${schemaNamingContext#*,}
915     done
916     forest=${forest#}
917 }
    unchanged portion omitted
941 #
942 # The local variables used to calculate the IP address are of type unsigned
943 # integer (-ui), as this is required to restrict the integer to 32b.
944 # Starting in ksh88, Solaris has incorrectly assumed that -i represents 64b.
945 #
946 function ipAddr2num {
947     typeset OIFS
948     typeset -ui16 num
949     typeset -i16 num byte
950     if [[ "$1" != +([0-9]).+([0-9]).+([0-9]).+([0-9]) ]]
951     then
952         print 0
953         return 0
954     fi

```

```

956     OIFS="$IFS"
957     IFS=.
958     set -- $1
959     IFS="$OIFS"

961     num=$(( ${1}<<24 | ${2}<<16 | ${3}<<8 | ${4} ))

963     print -- $num
964 }

966 #
967 # The local variables used to calculate the IP address are of type unsigned
968 # integer (-ui), as this is required to restrict the integer to 32b.
969 # Starting in ksh88, Solaris has incorrectly assumed that -i represents 64b.
970 #
971 function num2ipAddr {
972     typeset -ui16 num
973     typeset -ui10 a b c d
974     typeset -i16 num
975     typeset -i10 a b c d

976     num=$1
977     a=$((num>>24))
978     b=$((num>>16 & 16#ff))
979     c=$((num>>8 & 16#ff))
980     d=$((num & 16#ff))
981     print -- $a.$b.$c.$d

983 #
984 # The local variables used to calculate the IP address are of type unsigned
985 # integer (-ui), as this is required to restrict the integer to 32b.
986 # Starting in ksh88, Solaris has incorrectly assumed that -i represents 64b.
987 #
988 function netmask2length {
989     typeset -ui16 netmask
990     typeset -i16 netmask
991     typeset -i len

992     netmask=$1
993     len=32
994     while [[ $(($netmask % 2)) -eq 0 ]];
995     do
996         netmask=$((netmask>>1))
997         len=$((len - 1))
998     done
999     print $len
1000 }

1002 #
1003 # The local variables used to calculate the IP address are of type unsigned
1004 # integer (-ui), as this is required to restrict the integer to 32b.
1005 # Starting in ksh88, Solaris has incorrectly assumed that -i represents 64b.
1006 #
1007 function getSubnets {
1008     typeset -ui16 addr netmask
1009     typeset -ui16 classa=16\#ff000000
1010     typeset -i16 addr netmask
1011     typeset -i16 classa=16\#ff000000

1012     ifconfig -a|while read line
1013     do
1014         addr=0
1015         netmask=0
1016         set -- $line

```

```

1016     [[ $1 == inet ]] || continue
1017     while [[ $# -gt 0 ]]
1018     do
1019         case "$1" in
1020             inet) addr=$(ipAddr2num $2); shift;;
1021             netmask) eval netmask=16\#$2; shift;;
1022             *) ;;
1023         esac
1024     done
1025     shift

1027     [[ $addr -eq 0 || $netmask -eq 0 ]] && continue
1028     [[ $(($addr & classa)) -eq 16\#f000000 ]] && continue

1030     print $(num2ipAddr $(($addr & netmask)))/$(netmask2length $netmas
1031     done
1032 }

unchanged_portion_omitted

1118 function compareDomains {
1119     typeset oldDom hspn newDom=$1

1121     # If the client has been previously configured in a different
1122     # realm/domain then we need to prompt the user to see if they wish to
1123     # switch domains.
1124     klist -k 2>&1 | grep @ | read j hspn
1125     [[ -z $hspn ]] && return

1127     oldDom=${hspn#*@}
1128     if [[ $oldDom != $newDom ]]; then
1129         printf "$(gettext "The client is currently configured in a diffe
1130         printf "$(gettext "Currently in the '%s' domain, trying to join
1131         query "$(gettext "Do you want the client to join a new domain")
1132         printf "\n"
1133         if [[ $answer != yes ]]; then
1134             printf "$(gettext "Client will not be joined to the new
1135             printf "$(gettext "Client will not be joined to the new
1136             error_message
1137         fi
1138     fi

1140 function getKDCDC {

1142     getKDC
1143     if [[ -n $kdc ]]; then
1144         KDC=$kdc
1145         dc=$kdc
1146     else
1147         getDC
1148         if [[ -n $dc ]]; then
1149             KDC=$dc
1150         else
1151             printf "$(gettext "Could not find domain controller serv
1152             printf "$(gettext "Could not find domain controller serv
1153             error_message
1154         fi
1155     fi

1157 function join_domain {
1158     typeset -u upcase_nodename
1159     typeset netbios_nodename fqdn

1160     container=Computers
1161     ldap_args="-o authzid= -o mech=gssapi"

```

```

1163     userAccountControlBASE=4096
1165     if [[ -z $ADMIN_PRINC ]]; then
1166         cprinc=Administrator
1167     else
1168         cprinc=$ADMIN_PRINC
1169     fi
1171     if ! discover_domain; then
1172         printf "$(gettext "Can not find realm") '%s'.\n" $realm >&2
1173         printf "$(gettext "Can not find realm") '%s'.\n" $realm
1174         error_message
1175     fi
1176     dom=$domain
1177     realm=$domain
1178     upcase_nodename=$hostname
1179     netbios_nodename="{upcase_nodename}\$"
1180     fqdn=$hostname.$domain
1181     upn=host/${fqdn}
1183     grep=/usr/xpg4/bin/grep
1185     object=$(mktemp -q -t kclient-computer-object.XXXXXX)
1186     if [[ -z $object ]]; then
1187         printf "\n$(gettext "Can not create temporary file, exiting").\n"
1188         error_message
1189     fi
1190     fi
1192     grep=/usr/xpg4/bin/grep
1194     modify_existing=false
1195     recreate=false
1197     DomainDnsZones=$(rev_resolve DomainDnsZones.$dom.)
1198     ForestDnsZones=$(rev_resolve ForestDnsZones.$dom.)
1200     getBaseDN "$container" "$dom"
1202     if [[ -n $KDC ]]; then
1203         dc=$KDC
1204     else
1205         getKDCDC
1206     fi
1208     write_ads_krb5conf
1210     printf "$(gettext "Attempting to join '%s' to the '%s' domain").\n\n" $u
1212     kinit $cprinc@$realm
1213     if [[ $? -ne 0 ]]; then
1214         printf "$(gettext "Could not authenticate %s. Exiting").\n" $cp
1215         printf "$(gettext "Could not authenticate %s. Exiting").\n" $cp
1216         error_message
1217     fi
1218     if getForestName
1219     then
1220         printf "\n$(gettext "Forest name found: %s")\n\n" $forest
1221     else
1222         printf "\n$(gettext "Forest name not found, assuming forest is t
1223     fi
1225     getGC
1226     getSite

```

```

1228     if [[ -z $siteName ]]
1229     then
1230         printf "$(gettext "Site name not found. Local DCs/GCs will not
1231     else
1232         printf "$(gettext "Looking for _local_ KDCs, DCs and global cata
1233         getKDCDC
1234         getGC
1236         write_ads_krb5conf
1237     fi
1239     if [[ ${#GCs} -eq 0 ]]; then
1240         printf "$(gettext "Could not find global catalogs. Exiting").\n
1241         printf "$(gettext "Could not find global catalogs. Exiting").\n
1242         error_message
1243     fi
1244     # Check to see if the client is transitioning between domains.
1245     compareDomains $realm
1247     # Here we check domainFunctionality to see which release:
1248     # 0, 1, 2: Windows 2000, 2003 Interim, 2003 respectively
1249     # 3: Windows 2008
1250     level=0
1251     ldapsearch -R -T -h "$dc" $ldap_args -b "" -s base "" \
1252         domainControllerFunctionality| grep ^domainControllerFunctionality| \
1253         read j level
1254     if [[ $? -ne 0 ]]; then
1255         printf "$(gettext "Search for domain functionality failed, exiti
1256         printf "$(gettext "Search for domain functionality failed, exiti
1257         error_message
1258     fi
1259     # Longhorn and above can't perform an init auth from service
1260     # keys if the realm is included in the UPN. w2k3 and below
1261     # can't perform an init auth when the realm is excluded.
1262     [[ $level -lt 3 ]] && upn="{upn}@${realm}
1263     if ldapsearch -R -T -h "$dc" $ldap_args -b "$baseDN" \
1264         -s sub sAMAccountName="$netbios_nodename" dn > /dev/null 2>&1
1265     then
1266         :
1267     else
1268         printf "$(gettext "Search for node failed, exiting").\n" >&2
1269         printf "$(gettext "Search for node failed, exiting").\n"
1270         error_message
1271     fi
1272     ldapsearch -R -T -h "$dc" $ldap_args -b "$baseDN" -s sub \
1273         sAMAccountName="$netbios_nodename" dn|grep "^dn:"|read j dn
1274     if [[ -z $dn ]]; then
1275         : # modify_existing is already false, which is what we want.
1276     else
1277         printf "$(gettext "Computer account '%s' already exists in the '
1278         query "$(gettext "Do you wish to recreate this computer account"
1279         printf "\n"
1280         if [[ $answer == yes ]]; then
1281             recreate=true
1282         else
1283             modify_existing=true
1284         fi
1285     fi
1287     if [[ $modify_existing == false && -n $dn ]]; then
1288         query "$(gettext "Would you like to delete any sub-object found
1289         if [[ $answer == yes ]]; then

```

```

1290 printf "$(gettext "Looking to see if the machine account
1291 ldapsearch -R -T -h "$dc" $ldap_args -b "$dn" -s sub ""
1292 do
1293     [[ $j != dn: || -z $sub_dn || $dn == $sub_dn ]]
1294     if $recreate; then
1295         printf "$(gettext "Deleting the followin
1296 ldapdelete -h "$dc" $ldap_args "$sub_dn"
1297 if [[ $? -ne 0 ]]; then
1298     printf "$(gettext "Error in dele
1299     fi
1300 else
1301     printf "$(gettext "The following object
1302     fi
1303 done
1304 fi

1306 if $recreate; then
1307     ldapdelete -h "$dc" $ldap_args "$dn" > /dev/null 2>&1
1308     if [[ $? -ne 0 ]]; then
1309         printf "$(gettext "Error in deleting object: %s"
1290         printf "$(gettext "Error in deleting object: %s"
1310         error_message
1311     fi
1312 elif $modify_existing; then
1313     : # Nothing to delete
1314 else
1315     printf "$(gettext "A machine account already exists").\n
1296     printf "$(gettext "A machine account already exists").\n
1316     error_message
1317 fi
1318 fi

1320 if $modify_existing; then
1321     cat > "$object" <<EOF
1322 dn: CN=$supcase_nodename,$baseDN
1323 changetype: modify
1324 replace: userPrincipalName
1325 userPrincipalName: $supn
1326 -
1327 replace: servicePrincipalName
1328 servicePrincipalName: host/${fqdn}
1329 -
1330 replace: userAccountControl
1331 userAccountControl: $((userAccountControlBASE + 32 + 2))
1332 -
1333 replace: dnsHostname
1334 dnsHostname: ${fqdn}
1335 EOF

1337 printf "$(gettext "A machine account already exists; updating it
1338 ldapadd -h "$dc" $ldap_args -f "$object" > /dev/null 2>&1
1339 if [[ $? -ne 0 ]]; then
1340     printf "$(gettext "Failed to create the AD object via LD
1341     printf "$(gettext "Failed to create the AD object via LD
1342     error_message
1343 fi
1344 else
1345     cat > "$object" <<EOF
1346 dn: CN=$supcase_nodename,$baseDN
1347 objectClass: computer
1348 cn: $supcase_nodename
1349 sAMAccountName: ${netbios_nodename}
1350 userPrincipalName: $supn
1351 servicePrincipalName: host/${fqdn}
1352 userAccountControl: $((userAccountControlBASE + 32 + 2))
1353 dnsHostname: ${fqdn}

```

```

1353 EOF

1355 printf "$(gettext "Creating the machine account in AD via LDAP")

1357 ldapadd -h "$dc" $ldap_args -f "$object" > /dev/null 2>&1
1358 if [[ $? -ne 0 ]]; then
1359     printf "$(gettext "Failed to create the AD object via LD
1340     printf "$(gettext "Failed to create the AD object via LD
1360     error_message
1361 fi
1362 fi

1364 # Generate a new password for the new account
1365 MAX_PASS=32
1366 i=0

1368 while :
1369 do
1370     while ((MAX_PASS > i))
1371     do
1372         # 94 elements in the printable character set starting
1373         # at decimal 33, contiguous.
1374         dig=$((RANDOM%94+33))
1375         c=$(printf "\\\`printf %o $dig`\n")
1376         p=$p$c
1377         ((i+=1))
1378     done

1380 # Ensure that we have four character classes.
1381 d=${p%[[:digit:]]*}
1382 a=${p%[[:lower:]]*}
1383 A=${p%[[:upper:]]*}
1384 x=${p%[[:punct:]]*}

1386 # Just compare the number of characters from what was previously
1387 # matched. If there is a difference then we found a match.
1388 n=${#p}
1389 [[ ${#d} -ne $n && ${#a} -ne $n && \
1390    ${#A} -ne $n && ${#x} -ne $n ]] && break
1391 i=0
1392 p=
1393 done
1394 newpw=$p

1396 # Set the new password
1397 printf "%s" $newpw | $KSETPW ${netbios_nodename}@${realm} > /dev/null 2>
1398 if [[ $? -ne 0 ]]
1399 then
1400     printf "$(gettext "Failed to set account password").\n" >&2
1381     printf "$(gettext "Failed to set account password").\n"
1401     error_message
1402 fi

1404 # Lookup the new principal's kvno:
1405 ldapsearch -R -T -h "$dc" $ldap_args -b "$baseDN" \
1406 -s sub cn=$supcase_nodename msDS-KeyVersionNumber | \
1407 grep "^msDS-KeyVersionNumber"|read j kvno
1408 [[ -z $kvno ]] && kvno=1

1410 # Set supported encyptes. This only works for Longhorn/Vista, so we
1411 # ignore errors here.
1412 userAccountControl=$((userAccountControlBASE + 524288 + 65536))
1413 set -A encyptes --

1415 # Do we have local support for AES?
1416 encrypt -l|grep ^aes|read j minkeysize maxkeysize

```

```

1417     val=
1418     if [[ $maxkeysize -eq 256 ]]; then
1419         val=16
1420         enctypees[{$#enctypees[@]}]=aes256-cts-hmac-shal-96
1421     fi
1422     if [[ $minkeysize -eq 128 ]]; then
1423         ((val=val+8))
1424         enctypees[{$#enctypees[@]}]=aes128-cts-hmac-shal-96
1425     fi

1427     # RC4 comes next (whether it's better than 1DES or not -- AD prefers it)
1428     if encrypt -l|$grep -q ^arcfour
1429     then
1430         ((val=val+4))
1431         enctypees[{$#enctypees[@]}]=arcfour-hmac-md5
1432     else
1433         # Use 1DES ONLY if we don't have arcfour
1434         userAccountControl=$((userAccountControl + 2097152))
1435     fi
1436     if encrypt -l | $grep -q ^des
1437     then
1438         ((val=val+2))
1439         ((val=val+1+2))
1440         enctypees[{$#enctypees[@]}]=des-cbc-crc
1441         enctypees[{$#enctypees[@]}]=des-cbc-md5
1442     fi

1442     if [[ {$#enctypees[@]} -eq 0 ]]
1443     then
1444         printf "$(gettext "No enctypees are supported").\n"
1445         printf "$(gettext "Please enable arcfour or 1DES, then re-join;
1446         printf "$(gettext "Please enable arcfour or 1DES, then re-join;
1447         error_message
1448     fi

1449     # If domain crontrroller is Longhorn or above then set new supported
1450     # encryption type attributes.
1451     if [[ $level -gt 2 ]]; then
1452         cat > "$object" <<EOF
1453 dn: CN=$supcase_nodename,$baseDN
1454 changetype: modify
1455 replace: msDS-SupportedEncryptionTypes
1456 msDS-SupportedEncryptionTypes: $val
1457 EOF
1458     ldapmodify -h "$dc" $ldap_args -f "$object" >/dev/null 2>&1
1459     if [[ $? -ne 0 ]]; then
1460         printf "$(gettext "Warning: Could not set the supported
1461         fi
1462     fi

1464     # We should probably check whether arcfour is available, and if not,
1465     # then set the 1DES only flag, but whatever, it's not likely NOT to be
1466     # available on S10/Nevada!

1468     # Reset userAccountControl
1469     #
1470     # NORMAL_ACCOUNT (512) | DONT_EXPIRE_PASSWORD (65536) |
1471     # TRUSTED_FOR_DELEGATION (524288)
1472     #
1473     # and possibly UseDesOnly (2097152) (see above)
1474     #
1475     cat > "$object" <<EOF
1476 dn: CN=$supcase_nodename,$baseDN
1477 changetype: modify
1478 replace: userAccountControl
1479 userAccountControl: $userAccountControl

```

```

1480 EOF
1481     ldapmodify -h "$dc" $ldap_args -f "$object" >/dev/null 2>&1
1482     if [[ $? -ne 0 ]]; then
1483         printf "$(gettext "ldapmodify failed to modify account attribute
1484         printf "$(gettext "ldapmodify failed to modify account attribute
1485         error_message
1486     fi

1487     # Setup a keytab file
1488     set -A args --
1489     for enctypee in "{$enctypees[@]}"
1490     do
1491         args[{$#args[@]}]=--e
1492         args[{$#args[@]}]=$enctypee
1493     done

1495     rm $new_keytab > /dev/null 2>&1

1497     cat > "$object" <<EOF
1498 dn: CN=$supcase_nodename,$baseDN
1499 changetype: modify
1500 add: servicePrincipalName
1501 servicePrincipalName: nfs/${fqdn}
1502 servicePrincipalName: HTTP/${fqdn}
1503 servicePrincipalName: root/${fqdn}
1504 servicePrincipalName: cifs/${fqdn}
1505 EOF
1506     ldapmodify -h "$dc" $ldap_args -f "$object" >/dev/null 2>&1
1507     if [[ $? -ne 0 ]]; then
1508         printf "$(gettext "ldapmodify failed to modify account attribute
1509         printf "$(gettext "ldapmodify failed to modify account attribute
1510         error_message
1511     fi

1512     #
1513     # In Windows, unlike MIT based implementations we salt the keys with
1514     # the UPN, which is based on the host/fqdn@realm elements, not with the
1515     # individual SPN strings.
1516     #
1517     salt=host/${fqdn}@${realm}

1519     printf "%s" $newpw | $KSETPW -n -s $salt -v $kvno -k "$new_keytab" "${ar
1520     printf "%s" $newpw | $KSETPW -n -v $kvno -k "$new_keytab" "${args[@]}" h
1521     if [[ $? -ne 0 ]]
1522     then
1523         printf "$(gettext "Failed to set account password").\n" >&2
1524         printf "$(gettext "Failed to set account password").\n"
1525         error_message
1526     fi

1526     printf "%s" $newpw | $KSETPW -n -s $salt -v $kvno -k "$new_keytab" "${ar
1527     if [[ $? -ne 0 ]]
1528     then
1529         printf "$(gettext "Failed to set account password").\n" >&2
1530         error_message
1531     fi

1533     # Could be setting ${netbios_nodename}@${realm}, but for now no one
1534     # is requesting this.

1536     printf "%s" $newpw | $KSETPW -n -s $salt -v $kvno -k "$new_keytab" "${ar
1537     print "%s" $newpw | $KSETPW -n -v $kvno -k "$new_keytab" "${args[@]}" nf
1538     if [[ $? -ne 0 ]]
1539     then
1540         printf "$(gettext "Failed to set account password").\n" >&2
1541         printf "$(gettext "Failed to set account password").\n"
1542     fi

```

new/usr/src/cmd/krb5/kadmin/kclient/kclient.sh 13

```
1540         error_message
1541     fi

1543     printf "%s" $newpw | $KSETPW -n -s $salt -v $kvno -k "$new_keytab" "${ar
1510     print "%s" $newpw | $KSETPW -n -v $kvno -k "$new_keytab" "${args[@]}" HT
1544     if [[ $? -ne 0 ]]
1545     then
1546         printf "$(gettext "Failed to set account password").\n" >&2
1513     printf "$(gettext "Failed to set account password").\n"
1547     error_message
1548     fi

1550     printf "%s" $newpw | $KSETPW -n -s $salt -v $kvno -k "$new_keytab" "${ar
1517     print "%s" $newpw | $KSETPW -n -v $kvno -k "$new_keytab" "${args[@]}" ro
1551     if [[ $? -ne 0 ]]
1552     then
1553         printf "$(gettext "Failed to set account password").\n" >&2
1520     printf "$(gettext "Failed to set account password").\n"
1554     error_message
1555     fi

1557     printf "%s" $newpw | $KSETPW -n -s $salt -v $kvno -k "$new_keytab" "${ar
1558     if [[ $? -ne 0 ]]
1559     then
1560         printf "$(gettext "Failed to set account password").\n" >&2
1561     error_message
1562     fi

1564     doKRB5config

1566     addDNSRR $dom

1568     setSMB $dom $dc

1570     printf -- "\n-----\n"
1571     printf "$(gettext "Setup COMPLETE").\n\n"

1573     kdestroy -q 1>$TMP_FILE 2>&1
1574     rm -f $TMP_FILE
1575     rm -rf $TMPDIR > /dev/null 2>&1

1577     exit 0
1578 }

1580 #####
1581 # Main section #
1582 #####
1583 #
1584 # Set the Kerberos config file and some default strings/files
1585 #
1586 KRB5_CONFIG_FILE=/etc/krb5/krb5.conf
1587 KRB5_KEYTAB_FILE=/etc/krb5/krb5.keytab
1588 RESOLV_CONF_FILE=/etc/resolv.conf

1590 KLOOKUP=/usr/lib/krb5/klookup; check_bin $KLOOKUP
1591 KSETPW=/usr/lib/krb5/ksetpw; check_bin $KSETPW
1592 KSMB=/usr/lib/krb5/ksmb; check_bin $KSMB
1593 KDYNDNS=/usr/lib/krb5/kdyndns; check_bin $KDYNDNS

1595 dns_lookup=no
1596 ask_fqdns=no
1597 adddns=no
1598 no_keytab=no
1599 checkval=""
1600 profile=""
1601 typeset -u realm
```

new/usr/src/cmd/krb5/kadmin/kclient/kclient.sh 14

```
1602 typeset -l hostname KDC

1604 export TMPDIR="/var/run/kclient"

1606 mkdir $TMPDIR > /dev/null 2>&1
1607 if [[ $? -ne 0 ]]; then
1608     printf "\n$(gettext "Can not create directory: %s")\n\n" $TMPDIR >&2
1609     exit 1
1610 fi

1612 TMP_FILE=$(mktemp -q -t kclient-tmpfile.XXXXXX)
1613 export KRB5_CONFIG=$(mktemp -q -t kclient-krb5conf.XXXXXX)
1614 export KRB5CCNAME=$(mktemp -q -t kclient-krb5ccache.XXXXXX)
1615 new_keytab=$(mktemp -q -t kclient-krb5keytab.XXXXXX)
1616 if [[ -z $TMP_FILE || -z $KRB5_CONFIG || -z $KRB5CCNAME || -z $new_keytab ]]
1617 then
1618     printf "\n$(gettext "Can not create temporary files, exiting").\n\n" >&2
1619     exit 1
1574     printf "\n$(gettext "Can not create temporary file, exiting").\n" >&2
1575     error_message
1620 fi

1622 #
1623 # If we are interrupted, cleanup after ourselves
1624 #
1625 trap "exiting 1" HUP INT QUIT TERM

1627 if [[ -d /usr/bin ]]; then
1628     if [[ -d /usr/sbin ]]; then
1629         PATH=/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:$PATH
1630         export PATH
1631     else
1632         printf "\n$(gettext "Directory /usr/sbin not found, exiting").\n"
1633         exit 1
1634     fi
1635 else
1636     printf "\n$(gettext "Directory /usr/bin not found, exiting").\n" >&2
1637     exit 1
1638 fi

1640 printf "\n$(gettext "Starting client setup")\n\n"
1641 printf -- "-----\n"

1643 #
1644 # Check for uid 0, disallow otherwise
1645 #
1646 id 1>$TMP_FILE 2>&1
1647 if [[ $? -eq 0 ]]; then
1648     if egrep -s "uid=0\(root\) " $TMP_FILE; then
1649         # uid is 0, go ahead ...
1650         :
1651     else
1652         printf "\n$(gettext "Administrative privileges are required to r
1653         error_message
1654     fi
1655 else
1656     cat $TMP_FILE;
1657     printf "\n$(gettext "uid check failed, exiting").\n" >&2
1658     error_message
1659 fi

1661 uname=$(uname -n)
1662 hostname=${uname%%.*}

1664 #
1665 # Process the command-line arguments (if any)
```

```

1666 #
1667 OPTIND=1
1668 while getopts nD:Kp:R:k:a:c:d:f:h:m:s:T: OPTIONS
1669 do
1670     case $OPTIONS in
1671         D) options="$options -D"
1672            domain_list="$OPTARG"
1673            ;;
1674         K) options="$options -K"
1675            no_keytab=yes
1676            ;;
1677         R) options="$options -R"
1678            realm="$OPTARG"
1679            checkval="REALM"; check_value $realm
1680            ;;
1681         T) options="$options -T"
1682            type="$OPTARG"
1683            if [[ $type == ms_ad ]]; then
1684                msad=yes
1685                adddns=yes
1686            else
1687                non_solaris=yes
1688                no_keytab=yes
1689            fi
1690            ;;
1691         a) options="$options -a"
1692            ADMIN_PRINC="$OPTARG"
1693            checkval="ADMIN_PRINC"; check_value $ADMIN_PRINC
1694            ;;
1695         c) options="$options -c"
1696            filepath="$OPTARG"
1697            ;;
1698         d) options="$options -d"
1699            dnsarg="$OPTARG"
1700            checkval="DNS_OPTIONS"; check_value $dnsarg
1701            ;;
1702         f) options="$options -f"
1703            fqdnlist="$OPTARG"
1704            ;;
1705         h) options="$options -h"
1706            logical_hn="$OPTARG"
1707            checkval="LOGICAL_HOSTNAME"; check_value $logical_hn
1708            ;;
1709         k) options="$options -k"
1710            kdc_list="$OPTARG"
1711            ;;
1712         m) options="$options -m"
1713            KDC="$OPTARG"
1714            checkval="KDC"; check_value $KDC
1715            ;;
1716         n) options="$options -n"
1717            add_nfs=yes
1718            ;;
1719         p) options="$options -p"
1720            profile="$OPTARG"
1721            read_profile $profile
1722            ;;
1723         s) options="$options -s"
1724            svc_list="$OPTARG"
1725            SVCs=${svc_list//,/ }
1726            ;;
1727         \?) usage
1728            ;;
1729         *) usage
1730            ;;
1731     esac

```

```

1732 done
1733
1734 #correct argument count after options
1735 shift `expr $OPTIND - 1`
1736
1737 if [[ -z $options ]]; then
1738     :
1739 else
1740     if [[ $# -ne 0 ]]; then
1741         usage
1742     fi
1743 fi
1744
1745 #
1746 # Check to see if we will be a client of a MIT, Heimdal, Shishi, etc.
1747 #
1748 if [[ -z $options ]]; then
1749     query "$(gettext "Is this a client of a non-Solaris KDC") ?"
1750     non_solaris=$answer
1751     if [[ $non_solaris == yes ]]; then
1752         printf "$(gettext "Which type of KDC is the server"):\n"
1753         printf "\t$(gettext "ms_ad: Microsoft Active Directory")\n"
1754         printf "\t$(gettext "mit: MIT KDC server")\n"
1755         printf "\t$(gettext "heimdal: Heimdal KDC server")\n"
1756         printf "\t$(gettext "shishi: Shishi KDC server")\n"
1757         printf "$(gettext "Enter required KDC type"): "
1758         read kdc_type
1759         if [[ $kdc_type == ms_ad ]]; then
1760             msad=yes
1761         elif [[ $kdc_type == mit || $kdc_type == heimdal || \
1762               $kdc_type == shishi ]]; then
1763             no_keytab=yes
1764         else
1765             printf "\n$(gettext "Invalid KDC type option, valid type\n"
1766                error_message
1767             fi
1768         fi
1769     fi
1770
1771     [[ $msad == yes ]] && join_domain
1772
1773 #
1774 # Check for /etc/resolv.conf
1775 #
1776 if [[ -r $RESOLV_CONF_FILE ]]; then
1777     client_machine=`$KLOOKUP`
1778
1779     if [[ $? -ne 0 ]]; then
1780         if [[ $adddns == no ]]; then
1781             printf "\n$(gettext "%s does not have a DNS record and i\n"
1782                error_message
1783             fi
1784         else
1785             #
1786             # If client entry already exists then do not recreate it
1787             #
1788             adddns=no
1789
1790             hostname=${client_machine%.*}
1791             domain=${client_machine#*.*}
1792         fi
1793
1794         short_fqdn=${domain#*.*}
1795         short_fqdn=$(echo $short_fqdn | grep "\.")
1796     else
1797

```

```

1798 #
1799 # /etc/resolv.conf not present, exit ...
1800 #
1801 printf "\n$(gettext "%s does not exist and is required for Kerberos setu
1802 printf "$(gettext "Refer to resolv.conf(4), exiting").\n" >&2
1803 error_message
1804 fi

1806 check_nss_conf || printf "$(gettext "/etc/nsswitch.conf does not make use of DNS

1808 [[ -n $fqdnlist ]] && verify_fqdnlist "$fqdnlist"

1810 if [[ -z $dnsarg && (-z $options || -z $filepath) ]]; then
1811 query "$(gettext "Do you want to use DNS for kerberos lookups") ?"
1812 if [[ $answer == yes ]]; then
1813 printf "\n$(gettext "Valid DNS lookup options are dns_lookup_kdc
1814 printf "\n$(gettext "Enter required DNS option)": "
1815 read dnsarg
1816 checkval="DNS_OPTIONS"; check_value $dnsarg
1817 set_dns_value $dnsarg
1818 fi
1819 else
1820 [[ -z $dnsarg ]] && dnsarg=none
1821 set_dns_value $dnsarg
1822 fi

1824 if [[ -n $kdc_list ]]; then
1825 if [[ -z $KDC ]]; then
1826 for kdc in $kdc_list; do
1827 break
1828 done
1829 KDC="$kdc"
1830 fi
1831 fi

1833 if [[ -z $realm ]]; then
1834 printf "$(gettext "Enter the Kerberos realm)": "
1835 read realm
1836 checkval="REALM"; check_value $realm
1837 fi
1838 if [[ -z $KDC ]]; then
1839 printf "$(gettext "Specify the master KDC hostname for the above realm")
1840 read KDC
1841 checkval="KDC"; check_value $KDC
1842 fi

1844 FKDC=`$KLOOKUP $KDC`

1846 #
1847 # Ping to see if the kdc is alive !
1848 #
1849 ping_check $FKDC "KDC"

1851 if [[ -z $kdc_list && (-z $options || -z $filepath) ]]; then
1852 query "$(gettext "Do you have any slave KDC(s)") ?"
1853 if [[ $answer == yes ]]; then
1854 printf "$(gettext "Enter a comma-separated list of slave KDC hos
1855 read kdc_list
1856 fi
1857 fi

1859 [[ -n $kdc_list ]] && verify_kdcs "$kdc_list"

1861 #
1862 # Check to see if we will have a dynamic presence in the realm
1863 #

```

```

1864 if [[ -z $options ]]; then
1865 query "$(gettext "Will this client need service keys") ?"
1866 if [[ $answer == no ]]; then
1867 no_keytab=yes
1868 fi
1869 fi

1871 #
1872 # Check to see if we are configuring the client to use a logical host name
1873 # of a cluster environment
1874 #
1875 if [[ -z $options ]]; then
1876 query "$(gettext "Is this client a member of a cluster that uses a logic
1877 if [[ $answer == yes ]]; then
1878 printf "$(gettext "Specify the logical hostname of the cluster")
1879 read logical_hn
1880 checkval="LOGICAL_HOSTNAME"; check_value $logical_hn
1881 setup_lhn
1882 fi
1883 fi

1885 if [[ -n $domain_list && (-z $options || -z $filepath) ]]; then
1886 query "$(gettext "Do you have multiple domains/hosts to map to realm %s"
1887 ) ?" $realm
1888 if [[ $answer == yes ]]; then
1889 printf "$(gettext "Enter a comma-separated list of domain/hosts
1890 to map to the default realm)": "
1891 read domain_list
1892 fi
1893 fi
1894 [[ -n domain_list ]] && domain_list=${domain_list//,/, }

1896 #
1897 # Start writing up the krb5.conf file, save the existing one
1898 # if already present
1899 #
1900 writeup_krb5_conf

1902 #
1903 # Is this client going to use krb-nfs? If so then we need to at least
1904 # uncomment the krb5* sec flavors in nfssec.conf.
1905 #
1906 if [[ -z $options ]]; then
1907 query "$(gettext "Do you plan on doing Kerberized nfs") ?"
1908 add_nfs=$answer
1909 fi

1911 if [[ $add_nfs == yes ]]; then
1912 modify_nfssec_conf

1914 #
1915 # We also want to enable gss as we now live in a SBD world
1916 #
1917 svcadm enable svc:/network/rpc/gss:default
1918 [[ $? -ne 0 ]] && printf "$(gettext "Warning: could not enable gss servi
1919 fi

1921 if [[ -z $options ]]; then
1922 query "$(gettext "Do you want to update /etc/pam.conf") ?"
1923 if [[ $answer == yes ]]; then
1924 printf "$(gettext "Enter a list of PAM service names in the foll
1925 read svc_list
1926 SVCs=${svc_list//,/, }
1927 fi
1928 fi
1929 [[ -n $svc_list ]] && update_pam_conf

```

```
1931 #
1932 # Copy over krb5.conf master copy from filepath
1933 #
1934 if [[ -z $options || -z $filepath ]]; then
1935     query "$(gettext "Do you want to copy over the master krb5.conf file") ?
1936     if [[ $answer == yes ]]; then
1937         printf "$(gettext "Enter the pathname of the file to be copied")
1938         read filepath
1939     fi
1940 fi
1941
1942 if [[ -n $filepath && -r $filepath ]]; then
1943     cp $filepath $KRB5_CONFIG
1944     if [[ $? -eq 0 ]]; then
1945         printf "$(gettext "Copied %s to %s").\n" $filepath $KRB5_CONFIG
1946     else
1947         printf "$(gettext "Copy of %s failed, exiting").\n" $filepath >&
1948         error_message
1949     fi
1950 elif [[ -n $filepath ]]; then
1951     printf "\n$(gettext "%s not found, exiting").\n" $filepath >&2
1952     error_message
1953 fi
1954
1955 doKRB5config
1956
1957 #
1958 # Populate any service keys needed for the client in the keytab file
1959 #
1960 if [[ $no_keytab != yes ]]; then
1961     setup_keytab
1962 else
1963     printf "\n$(gettext "Note: %s file not created, please refer to verify_a
1964     printf "$(gettext "Client will also not be able to host services that us
1965 fi
1966
1967 printf -- "\n-----\n"
1968 printf "$(gettext "Setup COMPLETE").\n\n"
1969
1970 #
1971 # If we have configured the client in a cluster we need to remind the user
1972 # to propagate the keytab and configuration files to the other members.
1973 #
1974 if [[ -n $logical_hn ]]; then
1975     printf "\n$(gettext "Note, you will need to securely transfer the /etc/k
1976 fi
1977
1978 #
1979 # Cleanup.
1980 #
1981 kdestroy -q 1>$TMP_FILE 2>&1
1982 rm -f $TMP_FILE
1983 rm -rf $TMPDIR > /dev/null 2>&1
1984 exit 0
```



```

115         len = strlen(optarg);
116         token = strtok_r(optarg, "\\t", &lasts);

118         if (token == NULL)
119             usage();

121     do {
122         if (enctype_count++ == 0) {
123             enctypees = malloc(sizeof (*enctypees));
124         } else {
125             enctypees = realloc(enctypees,
126                               sizeof (*enctypees) * enctype_count);
127         }
128         if (enctypees == NULL) {
129             (void) fprintf(stderr, gettext
130                          ("Couldn't allocate memory"));
131             exit(1);
132         }
133         code = krb5_string_to_enctype(token,
134                                     &enctypees[enctype_count - 1]);

136         if (code != 0) {
137             com_err(whoami, code, gettext("Unknown "
138                                         "or unsupported enctype %s"),
139                   optarg);
140             exit(1);
141         }
142     } while ((token = strtok_r(NULL, "\\t ", &lasts)) !=
143            NULL);
144     break;
145 case 'v':
146     kvno = (krb5_kvno) atoi(optarg);
147     break;
148 case 's':
149     vprincstr = optarg;
150     code = krb5_parse_name(ctx, vprincstr, &salt);
151     if (code != 0) {
152         com_err(whoami, code,
153               gettext("krb5_parse_name(%s) failed"),
154               vprincstr);
155         exit(1);
156     }
157     break;
158 default:
159     usage();
160     break;
161 }
162 }

164 if (nflag && enctype_count == 0)
165     usage();

167 if (nflag == 0 && cc == NULL &&
168     (code = krb5_cc_default(ctx, &cc)) != 0) {
169     com_err(whoami, code, gettext("Could not find a ccache"));
170     exit(1);
171 }

173 if (enctype_count > 0 && kt == NULL &&
174     (code = krb5_kt_default(ctx, &kt)) != 0) {
175     com_err(whoami, code, gettext("No keytab specified"));
176     exit(1);
177 }

179 if (argc != (optind + 1))
180     usage();

```

```

182     vprincstr = argv[optind];
183     code = krb5_parse_name(ctx, vprincstr, &victim);
184     if (code != 0) {
185         com_err(whoami, code, gettext("krb5_parse_name(%s) failed"),
186               vprincstr);
187         exit(1);
188     }

190     if (!isatty(fileno(stdin))) {
191         char buf[PASS_MAX + 1];

193         if (scanf("%* VAL2STR(PASS_MAX) "s", &buf) != 1) {
194             (void) fprintf(stderr,
195                          gettext("Couldn't read new password\n"));
196             exit(1);
197         }

199         newpw = strdup(buf);
200         if (newpw == NULL) {
201             (void) fprintf(stderr,
202                          gettext("Couldn't allocate memory\n"));
203             exit(1);
204         }
205     } else {
206         newpw = getpassphrase(gettext("Enter new password: "));
207         if (newpw == NULL) {
208             (void) fprintf(stderr,
209                          gettext("Couldn't read new password\n"));
210             exit(1);
211         }

213         newpw = strdup(newpw);
214         if (newpw == NULL) {
215             (void) fprintf(stderr,
216                          gettext("Couldn't allocate memory\n"));
217             exit(1);
218         }
219     }

221     if (nflag == 0) {
222         code = krb5_set_password_using_ccache(ctx, cc, newpw, victim,
223                                             &result_code, &result_code_string, &result_string);
224         if (code != 0) {
225             com_err(whoami, code,
226                   gettext("krb5_set_password() failed"));
227             exit(1);
228         }
229         krb5_cc_close(ctx, cc);

231         (void) printf("Result: %.*s (%d) %.*s\n",
232                      result_code == 0 ?
233                      strlen("success") : result_code_string.length,
234                      result_code == 0 ? "success" : result_code_string.data,
235                      result_code,
236                      result_string.length, result_string.data);

238         if (result_code != 0) {
239             (void) fprintf(stderr, gettext("Exiting...\n"));
240             exit(result_code);
241         }
242     }

244     if (enctype_count && (code = kt_remove_entries(ctx, kt, victim)))
245         goto error;

```

```
247     for (i = 0; i < enctype_count; i++)
248         kt_add_entry(ctx, kt, victim, salt, enctypees[i], kvno, newpw);
249         kt_add_entry(ctx, kt, victim, salt, enctypees[i], kvno, newpw);
250 error:
251     if (kt != NULL)
252         krb5_kt_close(ctx, kt);
253
254     return (code ? 1 : 0);
255 }
    unchanged_portion_omitted_

327 static
328 void
329 kt_add_entry(krb5_context ctx, krb5_keytab kt, const krb5_principal princ,
330             const krb5_principal sprinc, krb5_enctype enctype, krb5_kvno kvno,
331             const char *pw)
332     krb5_enctype enctype, krb5_kvno kvno, const char *pw)
333 {
334     krb5_keytab_entry *entry;
335     krb5_data password, salt;
336     krb5_keyblock key;
337     krb5_error_code code;
338     char buf[100];
339
340     if ((code = krb5_enctype_to_string(enctype, buf, sizeof (buf))) {
341         com_err(whoami, code, gettext("Enctype %d has no name!"),
342               enctype);
343         return;
344     }
345     if ((entry = (krb5_keytab_entry *) malloc(sizeof (*entry))) == NULL) {
346         (void) fprintf(stderr, gettext("Couldn't allocate memory"));
347         return;
348     }
349     (void) memset((char *)entry, 0, sizeof (*entry));
350
351     password.length = strlen(pw);
352     password.data = (char *)pw;
353
354     if ((code = krb5_principal2salt(ctx, sprinc, &salt)) != 0) {
355         if ((code = krb5_principal2salt(ctx, princ, &salt)) != 0) {
356             com_err(whoami, code,
357                   gettext("Could not compute salt for %s"), enctype);
358             return;
359         }
360     }
361     code = krb5_c_string_to_key(ctx, enctype, &password, &salt, &key);
362
363     if (code != 0) {
364         com_err(whoami, code, gettext("Could not compute salt for %s"),
365               enctype);
366         krb5_xfree(salt.data);
367         return;
368     }
369     (void) memcpy(&entry->key, &key, sizeof (krb5_keyblock));
370     entry->vno = kvno;
371     entry->principal = princ;
372
373     if ((code = krb5_kt_add_entry(ctx, kt, entry)) != 0) {
374         com_err(whoami, code,
375               gettext("Could not add entry to keytab"));
376     }
377 }
    unchanged_portion_omitted_
```

new/usr/src/lib/gss\_mechs/mech\_krb5/mech/accept\_sec\_context.c

1

```
*****
36748 Thu May 7 01:13:46 2009
new/usr/src/lib/gss_mechs/mech_krb5/mech/accept_sec_context.c
6817447 libgss and various mechs are hiding both the real minor_status and the e
6405422 Solaris acceptors fail in AD-KDC environments when using non-"host" serv
6824434 Unable to accept context establishment initiated by Windows 2000 clients
6787343 kclient's site lookups fail in certain network environments
6692646 kclient should output errors to stderr
6525327 kinit failed when arcfour-hmac-md5-exp was used for the principal's key
6745582 SUNWkdcu missing package dependencies after kclientv2 integration
*****
1 /*
2  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
3  * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
4  * Use is subject to license terms.
5  */
6 /*
7  * Copyright 2000, 2004 by the Massachusetts Institute of Technology.
8  * All Rights Reserved.
9  *
10 * Export of this software from the United States of America may
11 * require a specific license from the United States Government.
12 * It is the responsibility of any person or organization contemplating
13 * export to obtain such a license before exporting.
14 *
15 * WITHIN THAT CONSTRAINT, permission to use, copy, modify, and
16 * distribute this software and its documentation for any purpose and
17 * without fee is hereby granted, provided that the above copyright
18 * notice appear in all copies and that both that copyright notice and
19 * this permission notice appear in supporting documentation, and that
20 * the name of M.I.T. not be used in advertising or publicity pertaining
21 * to distribution of the software without specific, written prior
22 * permission. Furthermore if you modify this software you must label
23 * your software as modified software and not distribute it in such a
24 * fashion that it might be confused with the original M.I.T. software.
25 * M.I.T. makes no representations about the suitability of
26 * this software for any purpose. It is provided "as is" without express
27 * or implied warranty.
28 *
29 */
30 /*
31 * Copyright 1993 by OpenVision Technologies, Inc.
32 *
33 * Permission to use, copy, modify, distribute, and sell this software
34 * and its documentation for any purpose is hereby granted without fee,
35 * provided that the above copyright notice appears in all copies and
36 * that both that copyright notice and this permission notice appear in
37 * supporting documentation, and that the name of OpenVision not be used
38 * in advertising or publicity pertaining to distribution of the software
39 * without specific, written prior permission. OpenVision makes no
40 * representations about the suitability of this software for any
41 * purpose. It is provided "as is" without express or implied warranty.
42 *
43 * OPENVISION DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
44 * INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
45 * EVENT SHALL OPENVISION BE LIABLE FOR ANY SPECIAL, INDIRECT OR
46 * CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
47 * USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
48 * OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
49 * PERFORMANCE OF THIS SOFTWARE.
50 */
52 /*
53 * Copyright (C) 1998 by the FundsXpress, INC.
```

new/usr/src/lib/gss\_mechs/mech\_krb5/mech/accept\_sec\_context.c

2

```
54 *
55 * All rights reserved.
56 *
57 * Export of this software from the United States of America may require
58 * a specific license from the United States Government. It is the
59 * responsibility of any person or organization contemplating export to
60 * obtain such a license before exporting.
61 *
62 * WITHIN THAT CONSTRAINT, permission to use, copy, modify, and
63 * distribute this software and its documentation for any purpose and
64 * without fee is hereby granted, provided that the above copyright
65 * notice appear in all copies and that both that copyright notice and
66 * this permission notice appear in supporting documentation, and that
67 * the name of FundsXpress. not be used in advertising or publicity pertaining
68 * to distribution of the software without specific, written prior
69 * permission. FundsXpress makes no representations about the suitability of
70 * this software for any purpose. It is provided "as is" without express
71 * or implied warranty.
72 *
73 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
74 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
75 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
76 */
78 #include "k5-int.h"
79 #include "gssapiP_krb5.h"
80 #ifdef HAVE_MEMORY_H
81 #include <memory.h>
82 #endif
83 #include <assert.h>
84 #include "auth_con.h"
86 #ifdef CFX_EXERCISE
87 #define CFX_ACCEPTOR_SUBKEY (time(0) & 1)
88 #else
89 #define CFX_ACCEPTOR_SUBKEY 1
90 #endif
92 /*
93 * Decode, decrypt and store the forwarded creds in the local ccache.
94 * and populate the callers delegated credential handle if it
95 * was provided.
96 */
97 static krb5_error_code
98 rd_and_store_for_creds(context, auth_context, inbuf, out_cred)
99     krb5_context context;
100     krb5_auth_context auth_context;
101     krb5_data *inbuf;
102     krb5_gss_cred_id_t *out_cred;
103 {
104     krb5_creds ** creds = NULL;
105     krb5_error_code retval;
106     krb5_ccache ccache = NULL;
107     krb5_gss_cred_id_t cred = NULL;
108     krb5_auth_context new_auth_ctx = NULL;
109     krb5_int32 flags_org;
111     /* Solaris Kerberos */
112     KRB5_LOG0(KRB5_INFO, "rd_and_store_for_creds() start");
114     if ((retval = krb5_auth_con_getflags(context, auth_context, &flags_org))
115         return retval;
116     krb5_auth_con_setflags(context, auth_context,
117                             0);
119     /*
```

```

120  * By the time krb5_rd_cred is called here (after krb5_rd_req has been
121  * called in krb5_gss_accept_sec_context), the "keyblock" field of
122  * auth_context contains a pointer to the session key, and the
123  * "recv_subkey" field might contain a session subkey. Either of
124  * these (the "recv_subkey" if it isn't NULL, otherwise the
125  * "keyblock") might have been used to encrypt the encrypted part of
126  * the KRB_CRED message that contains the forwarded credentials. (The
127  * Java Crypto and Security Implementation from the DSTC in Australia
128  * always uses the session key. But apparently it never negotiates a
129  * subkey, so this code works fine against a JCSI client.) Up to the
130  * present, though, GSSAPI clients linked against the MIT code (which
131  * is almost all GSSAPI clients) don't encrypt the KRB_CRED message at
132  * all -- at this level. So if the first call to krb5_rd_cred fails,
133  * we should call it a second time with another auth context freshly
134  * created by krb5_auth_con_init. All of its keyblock fields will be
135  * NULL, so krb5_rd_cred will assume that the KRB_CRED message is
136  * unencrypted. (The MIT code doesn't actually send the KRB_CRED
137  * message in the clear -- the "authenticator" whose "checksum" ends up
138  * containing the KRB_CRED message does get encrypted.)
139  */
140  /* Solaris Kerberos */
141  if ((retval = krb5_rd_cred(context, auth_context, inbuf, &creds, NULL))) {
142      krb5_etype enctype = ENCTYPE_NULL;
143      /*
144       * If the client is using non-DES etypes it really ought to
145       * send encrypted KRB-CREDS...
146       */
147      if (auth_context->keyblock != NULL)
148          enctype = auth_context->keyblock->enctype;
149      switch (enctype) {
150      case ENCTYPE_DES_CBC_MD5:
151      case ENCTYPE_DES_CBC_CRC:
152      case ENCTYPE_DES3_CBC_SHA1:
153          break;
154      default:
155          KRB5_LOG(KRB5_ERR, "rd_and_store_for_creds() error "
156                  "krb5_rd_cred() retval = %d\n", retval);
157          goto cleanup;
158          /* NOTREACHED */
159          break;
160      }
161
162      /* Try to krb5_rd_cred() likely unencrypted KRB-CRED */
163      if ((retval = krb5_auth_con_init(context, &new_auth_ctx))
164          goto cleanup;
165      krb5_auth_con_setflags(context, new_auth_ctx, 0);
166      if ((retval = krb5_rd_cred(context, new_auth_ctx, inbuf,
167                               &creds, NULL))) {
168          /* Solaris Kerberos */
169          KRB5_LOG(KRB5_ERR, "rd_and_store_for_creds() error "
170                  "krb5_rd_cred() retval = %d\n", retval);
171          goto cleanup;
172      }
173  }
174
175  if ((retval = krb5_cc_new_unique(context, "MEMORY", NULL, &ccache)) {
176      ccache = NULL;
177      goto cleanup;
178  }
179
180  if ((retval = krb5_cc_initialize(context, ccache, creds[0]->client)) {
181      /* Solaris Kerberos */
182      KRB5_LOG(KRB5_ERR, "rd_and_store_for_creds() error "
183              "krb5_cc_initialize() retval = %d\n", retval);
184      goto cleanup;
185  }

```

```

187  if ((retval = krb5_cc_store_cred(context, ccache, creds[0]))) {
188      /* Solaris Kerberos */
189      KRB5_LOG(KRB5_ERR, "rd_and_store_for_creds() error "
190              "krb5_cc_store_cred() retval = %d\n", retval);
191      goto cleanup;
192  }
193
194  /* generate a delegated credential handle */
195  if (out_cred) {
196      /* allocate memory for a cred_t... */
197      if (!(cred =
198            (krb5_gss_cred_id_t) xmalloc(sizeof(krb5_gss_cred_id_rec))) {
199          retval = ENOMEM; /* out of memory? */
200          *out_cred = NULL;
201          goto cleanup;
202      }
203
204      /* zero it out... */
205      /* Solaris Kerberos */
206      (void) memset(cred, 0, sizeof(krb5_gss_cred_id_rec));
207
208      retval = k5_mutex_init(&cred->lock);
209      if (retval) {
210          xfree(cred);
211          cred = NULL;
212          goto cleanup;
213      }
214
215      /* copy the client principle into it... */
216      if ((retval =
217            krb5_copy_principal(context, creds[0]->client, &(cred->princ))) {
218          /* Solaris Kerberos */
219          KRB5_LOG(KRB5_ERR, "rd_and_store_for_creds() error "
220                  "krb5_copy_principal() retval = %d\n", retval);
221          k5_mutex_destroy(&cred->lock);
222          retval = ENOMEM; /* out of memory? */
223          xfree(cred); /* clean up memory on failure */
224          cred = NULL;
225          goto cleanup;
226      }
227
228      cred->usage = GSS_C_INITIATE; /* we can't accept with this */
229      /* cred->princ already set */
230      cred->prerfc_mech = 1; /* this cred will work with all three mechs */
231      cred->rfdc_mech = 1;
232      cred->keytab = NULL; /* no keytab associated with this... */
233      cred->tgt_expire = creds[0]->times.endtime; /* store the end time */
234      cred->ccache = ccache; /* the ccache containing the credential */
235      ccache = NULL; /* cred takes ownership so don't destroy */
236  }
237
238  /* If there were errors, there might have been a memory leak
239  if (!cred)
240  if ((retval = krb5_cc_close(context, ccache))
241      goto cleanup;
242  */
243  cleanup:
244  if (creds)
245      krb5_free_tgt_creds(context, creds);
246
247  if (ccache)
248      (void)krb5_cc_destroy(context, ccache);
249
250  if (out_cred)
251      *out_cred = cred; /* return credential */

```

```

253     if (new_auth_ctx)
254         krb5_auth_con_free(context, new_auth_ctx);

256     krb5_auth_con_setflags(context, auth_context, flags_org);

258     /* Solaris Kerberos */
259     KRB5_LOG(KRB5_INFO, "rd_and_store_for_creds() end retval %d", retval);
260     return retval;
261 }

263 /*
264  * SUNW15resync
265  * Most of the logic here left "as is" because of lots of fixes MIT
266  * does not have yet
267  */
268 OM_uint32
269 krb5_gss_accept_sec_context(minor_status, context_handle,
270                             verifier_cred_handle, input_token,
271                             input_chan_bindings, src_name, mech_type,
272                             output_token, ret_flags, time_rec,
273                             delegated_cred_handle)
274 {
275     OM_uint32 *minor_status;
276     gss_ctx_id_t *context_handle;
277     gss_cred_id_t verifier_cred_handle;
278     gss_buffer_t input_token;
279     gss_channel_bindings_t input_chan_bindings;
280     gss_name_t *src_name;
281     gss_OID *mech_type;
282     gss_buffer_t output_token;
283     OM_uint32 *ret_flags;
284     OM_uint32 *time_rec;
285     gss_cred_id_t *delegated_cred_handle;
286     krb5_context context;
287     unsigned char *ptr, *ptr2;
288     char *sptr;
289     long tmp;
290     size_t md5len;
291     int bigend;
292     krb5_gss_cred_id_t cred = 0;
293     krb5_data ap_rep, ap_req;
294     krb5_ap_req *request = NULL;
295     int i;
296     krb5_error_code code;
297     krb5_address addr, *paddr;
298     krb5_authenticator *authdat = 0;
299     krb5_checksum reqcksum;
300     krb5_principal name = NULL;
301     krb5_ui_4 gss_flags = 0;
302     krb5_gss_ctx_id_rec *ctx = 0;
303     krb5_timestamp now;
304     gss_buffer_desc token;
305     krb5_auth_context auth_context = NULL;
306     krb5_ticket *ticket = NULL;
307     int option_id;
308     krb5_data option;
309     const gss_OID_desc *mech_used = NULL;
310     OM_uint32 major_status = GSS_S_FAILURE;
311     krb5_error krb_error_data;
312     krb5_data scratch;
313     gss_cred_id_t cred_handle = NULL;
314     krb5_gss_cred_id_t deleg_cred = NULL;
315     OM_uint32 saved_ap_options = 0;
316     krb5int_access kaccess;
317     int cred_rcache = 0;

```

```

318     OM_uint32 t_minor_status = 0;

320     KRB5_LOG0(KRB5_INFO, "krb5_gss_accept_sec_context() start");

322     code = krb5int_accessor (&kaccess, KRB5INT_ACCESS_VERSION);
323     if (code) {
324         *minor_status = code;
325         return (GSS_S_FAILURE);
326     }

328     code = krb5_gss_init_context(&context);
329     if (code) {
330         *minor_status = code;
331         return GSS_S_FAILURE;
332     }

334     /* set up returns to be freeable */

336     if (src_name)
337         *src_name = (gss_name_t) NULL;
338     output_token->length = 0;
339     output_token->value = NULL;
340     token.value = 0;
341     reqcksum.contents = 0;
342     ap_req.data = 0;
343     ap_rep.data = 0;
344
345     if (mech_type)
346         *mech_type = GSS_C_NULL_OID;
347     /* initialize the delegated cred handle to NO_CREDENTIAL for now */
348     if (delegated_cred_handle)
349         *delegated_cred_handle = GSS_C_NO_CREDENTIAL;

351     /*
352     * Context handle must be unspecified.  Actually, it must be
353     * non-established, but currently, accept_sec_context never returns
354     * a non-established context handle.
355     */
356     /*SUPPRESS 29*/
357     if (*context_handle != GSS_C_NO_CONTEXT) {
358         *minor_status = 0;

360         /* Solaris kerberos: the original Solaris code returned GSS_S_NO_CONTEXT
361         * for this error.  This conflicts somewhat with RFC2743 which states
362         * GSS_S_NO_CONTEXT should be returned only for successor calls following
363         * GSS_S_CONTINUE_NEEDED status returns.  Note the MIT code doesn't
364         * return GSS_S_NO_CONTEXT at all.
365         */

367         major_status = GSS_S_NO_CONTEXT;
368         KRB5_LOG0(KRB5_ERR, "krb5_gss_accept_sec_context() "
369                 "error GSS_S_NO_CONTEXT");
370         goto cleanup;
371     }

373     /* verify the token's integrity, and leave the token in ap_req.
374     figure out which mech oid was used, and save it */

376     ptr = (unsigned char *) input_token->value;

378     if (!(code = g_verify_token_header(gss_mech_krb5,
379                                     (uint32_t *)&(ap_req.length),
380                                     &ptr, KG_TOK_CTX_AP_REQ,
381                                     input_token->length, 1))) {
382         mech_used = gss_mech_krb5;
383     } else if ((code == G_WRONG_MECH) &&

```

```

384         !(code = g_verify_token_header(gss_mech_krb5_old,
385                                     (uint32_t *)&(ap_req.length),
386                                     &ptr, KG_TOK_CTX_AP_REQ,
387                                     input_token->length, 1))) {
388     /*
389     * Previous versions of this library used the old mech_id
390     * and some broken behavior (wrong IV on checksum
391     * encryption). We support the old mech_id for
392     * compatibility, and use it to decide when to use the
393     * old behavior.
394     */
395     mech_used = gss_mech_krb5_old;
396 } else {
397     major_status = GSS_S_DEFECTIVE_TOKEN;
398     goto fail;
399 }

401     sptr = (char *) ptr;
402     TREAD_STR(sptr, ap_req.data, ap_req.length);

404     /*
405     * Solaris Kerberos:
406     * We need to decode the request now so that we can get the
407     * service principal in order to try and acquire a cred for it.
408     * below in the "handle default cred handle" code block.
409     */
410     if (!krb5_is_ap_req(&ap_req)) {
411         code = KRB5KRB_AP_ERR_MSG_TYPE;
412         goto fail;
413     }
414     /* decode the AP-REQ into request */
415     if ((code = decode_krb5_ap_req(&ap_req, &request))) {
416         if (code == KRB5_BADMSGTYPE)
417             code = KRB5KRB_AP_ERR_BADVERSION;
418         goto fail;
419     }

421     /* handle default cred handle */
422     /*
423     * Solaris Kerberos:
424     * If there is no princ associated with the cred then treat it the
425     * the same as GSS_C_NO_CREDENTIAL.
426     */
427     if (verifier_cred_handle == GSS_C_NO_CREDENTIAL ||
428         ((krb5_gss_cred_id_t)verifier_cred_handle->princ == NULL) {
429         /* Note that we try to acquire a cred for the service principal
430         * named in the AP-REQ. This allows us to implement option (ii)
431         * of the recommended behaviour for GSS_Accept_sec_context() as
432         * described in section 1.1.1.3 of RFC2743.

434         * This is far more useful that option (i), for which we would
435         * acquire a cred for GSS_C_NO_NAME.
436         */
437         /* copy the princ from the ap-req or we'll lose it when we free
438         the ap-req */
439         krb5_principal princ;
440         if ((code = krb5_copy_principal(context, request->ticket->server,
441                                     &princ))) {
442             KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
443                     "krb5_copy_principal() error code %d", code);
444             major_status = GSS_S_FAILURE;
445             goto fail;
446         }
447         /* intern the acceptor name */
448         if (!kg_save_name((gss_name_t) princ)) {
449             code = G_VALIDATE_FAILED;

```

```

450         major_status = GSS_S_FAILURE;
451         goto fail;
452     }
453     major_status = krb5_gss_acquire_cred((OM_uint32*) &code,
454                                         (gss_name_t) princ,
455                                         GSS_C_INDEFINITE, GSS_C_NO_OID_SET,
456                                         GSS_C_ACCEPT, &cred_handle,
457                                         NULL, NULL);

459     if (major_status != GSS_S_COMPLETE){
461         /* Solaris kerberos: RFC2743 indicate this should be returned if we
462         * can't aquire a default cred.
463         */
464         KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
465                 "krb5_gss_acquire_cred() error"
466                 "orig major_status = %d, now = GSS_S_NO_CRED\n",
467                 major_status);

469         major_status = GSS_S_NO_CRED;
470         goto fail;
471     }

473     } else {
474         cred_handle = verifier_cred_handle;
475     }

477     major_status = krb5_gss_validate_cred((OM_uint32*) &code,
478                                         cred_handle);

480     if (GSS_ERROR(major_status)){
482         /* Solaris kerberos: RFC2743 indicate GSS_S_NO_CRED should be returned if
483         * the supplied cred isn't valid.
484         */

486         KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
487                 "krb5_gss_validate_cred() error"
488                 "orig major_status = %d, now = GSS_S_NO_CRED\n",
489                 major_status);

491         major_status = GSS_S_NO_CRED;
492         goto fail;
493     }

495     cred = (krb5_gss_cred_id_t) cred_handle;

497     /* make sure the supplied credentials are valid for accept */

499     if ((cred->usage != GSS_C_ACCEPT) &&
500         (cred->usage != GSS_C_BOTH)) {
501         code = 0;
502         KRB5_LOG0(KRB5_ERR, "krb5_gss_accept_sec_context() "
503                 "error GSS_S_NO_CONTEXT");
504         major_status = GSS_S_NO_CRED;
505         goto fail;
506     }

508     /* construct the sender_addr */

510     if ((input_chan_bindings != GSS_C_NO_CHANNEL_BINDINGS) &&
511         (input_chan_bindings->initiator_addrtype == GSS_C_AF_INET)) {
512         /* XXX is this right? */
513         addr.addrtype = ADDRTYPE_INET;
514         addr.length = input_chan_bindings->initiator_address.length;
515         addr.contents = input_chan_bindings->initiator_address.value;

```

```

517     paddr = &addr;
518 } else {
519     paddr = NULL;
520 }

522 /* verify the AP_REQ message - setup the auth_context and rcache */

524 if ((code = krb5_auth_con_init(context, &auth_context)) {
525     major_status = GSS_S_FAILURE;
526     /* Solaris Kerberos */
527     KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
528             "krb5_auth_con_init() error code %d", code);
529     goto fail;
530 }

532 (void) krb5_auth_con_setflags(context, auth_context,
533                             KRB5_AUTH_CONTEXT_DO_SEQUENCE);

535 if (cred->rcache) {
536     cred_rcache = 1;
537     if ((code = krb5_auth_con_setrcache(context, auth_context, cred->rcache))
538         major_status = GSS_S_FAILURE;
539         /* Solaris Kerberos */
540         KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
541                 "krb5_auth_con_setrcache() error code %d", code);
542         goto fail;
543     }
544 }
545 if ((code = krb5_auth_con_setaddrs(context, auth_context, NULL, paddr)) {
546     major_status = GSS_S_FAILURE;
547     /* Solaris Kerberos */
548     KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
549             "krb5_auth_con_setaddrs() error code %d", code);
550     goto fail;
551 }

553 if ((code = krb5_rd_req_decoded(context, &auth_context, request,
554                               cred->princ, cred->keytab, NULL, &ticket)) {
555     KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
556             "krb5_rd_req() error code %d", code);
557     if (code == KRB5_KT_KVNONOTFOUND || code == KRB5_KT_NOTFOUND) {
558         major_status = GSS_S_DEFECTIVE_CREDENTIAL;
559         code = KRB5KRB_AP_ERR_NOKEY;
560     }
561     else if (code == KRB5KRB_AP_WRONG_PRINC) {
562         major_status = GSS_S_NO_CRED;
563         code = KRB5KRB_AP_ERR_NOT_US;
564     }
565     else if (code == KRB5KRB_AP_ERR_REPEAT)
566         major_status = GSS_S_DUPLICATE_TOKEN;
567     else
568         major_status = GSS_S_FAILURE;
569     goto fail;
570 }
571 krb5_auth_con_setflags(context, auth_context,
572                       KRB5_AUTH_CONTEXT_DO_SEQUENCE);

574 krb5_auth_con_getauthenticator(context, auth_context, &authdat);

576 #if 0
577 /* make sure the necessary parts of the authdat are present */

579 if ((authdat->authenticator->subkey == NULL) ||
580     (authdat->ticket->enc_part2 == NULL)) {
581     code = KG_NO_SUBKEY;

```

```

582     major_status = GSS_S_FAILURE;
583     goto fail;
584 }
585 #endif

587 {
588     /* gss krb5 v1 */

590     /* stash this now, for later. */
591     code = krb5_c_checksum_length(context, CKSUMTYPE_RSA_MD5, &md5len);
592     if (code) {
593         /* Solaris Kerberos */
594         KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
595                 "krb5_c_checksum_length() error code %d", code);
596         major_status = GSS_S_FAILURE;
597         goto fail;
598     }

600     /* verify that the checksum is correct */

602     /*
603     The checksum may be either exactly 24 bytes, in which case
604     no options are specified, or greater than 24 bytes, in which case
605     one or more options are specified. Currently, the only valid
606     option is KRB5_GSS_FOR_CREDS_OPTION (= 1 ).
607     */

609     if ((authdat->checksum->checksum_type != CKSUMTYPE_KG_CB) ||
610         (authdat->checksum->length < 24)) {
611         code = 0;
612         major_status = GSS_S_BAD_BINDINGS;
613         goto fail;
614     }

616     /*
617     "Be liberal in what you accept, and
618     conservative in what you send"
619     -- rfc1123

621     This code will let this acceptor interoperate with an initiator
622     using little-endian or big-endian integer encoding.
623     */

625     ptr = (unsigned char *) authdat->checksum->contents;
626     bigend = 0;

628     TREAD_INT(ptr, tmp, bigend);

630     if (tmp != md5len) {
631         ptr = (unsigned char *) authdat->checksum->contents;
632         bigend = 1;

634         TREAD_INT(ptr, tmp, bigend);

636         if (tmp != md5len) {
637             code = KG_BAD_LENGTH;
638             major_status = GSS_S_FAILURE;
639             goto fail;
640         }
641     }

643     /* at this point, bigend is set according to the initiator's
644     byte order */

647     /*

```

```

648     The following section of code attempts to implement the
649     optional channel binding facility as described in RFC2743.

651     Since this facility is optional channel binding may or may
652     not have been provided by either the client or the server.

654     If the server has specified input_chan_bindings equal to
655     GSS_C_NO_CHANNEL_BINDINGS then we skip the check.  If
656     the server does provide channel bindings then we compute
657     a checksum and compare against those provided by the
658     client.  If the check fails we test the clients checksum
659     to see whether the client specified GSS_C_NO_CHANNEL_BINDINGS.
660     If either test succeeds we continue without error.
661 */
662 if ((code = kg_checksum_channel_bindings(context,
663     input_chan_bindings,
664     &reqcksum, bigend))) {
665     /* Solaris Kerberos */
666     KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
667     "kg_checksum_channel_bindings() error code %d", code);
668     major_status = GSS_S_BAD_BINDINGS;
669     goto fail;
670 }

672 TREAD_STR(ptr, ptr2, reqcksum.length);

674 if (input_chan_bindings != GSS_C_NO_CHANNEL_BINDINGS ) {
675     if (memcmp(ptr2, reqcksum.contents, reqcksum.length) != 0) {
676         xfree(reqcksum.contents);
677         reqcksum.contents = 0;
678         if ((code = kg_checksum_channel_bindings(context,
679             GSS_C_NO_CHANNEL_BINDINGS,
680             &reqcksum, bigend))) {
681             major_status = GSS_S_BAD_BINDINGS;
682             goto fail;
683         }
684         if (memcmp(ptr2, reqcksum.contents, reqcksum.length) != 0) {
685             code = 0;
686             major_status = GSS_S_BAD_BINDINGS;
687             goto fail;
688         }
689     }
690 }

691 }

693 TREAD_INT(ptr, gss_flags, bigend);

695 /* if the checksum length > 24, there are options to process */

697 if(authdat->checksum->length > 24 && (gss_flags & GSS_C_DELEG_FLAG)) {

699     i = authdat->checksum->length - 24;

701     if (i >= 4) {

703         TREAD_INT16(ptr, option_id, bigend);

705         TREAD_INT16(ptr, option.length, bigend);

707         i -= 4;

709     if (i < option.length || option.length < 0) {
710         code = KG_BAD_LENGTH;
711         major_status = GSS_S_FAILURE;
712         goto fail;
713     }

```

```

715     /* have to use ptr2, since option.data is wrong type and
716     macro uses ptr as both lvalue and rvalue */

718     TREAD_STR(ptr, ptr2, option.length);
719     option.data = (char *) ptr2;

721     i -= option.length;

723     if (option_id != KRB5_GSS_FOR_CREDS_OPTION) {
724         major_status = GSS_S_FAILURE;
725         goto fail;
726     }

728     /* store the delegated credential */

730     code = rd_and_store_for_creds(context, auth_context, &option,
731         (delegated_cred_handle) ?
732         &deleg_cred : NULL);

733     if (code) {
734         major_status = GSS_S_FAILURE;
735         goto fail;
736     }

738     } /* if i >= 4 */
739     /* ignore any additional trailing data, for now */
740 #ifdef CFX_EXERCISE
741     {
742         FILE *f = fopen("/tmp/gsslog", "a");
743         if (f) {
744             fprintf(f,
745                 "initial context token with delegation, %d extra byte
746                 i);
747             fclose(f);
748         }
749     }
750 #endif
751     } else {
752 #ifdef CFX_EXERCISE
753     {
754         FILE *f = fopen("/tmp/gsslog", "a");
755         if (f) {
756             if (gss_flags & GSS_C_DELEG_FLAG)
757                 fprintf(f,
758                     "initial context token, delegation flag but too s
759             else
760                 /* no deleg flag, length might still be too big */
761                 fprintf(f,
762                     "initial context token, %d extra bytes\n",
763                     authdat->checksum->length - 24);
764             fclose(f);
765         }
766     }
767 #endif
768     }
769 }

771 /* create the ctx struct and start filling it in */

773 if ((ctx = (krb5_gss_ctx_id_rec *) xmalloc(sizeof(krb5_gss_ctx_id_rec)))
774     == NULL) {
775     code = ENOMEM;
776     major_status = GSS_S_FAILURE;
777     goto fail;
778 }

```

```

780 memset(ctx, 0, sizeof(krb5_gss_ctx_id_rec));
781 ctx->mech_used = (gss_OID) mech_used;
782 ctx->auth_context = auth_context;
783 ctx->initiate = 0;
784 ctx->gss_flags = (GSS_C_TRANS_FLAG |
785                 ((gss_flags) & (GSS_C_INTEG_FLAG | GSS_C_CONF_FLAG |
786                 GSS_C_MUTUAL_FLAG | GSS_C_REPLAY_FLAG |
787                 GSS_C_SEQUENCE_FLAG | GSS_C_DELEG_FLAG)));
788 ctx->seed_init = 0;
789 ctx->big_endian = bigend;
790 ctx->cred_rcache = cred_rcache;

792 /* Intern the ctx pointer so that delete_sec_context works */
793 if (! kg_save_ctx_id((gss_ctx_id_t) ctx)) {
794     xfree(ctx);
795     ctx = 0;

797     /* Solaris Kerberos */
798     KRB5_LOG0(KRB5_ERR, "krb5_gss_accept_sec_context() "
799             "kg_save_ctx_id() error");
800     code = G_VALIDATE_FAILED;
801     major_status = GSS_S_FAILURE;
802     goto fail;
803 }

805 if ((code = krb5_copy_principal(context, cred->princ, &ctx->here)) {
806     /* Solaris Kerberos */
807     KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
808             "krb5_copy_principal() error code %d", code);
809     major_status = GSS_S_FAILURE;
810     goto fail;
811 }

813 if ((code = krb5_copy_principal(context, authdat->client, &ctx->there)) {
814     /* Solaris Kerberos */
815     KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
816             "krb5_copy_principal() 2 error code %d", code);
817     major_status = GSS_S_FAILURE;
818     goto fail;
819 }

821 if ((code = krb5_auth_con_getrecvsbkey(context, auth_context,
822                                     &ctx->subkey)) {
823     /* Solaris Kerberos */
824     KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
825             "krb5_auth_con_getremotesubkey() error code %d", code);
826     major_status = GSS_S_FAILURE;
827     goto fail;
828 }

830 /* use the session key if the subkey isn't present */

832 if (ctx->subkey == NULL) {
833     if ((code = krb5_auth_con_getkey(context, auth_context,
834                                     &ctx->subkey)) {
835         /* Solaris Kerberos */
836         KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
837                 "krb5_auth_con_getkey() error code %d", code);
838         *minor_status = (OM_uint32) KRB5KDC_ERR_NULL_KEY;
839         major_status = GSS_S_FAILURE;
840         goto fail;
841     }
842 }

844 if (ctx->subkey == NULL) {
845     /* this isn't a very good error, but it's not clear to me this

```

```

846     can actually happen */
847     major_status = GSS_S_FAILURE;
848     code = KRB5KDC_ERR_NULL_KEY;
849     goto fail;
850 }

852 /* Solaris Kerberos */
853 KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() "
854         "ctx->subkey->enctype=%d", ctx->subkey->enctype);

856 ctx->proto = 0;
857 switch(ctx->subkey->enctype) {
858 case ENCTYPE_DES_CBC_MD5:
859 case ENCTYPE_DES_CBC_CRC:
860     ctx->subkey->enctype = ENCTYPE_DES_CBC_RAW;
861     ctx->signalg = SGN_ALG_DES_MAC_MD5;
862     ctx->cksum_size = 8;
863     ctx->sealalg = SEAL_ALG_DES;

865     /* fill in the encryption descriptors */

867     if ((code = krb5_copy_keyblock(context, ctx->subkey, &ctx->enc)) {
868         major_status = GSS_S_FAILURE;
869         goto fail;
870     }

872     for (i=0; i<ctx->enc->length; i++)
873         /*SUPPRESS 113*/
874         ctx->enc->contents[i] ^= 0xf0;

876     goto copy_subkey_to_seq;
877     break;

879 case ENCTYPE_DES3_CBC_SHA1:
880     ctx->subkey->enctype = ENCTYPE_DES3_CBC_RAW;
881     ctx->signalg = SGN_ALG_HMAC_SHA1_DES3_KD;
882     ctx->cksum_size = 20;
883     ctx->sealalg = SEAL_ALG_DES3KD;

885     /* fill in the encryption descriptors */
886     copy_subkey:
887     if ((code = krb5_copy_keyblock(context, ctx->subkey, &ctx->enc)) {
888         major_status = GSS_S_FAILURE;
889         goto fail;
890     }
891     copy_subkey_to_seq:
892     if ((code = krb5_copy_keyblock(context, ctx->subkey, &ctx->seq)) {
893         major_status = GSS_S_FAILURE;
894         goto fail;
895     }
896     break;

898 case ENCTYPE_ARCFOUR_HMAC:
899     ctx->signalg = SGN_ALG_HMAC_MD5 ;
900     ctx->cksum_size = 8;
901     ctx->sealalg = SEAL_ALG_MICROSOFT_RC4 ;
902     goto copy_subkey;

904 default:
905     ctx->signalg = -1;
906     ctx->sealalg = -1;
907     ctx->proto = 1;
908     code = (*kaccess.krb5int_c_mandatory_cksumtype)(context, ctx->subkey->enc
909                                                     &ctx->cksumtype);
910     if (code)
911         goto fail;

```

```

912     code = krb5_c_checksum_length(context, ctx->cksumtype,
913     (size_t *)&ctx->cksum_size);
914     if (code)
915         goto fail;
916     ctx->have_acceptor_subkey = 0;
917     goto copy_subkey;
918 }

920 /* Solaris Kerberos */
921 KRB5_LOG1(KRB5_ERR, "accept_sec_context: subkey enctype = %d proto = %d",
922     ctx->subkey->enctype, ctx->proto);

924 ctx->endtime = ticket->enc_part2->times.endtime;
925 ctx->krb_flags = ticket->enc_part2->flags;

927 krb5_free_ticket(context, ticket); /* Done with ticket */

929 {
930     krb5_ui_4 seq_temp;
931     krb5_auth_con_getremoteseqnumber(context, auth_context,
932     (krb5_int32 *)&seq_temp);
933     ctx->seq_recv = seq_temp;
934 }

936 if ((code = krb5_timeofday(context, &now))) {
937     major_status = GSS_S_FAILURE;
938     goto fail;
939 }

941 if (ctx->endtime < now) {
942     code = 0;
943     major_status = GSS_S_CREDENTIALS_EXPIRED;
944     goto fail;
945 }

947 g_order_init(&(ctx->seqstate), ctx->seq_recv,
948     (ctx->gss_flags & GSS_C_REPLAY_FLAG) != 0,
949     (ctx->gss_flags & GSS_C_SEQUENCE_FLAG) != 0, ctx->proto);

951 /* at this point, the entire context structure is filled in,
952     so it can be released. */

954 /* generate an AP_REP if necessary */

956 if (ctx->gss_flags & GSS_C_MUTUAL_FLAG) {
957     unsigned char * ptr3;
958     krb5_ui_4 seq_temp;
959     int cfx_generate_subkey;

961     if (ctx->proto == 1)
962         cfx_generate_subkey = CFX_ACCEPTOR_SUBKEY;
963     else
964         cfx_generate_subkey = 0;

966     if (cfx_generate_subkey) {
967         krb5_int32 acflags;
968         code = krb5_auth_con_getflags(context, auth_context, &acflags);
969         if (code == 0) {
970             acflags |= KRB5_AUTH_CONTEXT_USE_SUBKEY;
971             code = krb5_auth_con_setflags(context, auth_context, acflags);
972         }
973         if (code) {
974             major_status = GSS_S_FAILURE;
975             goto fail;
976         }
977     }

```

```

979     if ((code = krb5_mk_rep(context, auth_context, &ap_rep))) {
980         major_status = GSS_S_FAILURE;
981         goto fail;
982     }

984     krb5_auth_con_getlocalseqnumber(context, auth_context,
985     (krb5_int32 *)&seq_temp);
986     ctx->seq_send = seq_temp & 0xffffffffL;

988     if (cfx_generate_subkey) {
989         /* Get the new acceptor subkey. With the code above, there
990             should always be one if we make it to this point. */
991         code = krb5_auth_con_getsendsubkey(context, auth_context,
992             &ctx->acceptor_subkey);
993         if (code != 0) {
994             major_status = GSS_S_FAILURE;
995             goto fail;
996         }
997         code = (*kaccess.krb5int_c_mandatory_cksumtype)(context,
998             ctx->acceptor_subkey->enctype,
999             &ctx->acceptor_subkey_cksumtype);
1000     }
1001     if (code) {
1002         major_status = GSS_S_FAILURE;
1003         goto fail;
1004     }
1005     ctx->have_acceptor_subkey = 1;

1007     /* the reply token hasn't been sent yet, but that's ok. */
1008     ctx->gss_flags |= GSS_C_PROT_READY_FLAG;
1009     ctx->established = 1;

1011     token.length = g_token_size(mech_used, ap_rep.length);

1013     if ((token.value = (unsigned char *) xmalloc(token.length))
1014         == NULL) {
1015         major_status = GSS_S_FAILURE;
1016         code = ENOMEM;
1017         goto fail;
1018     }
1019     ptr3 = token.value;
1020     g_make_token_header(mech_used, ap_rep.length,
1021         &ptr3, KG_TOK_CTX_AP_REP);

1023     TWRITE_STR(ptr3, ap_rep.data, ap_rep.length);

1025     ctx->established = 1;

1027 } else {
1028     token.length = 0;
1029     token.value = NULL;
1030     ctx->seq_send = ctx->seq_recv;

1032     ctx->established = 1;
1033 }

1035 /* set the return arguments */

1037 if (src_name) {
1038     if ((code = krb5_copy_principal(context, ctx->there, &name))) {
1039         major_status = GSS_S_FAILURE;
1040         goto fail;
1041     }
1042     /* intern the src_name */
1043     if (! kg_save_name((gss_name_t) name)) {

```

```

1044     code = G_VALIDATE_FAILED;
1045     major_status = GSS_S_FAILURE;
1046     goto fail;
1047 }
1048 }

1050 if (mech_type)
1051     *mech_type = (gss_OID) mech_used;

1053 if (time_rec)
1054     *time_rec = ctx->endtime - now;

1056 if (ret_flags)
1057     *ret_flags = ctx->gss_flags;

1059 *context_handle = (gss_ctx_id_t)ctx;
1060 *output_token = token;

1062 if (src_name)
1063     *src_name = (gss_name_t) name;

1065 if (delegated_cred_handle && deleg_cred) {
1066     if (!kg_save_cred_id((gss_cred_id_t) deleg_cred)) {
1067         /* Solaris Kerberos */
1068         KRB5_LOG0(KRB5_ERR, "krb5_gss_accept_sec_context() "
1069                 "kg_save_cred_id() error");
1070         major_status = GSS_S_FAILURE;
1071         code = (OM_uint32) G_VALIDATE_FAILED;
1072         goto fail;
1073     }
1075     *delegated_cred_handle = (gss_cred_id_t) deleg_cred;
1076 }

1078 /* finally! */

1080 *minor_status = 0;
1081 major_status = GSS_S_COMPLETE;

1083 fail:
1084 if (authdat)
1085     krb5_free_authenticator(context, authdat);
1086 /* The ctx structure has the handle of the auth_context */
1087 if (auth_context && !ctx) {
1088     if (cred_rcache)
1089         (void)krb5_auth_con_setrcache(context, auth_context, NULL);

1091     krb5_auth_con_free(context, auth_context);
1092 }
1093 if (reqcksum.contents)
1094     xfree(reqcksum.contents);
1095 if (ap_rep.data)
1096     xfree(ap_rep.data);

1098 if (request != NULL) {
1099     saved_ap_options = request->ap_options;
1100     krb5_free_ap_req(context, request);
1101     request = NULL;
1102 }

1104 if (!GSS_ERROR(major_status) && major_status != GSS_S_CONTINUE_NEEDED) {
1105     if (!verifier_cred_handle && cred_handle) {
1106         krb5_gss_release_cred(minor_status, &cred_handle);
1107     }
1109     if (ctx)

```

```

1110     ctx->k5_context = context;

1112     return(major_status);
1113 }

1115 /* from here on is the real "fail" code */

1117 if (ctx)
1118     (void) krb5_gss_delete_sec_context(minor_status,
1119                                       (gss_ctx_id_t *) &ctx, NULL);
1120 if (deleg_cred) { /* free memory associated with the deleg credential */
1121     if (deleg_cred->ccache)
1122         (void)krb5_cc_close(context, deleg_cred->ccache);
1123     if (deleg_cred->princ)
1124         krb5_free_principal(context, deleg_cred->princ);
1125     xfree(deleg_cred);
1126 }
1127 if (token.value)
1128     xfree(token.value);
1129 if (name) {
1130     (void) kg_delete_name((gss_name_t) name);
1131     krb5_free_principal(context, name);
1132 }

1134 *minor_status = code;

1136 if (saved_ap_options & AP_OPTS_MUTUAL_REQUIRED)
1137     gss_flags |= GSS_C_MUTUAL_FLAG;

1139 if (cred
1140     && ((gss_flags & GSS_C_MUTUAL_FLAG)
1141         || (major_status == GSS_S_CONTINUE_NEEDED))) {
1142     unsigned int tmsglen;
1143     int toktype;

1145     /*
1146      * The client is expecting a response, so we can send an
1147      * error token back
1148      */
1149     memset(&krb_error_data, 0, sizeof(krb_error_data));

1151     code -= ERROR_TABLE_BASE_krb5;
1152     if (code < 0 || code > 128)
1153         code = 60 /* KRB_ERR_GENERIC */;

1155     krb_error_data.error = code;
1156     (void) krb5_us_timeofday(context, &krb_error_data.stime,
1157                             &krb_error_data.susec);
1158     krb_error_data.server = cred->princ;

1160     code = krb5_mk_error(context, &krb_error_data, &scratch);
1161     if (code)
1162         goto cleanup;

1164     tmsglen = scratch.length;
1165     toktype = KG_TOK_CTX_ERROR;

1167     token.length = g_token_size(mech_used, tmsglen);
1168     token.value = (unsigned char *) xmalloc(token.length);
1169     if (!token.value)
1170         goto cleanup;

1172     ptr = token.value;
1173     g_make_token_header(mech_used, tmsglen, &ptr, toktype);

1175     TWRITE_STR(ptr, scratch.data, scratch.length);

```

```
1176     xfree(scratch.data);
1177
1178     *output_token = token;
1179 }
1180
1181 cleanup:
1182     if (!verifier_cred_handle && cred_handle) {
1183         krb5_gss_release_cred(&minor_status, &cred_handle);
1184         krb5_gss_release_cred(minor_status, &cred_handle);
1185     }
1186     krb5_free_context(context);
1187
1188     /* Solaris Kerberos */
1189     KRB5_LOG(KRB5_ERR, "krb5_gss_accept_sec_context() end, "
1190             "major_status = %d", major_status);
1191     return (major_status);
1192 }
1193
1194 _____unchanged_portion_omitted_____
```

new/usr/src/lib/gss\_mechs/mech\_spnego/mech/spnego\_mech.c

1

```
*****
67495 Thu May  7 01:13:47 2009
new/usr/src/lib/gss_mechs/mech_spnego/mech/spnego_mech.c
6817447 libgss and various mechs are hiding both the real minor_status and the e
6405422 Solaris acceptors fail in AD-KDC environments when using non-"host" serv
6824434 Unable to accept context establishment initiated by Windows 2000 clients
6787343 kclient's site lookups fail in certain network environments
6692646 kclient should output errors to stderr
6525327 kinit failed when arcfour-hmac-md5-exp was used for the principal's key
6745582 SUNWkdcu missing package dependencies after kclintv2 integration
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[ ]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
24 * Copyright 2007 Sun Microsystems, Inc. All rights reserved.
25 * Use is subject to license terms.
26 *
27 * A module that implements the spnego security mechanism.
28 * It is used to negotiate the security mechanism between
29 * peers using the GSS-API.
30 */

31 #pragma ident      "%Z%M% %I%      %E% SMI"

32 #include <stdio.h>
33 #include <stdlib.h>
34 #include <errno.h>
35 #include "gssapiP_spnego.h"
36 #include <mechglueP.h>
37 #include <gssapi_err_generic.h>
38 #include <rpc/types.h>
39 #include <libintl.h>

41 /* der routines defined in libgss */
42 extern unsigned int der_length_size(OM_uint32);
43 extern int get_der_length(uchar_t **, OM_uint32, OM_uint32*);
44 extern int put_der_length(OM_uint32, uchar_t **, OM_uint32);

47 /* private routines for spnego_mechanism */
48 static spnego_token_t make_spnego_token(char *);
49 static gss_buffer_desc make_err_msg(char *);
50 static int g_token_size(gss_OID, OM_uint32);
51 static int g_make_token_header(gss_OID, int, uchar_t **, int);
52 static int g_verify_token_header(gss_OID, int *, uchar_t **, int, int);
```

new/usr/src/lib/gss\_mechs/mech\_spnego/mech/spnego\_mech.c

2

```
53 static int g_verify_neg_token_init(uchar_t **, int);
54 static OM_uint32 get_negResult(unsigned char **, int);
55 static gss_OID get_mech_oid(OM_uint32 *, uchar_t **, size_t);
56 static gss_buffer_t get_input_token(unsigned char **, int);
57 static gss_OID_set get_mech_set(OM_uint32 *, unsigned char **, int);
58 static OM_uint32 get_req_flags(uchar_t **, int *, OM_uint32 *);
59 static OM_uint32 get_available_mechs(OM_uint32 *, gss_name_t,
60     gss_cred_usage_t, gss_cred_id_t *, gss_OID_set *);
61 static void release_spnego_ctx(spnego_gss_ctx_id_t *);
62 static void check_spnego_options(spnego_gss_ctx_id_t);
63 static spnego_gss_ctx_id_t create_spnego_ctx(void);
64 static int put_mech_set(uchar_t **, gss_OID_set, int);
65 static int put_input_token(uchar_t **, gss_buffer_t, int);
66 static int put_mech_oid(uchar_t **, gss_OID_desc *, int);
67 static int put_negResult(uchar_t **, OM_uint32, int);

69 static gss_OID
70 negotiate_mech_type(OM_uint32 *, gss_OID_set, gss_OID_set,
71     OM_uint32 *, bool_t *);
72 static int
73 g_get_tag_and_length(unsigned char **, uchar_t, int, int *);

75 static int
76 make_spnego_tokenInit_msg(spnego_gss_ctx_id_t, gss_OID_set,
77     gss_buffer_t, send_token_flag,
78     gss_buffer_t);
79 static int
80 make_spnego_tokenTarg_msg(OM_uint32, gss_OID, gss_buffer_t,
81     gss_buffer_t, send_token_flag, int,
82     gss_buffer_t);

84 /*
85  * The Mech OID for SPNEGO:
86  * { iso(1) org(3) dod(6) internet(1) security(5)
87  *   mechanism(5) spnego(2) }
88  */
89 static struct gss_config spnego_mechanism =
90 {{SPNEGO_OID_LENGTH, SPNEGO_OID},
91     NULL,
92     spnego_gss_acquire_cred,
93     spnego_gss_release_cred,
94     spnego_gss_init_sec_context,
95     spnego_gss_accept_sec_context,
96 /* EXPORT DELETE START */ /* CRYPT DELETE START */
97     spnego_gss_unseal, /* gss_unseal */
98 /* EXPORT DELETE END */ /* CRYPT DELETE END */
99     NULL, /* gss_process_context_token */
100    spnego_gss_delete_sec_context, /* gss_delete_sec_context */
101    spnego_gss_context_time, /* gss_context_time */
102    spnego_gss_display_status,
103    NULL, /* gss_indicate_mechs */
104    NULL, /* gss_compare_name */
105    spnego_gss_display_name,
106    spnego_gss_import_name,
107    spnego_gss_release_name,
108    spnego_gss_inquire_cred, /* gss_inquire_cred */
109    NULL, /* gss_add_cred */
110 /* EXPORT DELETE START */ /* CRYPT DELETE START */
111    spnego_gss_seal, /* gss_seal */
112 /* EXPORT DELETE END */ /* CRYPT DELETE END */
113    spnego_gss_export_sec_context, /* gss_export_sec_context */
114    spnego_gss_import_sec_context, /* gss_import_sec_context */
115    NULL, /* gss_inquire_cred_by_mech */
116    spnego_gss_inquire_names_for_mech,
117    spnego_gss_inquire_context, /* gss_inquire_context */
118    NULL, /* gss_internal_release_oid */
```

```

119     spnego_gss_wrap_size_limit,      /* gss_wrap_size_limit */
120     NULL,                             /* gss_pname_to_uid */
121     NULL,                             /* __gss_userok */
122     NULL,                             /* gss_export_name */
123 /* EXPORT DELETE START */
124 /* CRYPT DELETE START */
125 #if 0
126 /* CRYPT DELETE END */
127     spnego_gss_seal,
128     spnego_gss_unseal,
129 /* CRYPT DELETE START */
130 #endif
131 /* CRYPT DELETE END */
132 /* EXPORT DELETE END */
133     spnego_gss_sign,                 /* gss_sign */
134     spnego_gss_verify,              /* gss_verify */
135     NULL,                           /* gss_store_cred */
136 };
    unchanged_portion_omitted

682 /*ARGSUSED*/
683 OM_uint32
684 spnego_gss_accept_sec_context(void *ct,
685                               OM_uint32 *minor_status,
686                               gss_ctx_id_t *context_handle,
687                               gss_cred_id_t verifier_cred_handle,
688                               gss_buffer_t input_token,
689                               gss_channel_bindings_t input_chan_bindings,
690                               gss_name_t *src_name,
691                               gss_OID *mech_type,
692                               gss_buffer_t output_token,
693                               OM_uint32 *ret_flags,
694                               OM_uint32 *time_rec,
695                               gss_cred_id_t *delegated_cred_handle)
696 {
697     spnego_gss_ctx_id_t spnego_ctx = NULL;
698     gss_OID mech_wanted = NULL;
699     gss_OID_set mechSet = GSS_C_NO_OID_SET;
700     gss_OID_set supported_mechSet = GSS_C_NO_OID_SET;
701     gss_buffer_t i_output_token = GSS_C_NO_BUFFER;
702     gss_buffer_t i_input_token = GSS_C_NO_BUFFER;
703     gss_buffer_t mechListMIC = GSS_C_NO_BUFFER;
704     gss_cred_id_t acquired_cred = NULL;
705     gss_name_t internal_name = GSS_C_NO_NAME;
706     OM_uint32 status = GSS_S_COMPLETE;
707     OM_uint32 ret = GSS_S_COMPLETE;
708     unsigned char *ptr;
709     unsigned char *bufstart;
710     int bodysize;
711     int err, len;
712     OM_uint32 negResult;
713     OM_uint32 minor_stat;
714     OM_uint32 mstat;
715     OM_uint32 req_flags;
716     OM_uint32 mechsetlen;
717     gss_qop_t qop_state;
718     send_token_flag return_token = NO_TOKEN_SEND;
719     bool_t firstMech;
720     bool_t Need_Cred = FALSE;
721     OM_uint32 local_ret_flags = 0;
722     uchar_t *buf, *tmp;

724     dsyslog("Entering accept_sec_context\n");

726     if (context_handle == NULL)
727         return (GSS_S_NO_CONTEXT);

```

```

729     if (src_name)
730         *src_name = (gss_name_t)NULL;

732     output_token->length = 0;
733     output_token->value = NULL;
734     *minor_status = 0;

736     if (mech_type)
737         *mech_type = GSS_C_NULL_OID;

739     /* return a bogus cred handle */
740     if (delegated_cred_handle)
741         *delegated_cred_handle = GSS_C_NO_CREDENTIAL;

743     if (verifier_cred_handle == GSS_C_NO_CREDENTIAL) {
744         Need_Cred = TRUE;
745     }

747     /* Check for defective input token. */
748     ptr = bufstart = (unsigned char *) input_token->value;
749     if (err = g_verify_token_header((gss_OID)gss_mech_spnego, &bodysize,
750                                   &ptr, 0, input_token->length)) {
751         *minor_status = err;
752         ret = GSS_S_DEFECTIVE_TOKEN;
753         negResult = REJECT;
754         return_token = ERROR_TOKEN_SEND;
755         goto sendererror;
756     }

758     /*
759      * set up of context, determine mech to be used, save mechset
760      * for use later in integrity check.
761      */
762     if (*context_handle == GSS_C_NO_CONTEXT) {
763         if ((spnego_ctx = create_spnego_ctx()) == NULL)
764             return (GSS_S_FAILURE);

766         /*
767          * Until the accept operation is complete, the
768          * context_handle returned should refer to
769          * the spnego context.
770          */
771         *context_handle = (gss_ctx_id_t)spnego_ctx;
772         minor_stat = get_available_mechs(minor_status,
773                                         GSS_C_NO_NAME, GSS_C_ACCEPT,
774                                         NULL, &supported_mechSet);

776         if (minor_stat != GSS_S_COMPLETE) {
777             release_spnego_ctx(&spnego_ctx);
778             *context_handle = GSS_C_NO_CONTEXT;
779             return (minor_stat);
780         }

782         if (Need_Cred) {
783             minor_stat = gss_acquire_cred(minor_status,
784                                         GSS_C_NO_NAME, NULL, supported_mechSet,
785                                         GSS_C_ACCEPT, &acquired_cred, NULL,
786                                         NULL);

788             if (minor_stat != GSS_S_COMPLETE) {
789                 (void) gss_release_oid_set(minor_status,
790                                           &supported_mechSet);
791                 release_spnego_ctx(&spnego_ctx);
792                 *context_handle = GSS_C_NO_CONTEXT;
793                 return (minor_stat);

```

```

794         } else {
795             verifier_cred_handle = acquired_cred;
796         }
797     }

799     if (err = g_verify_neg_token_init(&ptr, input_token->length)) {
800         *minor_status = err;
801         ret = GSS_S_DEFECTIVE_TOKEN;
802         negResult = REJECT;
803         return_token = ERROR_TOKEN_SEND;
804         goto senderror;
805     }

807     /*
808     * Allocate space to hold the mechTypes
809     * because we need it later.
810     */
811     mechsetlen = input_token->length - (ptr - bufstart);
812     buf = (uchar_t *)malloc(mechsetlen);
813     if (buf == NULL) {
814         ret = GSS_S_FAILURE;
815         goto cleanup;
816     }
817     (void) memcpy(buf, ptr, mechsetlen);
818     ptr = bufstart = buf;

820     /*
821     * Get pointers to the DER encoded MechSet so we
822     * can properly check and calculate a MIC later.
823     */
824     spnego_ctx->DER_mechTypes.value = ptr;
825     mechSet = get_mech_set(minor_status, &ptr, mechsetlen);
826     if (mechSet == NULL) {
827         ret = GSS_S_DEFECTIVE_TOKEN;
828         negResult = REJECT;
829         return_token = ERROR_TOKEN_SEND;
830         goto senderror;
831     }
832     spnego_ctx->DER_mechTypes.length = ptr - bufstart;
833     mechsetlen -= (ptr - bufstart);

835     /*
836     * Select the best match between the list of mechs
837     * that the initiator requested and the list that
838     * the acceptor will support.
839     */
840     mech_wanted = negotiate_mech_type(minor_status,
841         supported_mechSet, mechSet, &negResult,
842         &firstMech);

844     (void) gss_release_oid_set(&minor_stat, &supported_mechSet);
845     (void) gss_release_oid_set(&minor_stat, &mechSet);
846     supported_mechSet = NULL;
847     mechSet = NULL;

849     if (get_req_flags(&ptr, (int *)&mechsetlen, &req_flags) ==
850         ACCEPT_DEFECTIVE_TOKEN) {
851         negResult = REJECT;
852     }

854     tmp = ptr;
855     if (negResult == ACCEPT_COMPLETE) {
856         if (g_get_tag_and_length(&ptr, (CONTEXT | 0x02),
857             mechsetlen, &len) < 0) {
858             negResult = REJECT;
859         } else {

```

```

860             i_input_token = get_input_token(&ptr, len);
861             if (i_input_token == NULL) {
862                 negResult = REJECT;
863             }
864         }
865         return_token = INIT_TOKEN_SEND;
866     }
867     if (negResult == REJECT) {
868         ret = GSS_S_DEFECTIVE_TOKEN;
869         return_token = ERROR_TOKEN_SEND;
870     } else {
871         ret = GSS_S_CONTINUE_NEEDED;
872         return_token = INIT_TOKEN_SEND;
873     }

875     mechsetlen -= ptr - tmp;
876     /*
877     * Check to see if there is a MechListMIC field
878     */
879     if (negResult == ACCEPT_COMPLETE && mechsetlen > 0) {
880         tmp = ptr;
881         if (g_get_tag_and_length(&ptr, (CONTEXT | 0x03),
882             mechsetlen, &len) >= 0) {
883             mechListMIC = get_input_token(&ptr, len);
884             if (mechListMIC == GSS_C_NO_BUFFER) {
885                 negResult = REJECT;
886                 return_token = ERROR_TOKEN_SEND;
887                 ret = GSS_S_DEFECTIVE_TOKEN;
888             }
889             mechsetlen -= (ptr - tmp);
890         }
891     } else {
892         /*
893         * get internal input token and context for continued
894         * calls of spnego_gss_init_sec_context.
895         */
896         i_input_token = get_input_token(&ptr,
897             input_token->length - (ptr -
898                 (uchar_t *)input_token->value));
899         if (i_input_token == NULL) {
900             negResult = REJECT;
901             return_token = ERROR_TOKEN_SEND;
902             ret = GSS_S_DEFECTIVE_TOKEN;
903         } else {
904             spnego_ctx = (spnego_gss_ctx_id_t)(*context_handle);
905             return_token = CONT_TOKEN_SEND;
906         }
907     }
908 }

910     /*
911     * If we still don't have a cred, we have an error.
912     */
913     if (verifier_cred_handle == GSS_C_NO_CREDENTIAL) {
914         ret = GSS_S_FAILURE;
915         goto cleanup;
916     }

918     /* If we have an error already, bail out */
919     if (ret != GSS_S_COMPLETE && ret != GSS_S_CONTINUE_NEEDED)
920         goto senderror;

922     if (i_input_token != GSS_C_NO_BUFFER) {
923         i_output_token = (gss_buffer_t)malloc(sizeof (gss_buffer_desc));
924     }
925     if (i_output_token == NULL) {

```

```

926         ret = GSS_S_FAILURE;
927         goto cleanup;
928     }

930     i_output_token->length = 0;
931     i_output_token->value = NULL;

933     status = gss_accept_sec_context(&minor_stat,
934         &spnego_ctx->ctx_handle, verifier_cred_handle,
935         i_input_token, GSS_C_NO_CHANNEL_BINDINGS,
936         &internal_name, mech_type, i_output_token,
937         &local_ret_flags, time_rec, delegated_cred_handle);

939     if ((status != GSS_S_COMPLETE) &&
940         (status != GSS_S_CONTINUE_NEEDED)) {
941         *minor_status = minor_stat;
942         (void) gss_release_buffer(&mstat, i_input_token);

944         if (i_input_token != GSS_C_NO_BUFFER) {
945             free(i_input_token);
946             i_input_token = GSS_C_NO_BUFFER;
947         }

949         ret = status;

951         /*
952          * Reject the request with an error token.
953          */
954         negResult = REJECT;
955         return_token = ERROR_TOKEN_SEND;

957         goto senderror;
958     }

960     if (ret_flags)
961         *ret_flags = local_ret_flags;

963     if (i_input_token != GSS_C_NO_BUFFER) {
964         (void) gss_release_buffer(&mstat, i_input_token);
965         free(i_input_token);
966         i_input_token = GSS_C_NO_BUFFER;
967     }

969     /* If we got a MIC, verify it if possible */
970     if ((status == GSS_S_COMPLETE) &&
971         (local_ret_flags & GSS_C_INTEG_FLAG) &&
972         mechListMIC != GSS_C_NO_BUFFER &&
973         !spnego_ctx->MS_Interop) {

975         ret = gss_verify_mic(minor_status,
976             spnego_ctx->ctx_handle,
977             &spnego_ctx->DER_mechTypes,
978             mechListMIC, &qop_state);

980         (void) gss_release_buffer(&mstat, mechListMIC);
981         free(mechListMIC);
982         mechListMIC = GSS_C_NO_BUFFER;

984         if (ret != GSS_S_COMPLETE) {
985             negResult = REJECT;
986             return_token = ERROR_TOKEN_SEND;
987             goto senderror;
988         }
989     }

991     /*

```

```

992     * If the MIC was verified OK, create a new MIC
993     * for the response message.
994     */
995     if (status == GSS_S_COMPLETE &&
996         (local_ret_flags & GSS_C_INTEG_FLAG) &&
997         !spnego_ctx->MS_Interop) {
998         mechListMIC = (gss_buffer_t)
999             malloc(sizeof (gss_buffer_desc));

1001         if (mechListMIC == NULL ||
1002             spnego_ctx->DER_mechTypes.length == 0) {
1003             ret = GSS_S_FAILURE;
1004             goto cleanup;
1005         }

1007         ret = gss_get_mic(minor_status,
1008             spnego_ctx->ctx_handle,
1009             GSS_C_QOP_DEFAULT,
1010             &spnego_ctx->DER_mechTypes,
1011             mechListMIC);

1013         if (ret != GSS_S_COMPLETE) {
1014             negResult = REJECT;
1015             return_token = ERROR_TOKEN_SEND;
1016             goto senderror;
1017         }
1018     }
1019     ret = status;

1021     if (status == GSS_S_COMPLETE) {
1022         if (internal_name != NULL && src_name != NULL)
1023             *src_name = internal_name;
1024     }

1027     if (status == GSS_S_CONTINUE_NEEDED) {
1028         if (return_token == INIT_TOKEN_SEND)
1029             negResult = ACCEPT_INCOMPLETE;
1030     }
1031 }

1033 senderror:
1034     if ((return_token == INIT_TOKEN_SEND) ||
1035         (return_token == CONT_TOKEN_SEND) ||
1036         (return_token == ERROR_TOKEN_SEND)) {
1037         int MS_Interop = 0;

1039         if (spnego_ctx)
1040             MS_Interop = spnego_ctx->MS_Interop;

1042         /*
1043          * create response for the initiator.
1044          */
1045         err = make_spnego_tokenTarg_msg(negResult,
1046             mech_wanted, i_output_token,
1047             mechListMIC, return_token,
1048             MS_Interop, output_token);

1050         (void) gss_release_buffer(&mstat, mechListMIC);
1051         free(mechListMIC);

1053         /*
1054          * If we could not make the response token,
1055          * we will have to fail without sending a response.
1056          */
1057         if (err) {

```

```
1058             (void) gss_release_buffer(&mstat, output_token);
1059         }
1060     } else {
1061         (void) gss_release_buffer(&mstat, output_token);
1062     }

1064 cleanup:
1065     if (ret != GSS_S_COMPLETE &&
1066         ret != GSS_S_CONTINUE_NEEDED) {
1067         if (spnego_ctx != NULL) {
1068             (void) gss_delete_sec_context(&mstat,
1069                 &spnego_ctx->ctx_handle, NULL);
1071             spnego_ctx->ctx_handle = NULL;

1073             release_spnego_ctx(&spnego_ctx);
1074         }
1075         *context_handle = GSS_C_NO_CONTEXT;
1076     }
1077     if (mech_wanted != NULL) {
1078         generic_gss_release_oid(&mstat, &mech_wanted);
1079     }

1081     (void) gss_release_cred(&mstat, &acquired_cred);
1082     (void) gss_release_oid_set(&mstat, &supported_mechSet);
1083     (void) gss_release_cred(minor_status, &acquired_cred);
1084     (void) gss_release_oid_set(minor_status, &supported_mechSet);

1084     (void) gss_release_buffer(&mstat, i_output_token);
1085     free(i_output_token);

1087     return (ret);
1088 }
_____unchanged_portion_omitted_____
```

new/usr/src/lib/libgss/g\_accept\_sec\_context.c

1

```
*****
8674 Thu May 7 01:13:49 2009
new/usr/src/lib/libgss/g_accept_sec_context.c
6817447 libgss and various mechs are hiding both the real minor_status and the e
6405422 Solaris acceptors fail in AD-KDC environments when using non-"host" serv
6824434 Unable to accept context establishment initiated by Windows 2000 clients
6787343 kclient's site lookups fail in certain network environments
6692646 kclient should output errors to stderr
6525327 kinit failed when arcfour-hmac-md5-exp was used for the principal's key
6745582 SUNWkdcu missing package dependencies after kclientv2 integration
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  * Common Development and Distribution License, Version 1.0 only
8  * (the "License"). You may not use this file except in compliance
9  * with the License.
10 *
11 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
12 * or http://www.opensolaris.org/os/licensing.
13 * See the License for the specific language governing permissions
14 * and limitations under the License.
15 *
16 * When distributing Covered Code, include this CDDL HEADER in each
17 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
18 * If applicable, add the following below this CDDL HEADER, with the
19 * fields enclosed by brackets "[]" replaced with your own identifying
20 * information: Portions Copyright [yyyy] [name of copyright owner]
21 *
22 * CDDL HEADER END
23 */
24
25 /*
26  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
27  * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
28  * Use is subject to license terms.
29 */
30
31 #pragma ident "%Z%M% %I% %E% SMI"
32
33 /*
34  * glue routine for gss_accept_sec_context
35 */
36
37 #include <mechglueP.h>
38 #ifdef HAVE_STDLIB_H
39 #include <stdlib.h>
40 #endif
41 #include <string.h>
42 #include <errno.h>
43
44 OM_uint32
45 gss_accept_sec_context(minor_status,
46                       context_handle,
47                       verifier_cred_handle,
48                       input_token_buffer,
49                       input_chan_bindings,
50                       src_name,
51                       mech_type,
52                       output_token,
53                       ret_flags,
54                       time_rec,
55                       d_cred)
```

new/usr/src/lib/libgss/g\_accept\_sec\_context.c

2

```
51 OM_uint32 *minor_status;
52 gss_ctx_id_t *context_handle;
53 const gss_cred_id_t verifier_cred_handle;
54 const gss_buffer_t input_token_buffer;
55 const gss_channel_bindings_t input_chan_bindings;
56 gss_name_t *src_name;
57 gss_OID *mech_type;
58 gss_buffer_t output_token;
59 OM_uint32 *ret_flags;
60 OM_uint32 *time_rec;
61 gss_cred_id_t *d_cred; /* delegated cred handle */
62
63 {
64     OM_uint32 status, temp_status, t_minstat;
65     gss_union_ctx_id_t union_ctx_id;
66     gss_union_cred_t union_cred;
67     gss_cred_id_t input_cred_handle = GSS_C_NO_CREDENTIAL;
68     gss_cred_id_t tmp_d_cred = GSS_C_NO_CREDENTIAL;
69     gss_name_t internal_name = GSS_C_NO_NAME;
70     gss_name_t tmp_src_name = GSS_C_NO_NAME;
71     gss_OID_desc token_mech_type_desc;
72     gss_OID token_mech_type = &token_mech_type_desc;
73     gss_OID actual_mech = GSS_C_NO_OID;
74     OM_uint32 flags;
75     gss_mechanism mech;
76
77     /* check parameters first */
78     if (minor_status == NULL)
79         return (GSS_S_CALL_INACCESSIBLE_WRITE);
80     *minor_status = 0;
81
82     if (context_handle == NULL || output_token == NULL)
83         return (GSS_S_CALL_INACCESSIBLE_WRITE);
84
85     /* clear optional fields */
86     output_token->value = NULL;
87     output_token->length = 0;
88     if (src_name)
89         *src_name = NULL;
90
91     if (mech_type)
92         *mech_type = NULL;
93
94     if (d_cred)
95         *d_cred = NULL;
96     /*
97      * if context_handle is GSS_C_NO_CONTEXT, allocate a union context
98      * descriptor to hold the mech type information as well as the
99      * underlying mechanism context handle. Otherwise, cast the
100     * value of *context_handle to the union context variable.
101     */
102
103     if (*context_handle == GSS_C_NO_CONTEXT) {
104
105         if (GSS_EMPTY_BUFFER(input_token_buffer))
106             return (GSS_S_CALL_INACCESSIBLE_READ);
107
108         /* Get the token mech type */
109         status = __gss_get_mech_type(token_mech_type,
110                                     input_token_buffer);
111
112         if (status)
113             return (status);
114
115         status = GSS_S_FAILURE;
```

```

116     union_ctx_id = (gss_union_ctx_id_t)
117         malloc(sizeof (gss_union_ctx_id_desc));
118     if (!union_ctx_id)
119         return (GSS_S_FAILURE);
120
121     union_ctx_id->internal_ctx_id = GSS_C_NO_CONTEXT;
122     status = generic_gss_copy_oid(&t_minstat,
123                                 token_mech_type,
124                                 &union_ctx_id->mech_type);
125     if (status != GSS_S_COMPLETE) {
126         free(union_ctx_id);
127         return (status);
128     }
129
130     /* set the new context handle to caller's data */
131     *context_handle = (gss_ctx_id_t)union_ctx_id;
132 } else {
133     union_ctx_id = (gss_union_ctx_id_t)*context_handle;
134     token_mech_type = union_ctx_id->mech_type;
135 }
136
137 /*
138  * get the appropriate cred handle from the union cred struct.
139  * defaults to GSS_C_NO_CREDENTIAL if there is no cred, which will
140  * use the default credential.
141  */
142 union_cred = (gss_union_cred_t)verifier_cred_handle;
143 input_cred_handle = __gss_get_mechanism_cred(union_cred,
144                                             token_mech_type);
145
146 /*
147  * now select the appropriate underlying mechanism routine and
148  * call it.
149  */
150
151 mech = __gss_get_mechanism(token_mech_type);
152 if (mech && mech->gss_accept_sec_context) {
153     status = mech->gss_accept_sec_context(
154         mech->context,
155         minor_status,
156         &union_ctx_id->internal_ctx_id,
157         input_cred_handle,
158         input_token_buffer,
159         input_chan_bindings,
160         &internal_name,
161         &actual_mech,
162         output_token,
163         &flags,
164         time_rec,
165         d_cred ? &tmp_d_cred : NULL);
166
167     /* If there's more work to do, keep going... */
168     if (status == GSS_S_CONTINUE_NEEDED)
169         return (GSS_S_CONTINUE_NEEDED);
170
171     /* if the call failed, return with failure */
172     if (status != GSS_S_COMPLETE)
173         goto error_out;
174
175     if (mech_type != NULL)
176         *mech_type = actual_mech;
177
178     /*
179      * if src_name is non-NULL,
180      * convert internal_name into a union name equivalent
181      * First call the mechanism specific display_name()

```

```

182     * then call gss_import_name() to create
183     * the union name struct cast to src_name
184     */
185     if (internal_name != NULL) {
186         temp_status = __gss_convert_name_to_union_name(
187             &t_minstat, mech,
188             internal_name, &tmp_src_name);
189         if (temp_status != GSS_S_COMPLETE) {
190             *minor_status = t_minstat;
191             if (output_token->length)
192                 (void) gss_release_buffer(
193                     &t_minstat,
194                     output_token);
195             if (internal_name != GSS_C_NO_NAME)
196                 mech->gss_release_name(
197                     mech->context,
198                     &t_minstat,
199                     &internal_name);
200             return (temp_status);
201         }
202         if (src_name != NULL) {
203             *src_name = tmp_src_name;
204         }
205     } else if (src_name != NULL) {
206         *src_name = GSS_C_NO_NAME;
207     }
208
209     /* Ensure we're returning correct creds format */
210     if ((flags & GSS_C_DELEG_FLAG) &&
211         tmp_d_cred != GSS_C_NO_CREDENTIAL) {
212         /*
213          * If we got back an OID different from the original
214          * token OID, assume the delegated_cred is already
215          * a proper union_cred and just return it. Don't
216          * try to re-wrap it. This is for SPNEGO or other
217          * pseudo-mechanisms.
218          */
219         if (actual_mech != GSS_C_NO_OID &&
220             token_mech_type != GSS_C_NO_OID &&
221             !g_OID_equal(actual_mech, token_mech_type)) {
222             *d_cred = tmp_d_cred;
223         } else {
224             gss_union_cred_t d_u_cred = NULL;
225
226             d_u_cred = malloc(sizeof (gss_union_cred_desc));
227             if (d_u_cred == NULL) {
228                 status = GSS_S_FAILURE;
229                 goto error_out;
230             }
231             (void) memset(d_u_cred, 0,
232                 sizeof (gss_union_cred_desc));
233
234             d_u_cred->count = 1;
235
236             status = generic_gss_copy_oid(
237                 &t_minstat,
238                 actual_mech,
239                 &d_u_cred->mechs_array);
240
241             if (status != GSS_S_COMPLETE) {
242                 free(d_u_cred);
243                 goto error_out;
244             }
245
246             d_u_cred->cred_array = malloc(
247                 sizeof (gss_cred_id_t));

```

```

248     if (d_u_cred->cred_array != NULL) {
249         d_u_cred->cred_array[0] = tmp_d_cred;
250     } else {
251         free(d_u_cred);
252         status = GSS_S_FAILURE;
253         goto error_out;
254     }
255
256     if (status != GSS_S_COMPLETE) {
257         free(d_u_cred->cred_array);
258         free(d_u_cred);
259         goto error_out;
260     }
261
262     internal_name = GSS_C_NO_NAME;
263
264     d_u_cred->auxinfo.creation_time = time(0);
265     d_u_cred->auxinfo.time_rec = 0;
266
267     if (mech->gss_inquire_cred) {
268         status = mech->gss_inquire_cred(
269             mech->context,
270             minor_status,
271             tmp_d_cred,
272             &internal_name,
273             &d_u_cred->auxinfo.time_rec,
274             &d_u_cred->auxinfo.cred_usage,
275             NULL);
276     }
277
278     if (internal_name != NULL) {
279         temp_status =
280             __gss_convert_name_to_union_name(
281                 &t_minstat, mech,
282                 internal_name, &tmp_src_name);
283         if (temp_status != GSS_S_COMPLETE) {
284             *minor_status = t_minstat;
285             if (output_token->length)
286                 (void) gss_release_buffer(
287                     &t_minstat,
288                     output_token);
289             free(d_u_cred->cred_array);
290             free(d_u_cred);
291             return (temp_status);
292         }
293     }
294
295     if (tmp_src_name != NULL) {
296         status = gss_display_name(
297             &t_minstat,
298             tmp_src_name,
299             &d_u_cred->auxinfo.name,
300             &d_u_cred->auxinfo.name_type);
301     }
302
303     *d_cred = (gss_cred_id_t)d_u_cred;
304 }
305
306 if (ret_flags != NULL) {
307     *ret_flags = flags;
308 }
309
310 if (src_name == NULL && tmp_src_name != NULL)
311     (void) gss_release_name(&t_minstat,
312                             &tmp_src_name);
313 return (status);

```

```

314     } else {
315
316         status = GSS_S_BAD_MECH;
317     }
318
319 error_out:
320     if (union_ctx_id) {
321         if (union_ctx_id->mech_type) {
322             if (union_ctx_id->mech_type->elements)
323                 free(union_ctx_id->mech_type->elements);
324             free(union_ctx_id->mech_type);
325         }
326         free(union_ctx_id);
327         *context_handle = GSS_C_NO_CONTEXT;
328     }
329
330     if (output_token->length)
331         (void) gss_release_buffer(&t_minstat, output_token);
332
333     if (src_name)
334         *src_name = GSS_C_NO_NAME;
335
336     if (tmp_src_name != GSS_C_NO_NAME)
337         (void) gss_release_buffer(&t_minstat,
338                                 (gss_buffer_t)tmp_src_name);
339
340     return (status);
341 }

```

unchanged\_portion\_omitted

```

*****
1935 Thu May 7 01:13:50 2009
new/usr/src/pkgdefs/SUNWkdcu/depend
6817447 libgss and various mechs are hiding both the real minor_status and the e
6405422 Solaris acceptors fail in AD-KDC environments when using non-"host" serv
6824434 Unable to accept context establishment initiated by Windows 2000 clients
6787343 kclient's site lookups fail in certain network environments
6692646 kclient should output errors to stderr
6525327 kinit failed when arcfour-hmac-md5-exp was used for the principal's key
6745582 SUNWkdcu missing package dependencies after kclientv2 integration
*****
1 #
2 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
2 # Copyright 2005 Sun Microsystems, Inc. All rights reserved.
3 # Use is subject to license terms.
4 #
5 # CDDL HEADER START
6 #
7 # The contents of this file are subject to the terms of the
8 # Common Development and Distribution License (the "License").
9 # You may not use this file except in compliance with the License.
8 # Common Development and Distribution License, Version 1.0 only
9 # (the "License"). You may not use this file except in compliance
10 # with the License.
10 #
11 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
12 # or http://www.opensolaris.org/os/licensing.
13 # See the License for the specific language governing permissions
14 # and limitations under the License.
15 #
16 # When distributing Covered Code, include this CDDL HEADER in each
17 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
18 # If applicable, add the following below this CDDL HEADER, with the
19 # fields enclosed by brackets "[]" replaced with your own identifying
20 # information: Portions Copyright [yyyy] [name of copyright owner]
21 #
22 # CDDL HEADER END
23 #
25 # ident "%Z%M% %I% %E% SMI"
26 #
24 # This package information file defines software dependencies associated
25 # with the pkg. You can define three types of pkg dependencies with this file:
26 # P indicates a prerequisite for installation
27 # I indicates an incompatible package
28 # R indicates a reverse dependency
29 # <pkg.abbr> see pkginfo(4), PKG parameter
30 # <name> see pkginfo(4), NAME parameter
31 # <version> see pkginfo(4), VERSION parameter
32 # <arch> see pkginfo(4), ARCH parameter
33 # <type> <pkg.abbr> <name>
34 # (<arch><version>
35 # (<arch><version>
36 # ...
37 # <type> <pkg.abbr> <name>
38 # ...
39 #

41 P SUNWcar Core Architecture, (Root)
42 P SUNWcakr Core Solaris Kernel Architecture (Root)
43 P SUNWkvm Core Architecture, (Kvm)
44 P SUNWcsr Core Solaris, (Root)
45 P SUNWckr Core Solaris Kernel (Root)
46 P SUNWcnetr Core Solaris Network Infrastructure (Root)
47 P SUNWcsu Core Solaris, (Usr)
48 P SUNWcsd Core Solaris Devices
49 P SUNWcsl Core Solaris Libraries

```

```

50 P SUNWrsg RPCSEC_GSS
51 P SUNWgss Generic Security Service App Program Int, Ver 2 - user
52 P SUNWlldap LDAP Libraries
53 P SUNWsmbsu SMB Server (Usr)

```

```

*****
74535 Thu May 7 01:13:51 2009
new/usr/src/uts/common/gssapi/gssd_clnt_stubs.c
6817447 libgss and various mechs are hiding both the real minor_status and the e
6405422 Solaris acceptors fail in AD-KDC environments when using non-"host" serv
6824434 Unable to accept context establishment initiated by Windows 2000 clients
6787343 kclient's site lookups fail in certain network environments
6692646 kclient should output errors to stderr
6525327 kinit failed when arcfour-hmac-md5-exp was used for the principal's key
6745582 SUNWkdcu missing package dependencies after kclientv2 integration
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
24 * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
25 * Use is subject to license terms.
26 */

26 #pragma ident "%Z%M% %I% %E% SMI"

27 /*
28 * GSSAPI library stub module for gssd.
29 */

31 #include <mechglueP.h>
32 #include "gssd_prot.h"
33 #include <rpc/rpc.h>

35 #include <sys/system.h>
36 #include <sys/types.h>
37 #include <sys/cmn_err.h>
38 #include <sys/kmem.h>
39 #include <gssapi/kgssapi_defs.h>
40 #include <sys/debug.h>

42 #ifdef GSSDEBUG
43 /*
44 * Kernel kgssd module debugging aid. The global variable "gss_log"
45 * is a bit mask which allows various types of debugging messages
46 * to be printed out.
47 *
48 * gss_log & 1 will cause actual failures to be printed.
49 * gss_log & 2 will cause informational messages to be
50 * printed on the client side of kgssd.
51 * gss_log & 4 will cause informational messages to be
52 * printed on the server side of kgssd.

```

```

53 * gss_log & 8 will cause informational messages to be
54 * printed on both client and server side of kgssd.
55 */

57 uint_t gss_log = 1;

59 #endif /* GSSDEBUG */

61 #ifdef DEBUG
62 extern void prom_printf();
63 #endif

65 char *server = "localhost";

67 static OM_uint32 kgss_sign_wrapped(void *, OM_uint32 *, gss_ctx_id_t, int,
68 gss_buffer_t, gss_buffer_t, OM_uint32);

70 static OM_uint32 kgss_verify_wrapped(void *, OM_uint32 *, gss_ctx_id_t,
71 gss_buffer_t, gss_buffer_t, int *qop_state, OM_uint32);

73 /* EXPORT DELETE START */
74 static OM_uint32 kgss_seal_wrapped(void *, OM_uint32 *, gss_ctx_id_t,
75 int, int, gss_buffer_t, int *, gss_buffer_t, OM_uint32);

77 static OM_uint32 kgss_unseal_wrapped(void *, OM_uint32 *, gss_ctx_id_t,
78 gss_buffer_t, gss_buffer_t, int *conf_state, int *qop_state,
79 OM_uint32);
80 /* EXPORT DELETE END */

82 static OM_uint32 kgss_delete_sec_context_wrapped(void *, OM_uint32 *,
83 gssd_ctx_id_t *, gssd_buffer_t, OM_uint32);

85 static void __kgss_reset_mech(gss_mechanism *, gss_OID);

87 #define DEFAULT_MINOR_STAT ((OM_uint32) ~0)

89 OM_uint32
90 kgss_acquire_cred_wrapped(minor_status,
91 desired_name,
92 time_req,
93 desired_mechs,
94 cred_usage,
95 output_cred_handle,
96 actual_mechs,
97 time_rec,
98 uid,
99 gssd_cred_verifier)
100 OM_uint32 *minor_status;
101 const gss_name_t desired_name;
102 OM_uint32 time_req;
103 const gss_OID_set desired_mechs;
104 int cred_usage;
105 gssd_cred_id_t *output_cred_handle;
106 gss_OID_set *actual_mechs;
107 OM_uint32 *time_rec;
108 uid_t uid;
109 OM_uint32 *gssd_cred_verifier;
110 {
111 CLIENT *clnt;

113 OM_uint32 minor_status_temp;
114 gss_buffer_desc external_name;
115 gss_OID name_type;
116 enum clnt_stat client_stat;
117 int i;

```

```

119     gss_acquire_cred_arg arg;
120     gss_acquire_cred_res res;

122     /* get the client handle to GSSD */

124     if ((clnt = getgssd_handle()) == NULL) {
125         GSSLOG(1, "kgss_acquire_cred: can't connect to server on %s\n",
126             server);
127         return (GSS_S_FAILURE);
128     }

130     /* convert the desired name from internal to external format */

132     if (gss_display_name(&minor_status_temp, desired_name, &external_name,
133         &name_type) != GSS_S_COMPLETE) {

135         *minor_status = (OM_uint32) minor_status_temp;
136         killgssd_handle(clnt);
137         GSSLOG(1, "kgss_acquire_cred: display name failed\n");
138         return ((OM_uint32) GSS_S_FAILURE);
139     }

142     /* copy the procedure arguments into the rpc arg parameter */

144     arg.uid = (OM_uint32) uid;

146     arg.desired_name.GSS_BUFFER_T_len = (uint_t)external_name.length;
147     arg.desired_name.GSS_BUFFER_T_val = (char *)external_name.value;

149     arg.name_type.GSS_OID_len =
150         name_type == GSS_C_NULL_OID ?
151         0 : (uint_t)name_type->length;

153     arg.name_type.GSS_OID_val =
154         name_type == GSS_C_NULL_OID ?
155         (char *)NULL : (char *)name_type->elements;

157     arg.time_req = time_req;

159     if (desired_mechs != GSS_C_NULL_OID_SET) {
160         arg.desired_mechs.GSS_OID_SET_len =
161             (uint_t)desired_mechs->count;
162         arg.desired_mechs.GSS_OID_SET_val = (GSS_OID *)
163             MALLOC(sizeof (GSS_OID) * desired_mechs->count);

165         for (i = 0; i < desired_mechs->count; i++) {
166             arg.desired_mechs.GSS_OID_SET_val[i].GSS_OID_len =
167                 (uint_t)desired_mechs->elements[i].length;
168             arg.desired_mechs.GSS_OID_SET_val[i].GSS_OID_val =
169                 (char *)MALLOC(desired_mechs->elements[i].length);
170             (void) memcpy(
171                 arg.desired_mechs.GSS_OID_SET_val[i].GSS_OID_val,
172                 desired_mechs->elements[i].elements,
173                 desired_mechs->elements[i].length);
174         }
175     } else
176         arg.desired_mechs.GSS_OID_SET_len = 0;

178     arg.cred_usage = cred_usage;

180     /* call the remote procedure */

182     bzero((caddr_t)&res, sizeof (res));
183     client_stat = gss_acquire_cred_1(&arg, &res, clnt);

```

```

185     (void) gss_release_buffer(&minor_status_temp, &external_name);
186     if (desired_mechs != GSS_C_NULL_OID_SET) {
187         for (i = 0; i < desired_mechs->count; i++)
188             FREE(arg.desired_mechs.GSS_OID_SET_val[i].GSS_OID_val,
189                 arg.desired_mechs.GSS_OID_SET_val[i].GSS_OID_len);
190         FREE(arg.desired_mechs.GSS_OID_SET_val,
191             arg.desired_mechs.GSS_OID_SET_len * sizeof (GSS_OID));
192     }

194     if (client_stat != RPC_SUCCESS) {

196         /*
197          * if the RPC call times out, null out all return arguments,
198          * set minor_status to its maximum value, and return
199          * GSS_S_FAILURE
200          */

202         if (minor_status != NULL)
203             *minor_status = DEFAULT_MINOR_STAT;
204         if (output_cred_handle != NULL)
205             *output_cred_handle = NULL;
206         if (actual_mechs != NULL)
207             *actual_mechs = NULL;
208         if (time_rec != NULL)
209             *time_rec = 0;

211         killgssd_handle(clnt);
212         GSSLOG(1, "kgss_acquire_cred: RPC call times out\n");
213         return (GSS_S_FAILURE);
214     }

216     /* copy the rpc results into the return arguments */

218     if (minor_status != NULL)
219         *minor_status = res.minor_status;

221     if (output_cred_handle != NULL &&
222         (res.status == GSS_S_COMPLETE)) {
223         *output_cred_handle =
224             *((gssd_cred_id_t *)res.output_cred_handle.GSS_CRED_ID_T_val);
225         *gssd_cred_verifier = res.gssd_cred_verifier;
226     }

228     if (res.status == GSS_S_COMPLETE &&
229         res.actual_mechs.GSS_OID_SET_len != 0 &&
230         actual_mechs != NULL) {
231         *actual_mechs = (gss_OID_set) MALLOC(sizeof (gss_OID_set_desc));
232         (*actual_mechs)->count =
233             (int)res.actual_mechs.GSS_OID_SET_len;
234         (*actual_mechs)->elements = (gss_OID)
235             MALLOC(sizeof (gss_OID_desc) * (*actual_mechs)->count);

237         for (i = 0; i < (*actual_mechs)->count; i++) {
238             (*actual_mechs)->elements[i].length = (OM_uint32)
239                 res.actual_mechs.GSS_OID_SET_val[i].GSS_OID_len;
240             (*actual_mechs)->elements[i].elements =
241                 (void *) MALLOC((*actual_mechs)->elements[i].length);
242             (void) memcpy((*actual_mechs)->elements[i].elements,
243                 res.actual_mechs.GSS_OID_SET_val[i].GSS_OID_val,
244                 (*actual_mechs)->elements[i].length);
245         }
246     } else {
247         if (res.status == GSS_S_COMPLETE &&
248             actual_mechs != NULL)
249             (*actual_mechs) = NULL;
250     }

```

```

252     if (time_rec != NULL)
253         *time_rec = res.time_rec;

255     /*
256     * free the memory allocated for the results and return with the status
257     * received in the rpc call
258     */

260     clnt_freeres(clnt, xdr_gss_acquire_cred_res, (caddr_t)&res);
261     killgssd_handle(clnt);
262     return (res.status);

264 }
    unchanged portion omitted

610 static OM_uint32
611 kgss_init_sec_context_wrapped(
612     OM_uint32 *minor_status,
613     const gssd_cred_id_t claimant_cred_handle,
614     OM_uint32 gssd_cred_verifier,
615     gssd_ctx_id_t *context_handle,
616     OM_uint32 *gssd_context_verifier,
617     const gss_name_t target_name,
618     const gss_OID mech_type,
619     int req_flags,
620     OM_uint32 time_req,
621     const gss_channel_bindings_t input_chan_bindings,
622     const gss_buffer_t input_token,
623     gss_OID *actual_mech_type,
624     gss_buffer_t output_token,
625     int *ret_flags,
626     OM_uint32 *time_rec,
627     uid_t uid)
628 {
629     CLIENT *clnt;

631     OM_uint32     minor_status_temp;
632     gss_buffer_desc external_name;
633     gss_OID      name_type;

635     gss_init_sec_context_arg arg;
636     gss_init_sec_context_res res;

638     /* get the client handle to GSSD */

640     if ((clnt = getgssd_handle()) == NULL) {
641         GSSLOG(1,
642             "kgss_init_sec_context: can't connect to server on %s\n",
643             server);
644         return (GSS_S_FAILURE);
645     }

647     /* convert the target name from internal to external format */

649     if (gss_display_name(&minor_status_temp, target_name,
650         &external_name, &name_type) != GSS_S_COMPLETE) {

652         *minor_status = (OM_uint32) minor_status_temp;
653         killgssd_handle(clnt);
654         GSSLOG(1, "kgss_init_sec_context: can't display name\n");
655         return ((OM_uint32) GSS_S_FAILURE);
656     }

659     /* copy the procedure arguments into the rpc arg parameter */

```

```

661     arg.uid = (OM_uint32)uid;

663     arg.context_handle.GSS_CTX_ID_T_len =
664         *context_handle == (gssd_ctx_id_t)GSS_C_NO_CONTEXT ?
665         0 : (uint_t)sizeof (gssd_ctx_id_t);
666     arg.context_handle.GSS_CTX_ID_T_val = (char *)context_handle;

668     arg.gssd_context_verifier = *gssd_context_verifier;

670     arg.claimant_cred_handle.GSS_CRED_ID_T_len =
671         claimant_cred_handle == (gssd_cred_id_t)GSS_C_NO_CREDENTIAL ?
672         0 : (uint_t)sizeof (gssd_cred_id_t);
673     arg.claimant_cred_handle.GSS_CRED_ID_T_val =
674         (char *)&claimant_cred_handle;
675     arg.gssd_cred_verifier = gssd_cred_verifier;

677     arg.target_name.GSS_BUFFER_T_len = (uint_t)external_name.length;
678     arg.target_name.GSS_BUFFER_T_val = (char *)external_name.value;

680     arg.name_type.GSS_OID_len =
681         name_type == GSS_C_NULL_OID ?
682         0 : (uint_t)name_type->length;

684     arg.name_type.GSS_OID_val =
685         name_type == GSS_C_NULL_OID ?
686         (char *)NULL : (char *)name_type->elements;

688     arg.mech_type.GSS_OID_len = (uint_t)(mech_type != GSS_C_NULL_OID ?
689         mech_type->length : 0);
690     arg.mech_type.GSS_OID_val = (char *) (mech_type != GSS_C_NULL_OID ?
691         mech_type->elements : 0);

693     arg.req_flags = req_flags;

695     arg.time_req = time_req;

697     if (input_chan_bindings != GSS_C_NO_CHANNEL_BINDINGS) {
698         arg.input_chan_bindings.present = YES;
699         arg.input_chan_bindings.initiator_addrtype =
700             input_chan_bindings->initiator_addrtype;
701         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_len =
702             (uint_t)input_chan_bindings->initiator_address.length;
703         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_val =
704             (void *)input_chan_bindings->initiator_address.value;
705         arg.input_chan_bindings.acceptor_addrtype =
706             input_chan_bindings->acceptor_addrtype;
707         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_len =
708             (uint_t)input_chan_bindings->acceptor_address.length;
709         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_val =
710             (void *)input_chan_bindings->acceptor_address.value;
711         arg.input_chan_bindings.application_data.GSS_BUFFER_T_len =
712             (uint_t)input_chan_bindings->application_data.length;
713         arg.input_chan_bindings.application_data.GSS_BUFFER_T_val =
714             (void *)input_chan_bindings->application_data.value;
715     } else {
716         arg.input_chan_bindings.present = NO;
717         arg.input_chan_bindings.initiator_addrtype = 0;
718         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_len = 0;
719         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_val = 0;
720         arg.input_chan_bindings.acceptor_addrtype = 0;
721         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_len = 0;
722         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_val = 0;
723         arg.input_chan_bindings.application_data.GSS_BUFFER_T_len = 0;
724         arg.input_chan_bindings.application_data.GSS_BUFFER_T_val = 0;
725     }

```

```

727     arg.input_token.GSS_BUFFER_T_len =
728     (uint_t)(input_token != GSS_C_NO_BUFFER ?
729     input_token->length : 0);
730     arg.input_token.GSS_BUFFER_T_val =
731     (char *) (input_token != GSS_C_NO_BUFFER ?
732     input_token->value : 0);

734     /* call the remote procedure */

736     bzero((caddr_t)&res, sizeof (res));
737     if (gss_init_sec_context_l(&arg, &res, clnt) != RPC_SUCCESS) {

739     /*
740     * if the RPC call times out, null out all return arguments, set
741     * minor_status to its maximum value, and return GSS_S_FAILURE
742     */

744         if (minor_status != NULL)
745             *minor_status = DEFAULT_MINOR_STAT;
746         if (actual_mech_type != NULL)
747             *actual_mech_type = NULL;
748         if (output_token != NULL)
749             output_token->length = 0;
750         if (ret_flags != NULL)
751             *ret_flags = 0;
752         if (time_rec != NULL)
753             *time_rec = 0;

755         killgssd_handle(clnt);
756         (void) gss_release_buffer(&minor_status_temp, &external_name);
757         GSSLOG0(1, "kgss_init_sec_context: RPC call times out\n");
758         return (GSS_S_FAILURE);
759     }

761     /* free the allocated memory for the flattened name */

763     (void) gss_release_buffer(&minor_status_temp, &external_name);

765     if (minor_status != NULL)
766         *minor_status = res.minor_status;

768     if (output_token != NULL && res.output_token.GSS_BUFFER_T_val != NULL) {
769         output_token->length =
770         (size_t)res.output_token.GSS_BUFFER_T_len;
771         output_token->value =
772         (void *)MALLOC(output_token->length);
773         (void) memcpy(output_token->value,
774         res.output_token.GSS_BUFFER_T_val,
775         output_token->length);
776     }

778     /* if the call was successful, copy out the results */
779     if (res.status == (OM_uint32) GSS_S_COMPLETE ||
780     res.status == (OM_uint32) GSS_S_CONTINUE_NEEDED) {
781     /*
782     * if the return code is GSS_S_CONTINUE_NEEDED
783     * ignore all return parameters except for
784     * status codes, output token and context handle.
785     */
786     *context_handle =
787     *((gssd_ctx_id_t *)
788     res.context_handle.GSS_CTX_ID_T_val);
789     *gssd_context_verifier = res.gssd_context_verifier;

782     if (output_token != NULL) {

```

```

783         output_token->length =
784         (size_t)res.output_token.GSS_BUFFER_T_len;
785         output_token->value =
786         (void *)MALLOC(output_token->length);
787         (void) memcpy(output_token->value,
788         res.output_token.GSS_BUFFER_T_val,
789         output_token->length);
790     }

791     if (res.status == GSS_S_COMPLETE) {
792         if (actual_mech_type != NULL) {
793             *actual_mech_type =
794             (gss_OID) MALLOC(sizeof (gss_OID_desc));
795             (*actual_mech_type)->length =
796             (OM_UINT32)
797             res.actual_mech_type.GSS_OID_len;
798             (*actual_mech_type)->elements =
799             (void *)
800             MALLOC((*actual_mech_type)->length);
801             (void) memcpy((*actual_mech_type)->elements,
802             (void *)
803             res.actual_mech_type.GSS_OID_val,
804             (*actual_mech_type)->length);
805         }

808         if (ret_flags != NULL)
809             *ret_flags = res.ret_flags;

811         if (time_rec != NULL)
812             *time_rec = res.time_rec;
813     }
814 }

816 /*
817 * free the memory allocated for the results and return with the status
818 * received in the rpc call
819 */

821     clnt_freeres(clnt, xdr_gss_init_sec_context_res, (caddr_t)&res);
822     killgssd_handle(clnt);
823     return (res.status);

825 }
      unchanged_portion_omitted

942 static OM_uint32
943 kgss_accept_sec_context_wrapped(
944     OM_uint32 *minor_status,
945     gssd_ctx_id_t *context_handle,
946     OM_uint32 *gssd_context_verifier,
947     const gssd_cred_id_t verifier_cred_handle,
948     OM_uint32 gssd_cred_verifier,
949     const gss_buffer_t input_token,
950     const gss_channel_bindings_t input_chan_bindings,
951     gss_buffer_t src_name,
952     gss_OID *mech_type,
953     gss_buffer_t output_token,
954     int *ret_flags,
955     OM_uint32 *time_rec,
956     gss_cred_id_t *delegated_cred_handle,
957     uid_t uid)
958 {
959     CLIENT *clnt;

961     gss_accept_sec_context_arg arg;

```

```

962     gss_accept_sec_context_res res;
963     struct kgss_cred *kcred;

965     /* get the client handle to GSSD */

967     if ((clnt = getgssd_handle()) == NULL) {
968         GSSLOG(1,
969             "kgss_accept_sec_context: can't connect to server on %s\n",
970             server);
971         return (GSS_S_FAILURE);
972     }

974     /* copy the procedure arguments into the rpc arg parameter */

976     arg.uid = (OM_uint32)uid;

978     arg.context_handle.GSS_CTX_ID_T_len =
979         *context_handle == (gssd_ctx_id_t)GSS_C_NO_CONTEXT ?
980         0 : (uint_t)sizeof (gssd_ctx_id_t);
981     arg.context_handle.GSS_CTX_ID_T_val = (char *)context_handle;
982     arg.gssd_context_verifier = *gssd_context_verifier;

984     arg.verifier_cred_handle.GSS_CRED_ID_T_len =
985         verifier_cred_handle ==
986         (gssd_cred_id_t)GSS_C_NO_CREDENTIAL ?
987         0 : (uint_t)sizeof (gssd_cred_id_t);
988     arg.verifier_cred_handle.GSS_CRED_ID_T_val =
989         (char *)&verifier_cred_handle;
990     arg.gssd_cred_verifier = gssd_cred_verifier;

992     arg.input_token_buffer.GSS_BUFFER_T_len =
993         (uint_t)(input_token != GSS_C_NO_BUFFER ?
994             input_token->length : 0);
995     arg.input_token_buffer.GSS_BUFFER_T_val =
996         (char *)(input_token != GSS_C_NO_BUFFER ?
997             input_token->value : 0);

999     if (input_chan_bindings != GSS_C_NO_CHANNEL_BINDINGS) {
1000         arg.input_chan_bindings.present = YES;
1001         arg.input_chan_bindings.initiator_addrtype =
1002             input_chan_bindings->initiator_addrtype;
1003         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_len =
1004             (uint_t)input_chan_bindings->initiator_address.length;
1005         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_val =
1006             (void *)input_chan_bindings->initiator_address.value;
1007         arg.input_chan_bindings.acceptor_addrtype =
1008             input_chan_bindings->acceptor_addrtype;
1009         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_len =
1010             (uint_t)input_chan_bindings->acceptor_address.length;
1011         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_val =
1012             (void *)input_chan_bindings->acceptor_address.value;
1013         arg.input_chan_bindings.application_data.GSS_BUFFER_T_len =
1014             (uint_t)input_chan_bindings->application_data.length;
1015         arg.input_chan_bindings.application_data.GSS_BUFFER_T_val =
1016             (void *)input_chan_bindings->application_data.value;
1017     } else {

1019         arg.input_chan_bindings.present = NO;
1020         arg.input_chan_bindings.initiator_addrtype = 0;
1021         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_len = 0;
1022         arg.input_chan_bindings.initiator_address.GSS_BUFFER_T_val = 0;
1023         arg.input_chan_bindings.acceptor_addrtype = 0;
1024         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_len = 0;
1025         arg.input_chan_bindings.acceptor_address.GSS_BUFFER_T_val = 0;
1026         arg.input_chan_bindings.application_data.GSS_BUFFER_T_len = 0;
1027         arg.input_chan_bindings.application_data.GSS_BUFFER_T_val = 0;

```

```

1028     }

1030     /* set the return parameters in case of errors.... */
1031     if (minor_status != NULL)
1032         *minor_status = DEFAULT_MINOR_STAT;
1033     if (src_name != NULL) {
1034         src_name->length = 0;
1035         src_name->value = NULL;
1036     }
1037     if (mech_type != NULL)
1038         *mech_type = NULL;
1039     if (output_token != NULL)
1040         output_token->length = 0;
1041     if (ret_flags != NULL)
1042         *ret_flags = 0;
1043     if (time_rec != NULL)
1044         *time_rec = 0;
1045     if (delegated_cred_handle != NULL)
1046         *delegated_cred_handle = NULL;

1048     /* call the remote procedure */

1050     bzero((caddr_t)&res, sizeof (res));
1051     if (gss_accept_sec_context_l(&arg, &res, clnt) != RPC_SUCCESS) {
1052         killgssd_handle(clnt);
1053         GSSLOG(1, "kgss_accept_sec_context: RPC call times out\n");
1054         return (GSS_S_FAILURE);
1055     }

1057     if (minor_status != NULL)
1058         *minor_status = res.minor_status;

1060     if (output_token != NULL && res.output_token.GSS_BUFFER_T_val != NULL) {
1061         output_token->length =
1062             res.output_token.GSS_BUFFER_T_len;
1063         output_token->value =
1064             (void *) MALLOC(output_token->length);
1065         (void) memcpy(output_token->value,
1066             res.output_token.GSS_BUFFER_T_val,
1067             output_token->length);
1068     }

1071     /* if the call was successful, copy out the results */

1073     if (res.status == (OM_uint32) GSS_S_COMPLETE ||
1074         res.status == (OM_uint32) GSS_S_CONTINUE_NEEDED) {

1076         /*
1077          * the only parameters that are ready when we
1078          * get GSS_S_CONTINUE_NEEDED are: minor, ctxt_handle,
1079          * and the output token to send to the peer.
1080          */

1082         *context_handle = *((gssd_ctx_id_t *)
1083             res.context_handle.GSS_CTX_ID_T_val);
1084         *gssd_context_verifier = res.gssd_context_verifier;

1073     if (output_token != NULL) {
1074         output_token->length =
1075             res.output_token.GSS_BUFFER_T_len;
1076         output_token->value =
1077             (void *) MALLOC(output_token->length);
1078         (void) memcpy(output_token->value,
1079             res.output_token.GSS_BUFFER_T_val,
1080             output_token->length);

```

```

1081     }
1083     if (minor_status != NULL)
1084         *minor_status = res.minor_status;
1086     /* these other parameters are only ready upon GSS_S_COMPLETE */
1087     if (res.status == (OM_uint32) GSS_S_COMPLETE) {
1089         if (src_name != NULL) {
1090             src_name->length = res.src_name.GSS_BUFFER_T_len;
1091             src_name->value = res.src_name.GSS_BUFFER_T_val;
1092             res.src_name.GSS_BUFFER_T_val = NULL;
1093             res.src_name.GSS_BUFFER_T_len = 0;
1094         }
1096         /*
1097          * move mech type returned to mech_type
1098          * for gss_import_name_for_mech()
1099          */
1100         if (mech_type != NULL) {
1101             *mech_type = (gss_OID)
1102                 MALLOC(sizeof (gss_OID_desc));
1103             (*mech_type)->length =
1104                 (OM_UINT32) res.mech_type.GSS_OID_len;
1105             (*mech_type)->elements =
1106                 (void *) MALLOC((*mech_type)->length);
1107             (void) memcpy((*mech_type)->elements,
1108                 res.mech_type.GSS_OID_val,
1109                 (*mech_type)->length);
1110         }
1112         if (ret_flags != NULL)
1113             *ret_flags = res.ret_flags;
1115         if (time_rec != NULL)
1116             *time_rec = res.time_rec;
1118         if ((delegated_cred_handle != NULL) &&
1119             (res.delegated_cred_handle.GSS_CRED_ID_T_len
1120              != 0)) {
1121             kcred = KGSS_CRED_ALLOC();
1122             kcred->gssd_cred =
1123                 *((gssd_cred_id_t *)
1124                 res.delegated_cred_handle.GSS_CRED_ID_T_val);
1125             kcred->gssd_cred_verifier =
1126                 res.gssd_context_verifier;
1127             *delegated_cred_handle = (gss_cred_id_t)kcred;
1128         }
1130     }
1131 }
1134 /*
1135  * free the memory allocated for the results and return with the status
1136  * received in the rpc call
1137  */
1139 clnt_freeres(clnt, xdr_gss_accept_sec_context_res, (caddr_t)&res);
1140 killgssd_handle(clnt);
1141 return (res.status);
1143 }

```

unchanged portion omitted

new/usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/init\_ctx.c

1

```
*****
24005 Thu May 7 01:13:52 2009
new/usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/init_ctx.c
6817447 libgss and various mechs are hiding both the real minor_status and the e
6405422 Solaris acceptors fail in AD-KDC environments when using non-"host" serv
6824434 Unable to accept context establishment initiated by Windows 2000 clients
6787343 kclient's site lookups fail in certain network environments
6692646 kclient should output errors to stderr
6525327 kinit failed when arcfour-hmac-md5-exp was used for the principal's key
6745582 SUNWkdcu missing package dependencies after kclientv2 integration
*****
1 /*
2 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
2 * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
3 * Use is subject to license terms.
4 */

6 /*
7 * lib/krb5/krb/init_ctx.c
8 *
9 * Copyright 1994,1999,2000, 2002, 2003 by the Massachusetts Institute of Techn
10 * All Rights Reserved.
11 *
12 * Export of this software from the United States of America may
13 * require a specific license from the United States Government.
14 * It is the responsibility of any person or organization contemplating
15 * export to obtain such a license before exporting.
16 *
17 * WITHIN THAT CONSTRAINT, permission to use, copy, modify, and
18 * distribute this software and its documentation for any purpose and
19 * without fee is hereby granted, provided that the above copyright
20 * notice appear in all copies and that both that copyright notice and
21 * this permission notice appear in supporting documentation, and that
22 * the name of M.I.T. not be used in advertising or publicity pertaining
23 * to distribution of the software without specific, written prior
24 * permission. Furthermore if you modify this software you must label
25 * your software as modified software and not distribute it in such a
26 * fashion that it might be confused with the original M.I.T. software.
27 * M.I.T. makes no representations about the suitability of
28 * this software for any purpose. It is provided "as is" without express
29 * or implied warranty.
30 *
31 * krb5_init_context()
32 */

34 /*
35 * Copyright (C) 1998 by the FundsXpress, INC.
36 *
37 * All rights reserved.
38 *
39 * Export of this software from the United States of America may require
40 * a specific license from the United States Government. It is the
41 * responsibility of any person or organization contemplating export to
42 * obtain such a license before exporting.
43 *
44 * WITHIN THAT CONSTRAINT, permission to use, copy, modify, and
45 * distribute this software and its documentation for any purpose and
46 * without fee is hereby granted, provided that the above copyright
47 * notice appear in all copies and that both that copyright notice and
48 * this permission notice appear in supporting documentation, and that
49 * the name of FundsXpress. not be used in advertising or publicity pertaining
50 * to distribution of the software without specific, written prior
51 * permission. FundsXpress makes no representations about the suitability of
52 * this software for any purpose. It is provided "as is" without express
53 * or implied warranty.
```

new/usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/init\_ctx.c

2

```
54 *
55 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
56 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
57 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
58 */

60 #include "k5-int.h"

62 /*
63 * Solaris Kerberos: the code related to EF/pkcs11 and fork safety are mods Sun
64 * has made to the MIT code.
65 */

67 #ifndef _KERNEL
68 #include <ctype.h>

70 pid_t __krb5_current_pid; /* fork safety: contains the current process ID */
71 #endif

73 #ifndef _KERNEL
74 #include <krb5_libinit.h>
75 #endif

77 /* The des-mdX entries are last for now, because it's easy to
78 configure KDCs to issue TGTs with des-mdX keys and then not accept
79 them. This'll be fixed, but for better compatibility, let's prefer
80 des-crc for now. */
81 /*
82 * Solaris Kerberos:
83 * Added arcfour-hmac-md5-exp as default enc type.
84 * Changed des3-hmac-shal to des3-cbc-shal-kd, as specified in RFC3961.
85 */
86 #define DEFAULT_ETYPE_LIST \
87     "aes256-cts-hmac-shal-96 " \
88     "aes128-cts-hmac-shal-96 " \
89     "des3-cbc-shal-kd " \
90     "des3-hmac-sha1 " \
91     "arcfour-hmac-md5 " \
92     "arcfour-hmac-md5-exp " \
93     "des-cbc-md5 " \
94     "des-cbc-crc"

96 /* The only functions that are needed from this file when in kernel are
97 * krb5_init_context and krb5_free_context.
98 * In krb5_init_context we need only os_init_context since we don't need the
99 * profile info unless we do init/accept in kernel. Currently only mport,
100 * delete , sign/verify, wrap/unwrap routines are ported to the kernel.
101 */

103 #if (defined(_WIN32))
104 extern krb5_error_code krb5_vercheck();
105 extern void krb5_win_cdll_load(krb5_context context);
106 #endif

108 static krb5_error_code init_common (krb5_context *, krb5_boolean, krb5_boolean);

110 krb5_error_code KRB5_CALLCONV
111 krb5_init_context(krb5_context *context)
112 {

114     return init_common (context, FALSE, FALSE);
115 }

    unchanged_portion_omitted
```