



CRESCO

**Centro computazionale di RicErca sui Sistemi
COMplessi**

**Dipartimento di Ingegneria dell'Informazione e Matematica
Applicata**

Università di Salerno

Responsabile Unità Operativa

Prof. Ciro D'Apice

LA1-SP I.2 - Sviluppo ed integrazione dell'architettura GRID e di grafica 3 D

Attività I.2.1 Integrazione ENEA GRID con altre architetture di GRID computing

**1c – Prototipazione di un componente software
realizzato in base alle specifiche emerse al punto 1b**

Realizzazione delle funzioni prototipali in ambiente Windows gettok e
puttok del kit LSF per AFS

V. 1.0

Indice

1. Varie	3
1.1. Bibliografia e Riferimenti	3
1.2. Acronimi.....	3
1.3. Glossario	4
2. Contesto, obiettivi e scopo	7
3. Realizzazioni di funzioni prototipali	9
3.1. Lettura del token tramite gettok_win	9
3.2. Scrittura del token tramite puttok_win	12
3.3. Comparazione dei token letti.....	15
4. I risultati.....	16
4.1. Principali differenze riscontrate fra i due ambienti	16
4.1.1. Le strutture dati	16
4.1.2. Il Process Authentication Group (PAG).....	17
4.2. Passaggio del token	19
5. Conclusioni	21

1. Varie

1.1. Bibliografia e Riferimenti

- [1] Analisi dell'infrastruttura GRID ENEA ed individuazione dei punti critici ai fini dell'interoperabilità e dell'estensione delle caratteristiche di gestione e controllo dell'infrastruttura stessa
- [2] Studio di fattibilità per il porting in ambiente Windows delle routine gettok e puttok del kit LSF per AFS
- [3] AFS User Guide di IBM
- [4] Open AFS Guide
- [5] Programming with the Andrew Cache Manager
- [6] Administering Platform LSF
- [7] <http://www.cresco.enea.it>
- [8] <http://www.platform.com>
- [9] <http://www.openafs.org>

1.2. Acronimi

ACL	A ccess C ontrol L ist
AFS	A ndrew F ile S ystem
API	A pplication P rogram I nterface
CRESCO	C entro C omputazionale di Ric erca sui Sistemi C omplexi
IOCTL	I nput O utput C on T ro L
KDC	K ey D istribution C enter
KTC	K ernel T oken C ache
KVNO	K ey V ersion N umber
LSF	L oad S haring F acility
PAG	P rocess A uthentication G roup
RPC	R emote P rocedure C all
TGS	T icket G ranting S ervice
TGT	T icket G ranting T icket
UID	U ser I dentifier

1.3. Glossario

Authentication Service (AS)

E' la parte di KDC che distribuisce i ticket sia i TGT sia altri tipi di ticket per l'accesso ai servizi.

Cache Manager

In AFS è il processo che gira sui sistemi client, trasferisce i dati da e verso il server, gestisce l'accesso, la visualizzazione, e la modifica dei file dalla directory principale /afs che rappresenta la radice di una struttura organizzata ad albero.

Clear Token

Parte del token di AFS che contiene al suo interno, fra l'altro, la Session Key (anche detta Handshake Key). Dai codici sorgenti esaminati la lunghezza del Clear Token dovrebbe essere di 24 byte.

Complex Mutual Authentication

Particolare tipo di Mutual Authentication che coinvolge tre chiavi di cifratura e tre parti. E' il tipo di autenticazione adottato da AFS

Handshake Key

Vedere Session Key.

Kerberos Authentication Server (kaserver)

E' uno dei processi server di AFS ed assicura che le comunicazioni sulla rete siano sicure. Verifica l'identità dell'utente al login e fornisce le strutture con le quali i partecipanti nelle transazioni verificano la loro identità uno rispetto all'altro. Gestisce l'Authentication DB.

KDC (Key Distribution Center)

E' il servizio che implementa l'autenticazione Kerberos mediante l'Authentication Service (AS) ed il Ticket Granting Service (TGS). Il KDC ha una copia di ogni encryption key associata a ciascun principal. La maggior parte delle implementazioni di KDC archivia i principals in un database e quindi a volte si impiega il termine Kerberos database applicato al KDC.

Mutual Authentication

Meccanismo con il quale le parti provano la loro effettiva identità e basato sul fatto che le parti provano la loro identità mostrando la conoscenza di uno shared secret.

Principal

E' il nome con cui si fa riferimento alle *entry* nel database dell'Authentication Server. Ad ogni utente, host o servizio di un dato realm viene associato un principal. Il generico formato

di un principal utente è : Nome[/Istanza]@REALM mentre per un servizio è Servizio/Hostname@REALM

Realm

(letteralmente regno) indica un dominio amministrativo di autenticazione che fissa i confini entro cui un server di autenticazione è autoritario nell'autenticare un utente, un host o un servizio.

Secret Key

Una chiave di cifratura permanente condivisa da un principal e dal KDC, impiegata per cifrare/decifrare la Session Key inclusa nel TGT dell'autenticazione iniziale. Nel caso il principal sia un utente la Secret Key è derivata dalla password Kerberos dell'utente stesso.

Secret Token

Parte del token riferita anche con il termine ticket dall'analisi del codice e dalle verifiche effettuate può avere diverse lunghezze (40, 56, 150, 166 byte).

Session Key

Una chiave di cifratura temporanea impiegata fra due principal, con un tempo di vita limitato alla durata di un relativo ticket. Nei codici sorgenti esaminati viene anche denominata Handshake Key.

Shared Secret

Informazione conosciuta solo dalle parti che si stanno mutuamente autenticando; esse potrebbero venire a conoscenza tramite una terza parte considerata fidata dalle prime due. Dovrebbe essere costituita dalla Session Key.

TGS (Ticket-Granting Service)

La parte del KDC che distribuisce i ticket ai client per gli specifici servizi. Il processo utente comunica con il TGS grazie ad un TGT.

TGT (Ticket-Granting Ticket)

Uno speciale ticket Kerberos che consente al client di ottenere, in maniera trasparente altri ticket.

Ticket

Un insieme di credenziali elettroniche che verifica l'identità di un client per la fruizione di uno specifico servizio.

Token

Struttura impiegata da AFS per l'autorizzazione e l'accesso al file system. Al suo interno contiene un ticket. Nei codici esaminati è suddivisa in due sezioni fondamentali il Secret Token (anche riferito con il termine Ticket) ed il Clear Token.

Vice

In ambiente AFS indica l'insieme dei server ed il termine deriva dal nome del software che gira su di essi. Il termine è anche usato come suffisso per l'identificazione, nei codici sorgenti, di strutture e dati (ViceIOCTL, ViceID).

2. Contesto, obiettivi e scopo

Il Progetto CRESCO: Centro computazionale di RicErca sui Sistemi Complessi ha come obiettivo la realizzazione, presso il Centro Ricerche ENEA di Portici (NA), di un importante Polo di calcolo multidisciplinare per lo studio dei sistemi complessi di natura biologica e tecnologica, collegato con *facilities* di elaborazione, archiviazione e visualizzazione 3D di dati ubicate presso i Centri ENEA di Brindisi e Trisaia (MT).

Sul piano delle applicazioni e dei contenuti scientifici, il Progetto si focalizza sulle seguenti linee d'attività:

- Linea di Attività 1 (LA1): Realizzazione del Polo di Calcolo e sviluppo di nuove funzionalità di GRID Computing
l'implementazione di soluzioni innovative in tema di architetture di sistemi di calcolo e di GRID computing per le attività di R&S di punta dell'ENEA che richiedano l'utilizzo di risorse computazionali estremamente importanti. Sul versante infrastrutturale il Progetto prevede la realizzazione di una piattaforma di calcolo di assoluta eccellenza in grado di erogare una potenza dell'ordine del Teraflop, corredata di una nuova classe di funzionalità GRID innovative;
- Linea di Attività 2 (LA2): Sviluppo di tecnologie e modelli computazionali per la descrizione di sistemi complessi di origine biologica e di materiali innovativi
lo studio di oggetti biologici dal punto di vista "sistemico" (in silico cell) e lo studio di sistemi naturali (comunità animali e sociali) secondo il paradigma dei sistemi complessi;
- Linea di Attività 3 (LA3): Sviluppo di modelli di simulazione ed analisi delle Reti tecnologiche complesse e delle loro interdipendenze
lo studio di sistemi tecnologici complessi e delle loro mutue interazioni, e la realizzazione di opportuni strumenti per il modelling, la simulazione e il controllo.

Nell'ambito delle attività previste dalla Linea di Attività 1 il sottoprogetto identificato come SP I.2, *Sviluppo ed integrazione dell'architettura GRID e di grafica 3D*, prevede, fra l'altro, l'attività di integrazione dell'ENEA-GRID con altre architetture di *GRID computing* e nell'ambito di tale attività, nella quale il Dipartimento di Ingegneria dell'Informazione e Matematica Applicata dell'Università degli Studi di Salerno è coinvolto, è prevista, primariamente, un'analisi dell'infrastruttura ENEA-GRID [1] con l'individuazione dei punti critici ai fini dell'interoperabilità, in particolar modo con sistemi Windows, e dell'estensione delle caratteristiche di gestione e controllo dell'infrastruttura stessa.

Uno dei problemi emersi dal documento di analisi è quello relativo al non supportato trasferimento del token da parte del gestore delle risorse LSF dal mondo Unix a quello

Windows. In effetti LSF non supporta AFS in ambiente Windows e quindi, nel momento in cui si desidera attivare un job su una macchina Windows, il mancato trasferimento del token AFS (che ovviamente consente all'utente di poter accedere al proprio volume) determina di fatto l'impossibilità di avere il proprio spazio a disposizione sull'host Windows e, nel caso il sistema di autorizzazione risultasse basato sulle ACL di AFS, l'operazione sarebbe di fatto impossibile. È dunque opportuno verificare se sia possibile effettuare l'operazione di trasferimento del token da e verso sistemi Windows, partendo dai codici sorgenti delle routine a ciò destinate e disponibili per il solo mondo Unix.

Lo studio di fattibilità per l'operazione del *porting* in ambiente Windows [2] del trasferimento del token AFS per LSF si è mosso lungo due direzioni:

1. studio ed analisi del kit che serve a LSF per trattare i token AFS denominato *Platform LSF Source Code Package for AFS*, disponibile per il solo mondo Unix, e nel seguito riferito semplicemente come kit LSF;
2. studio ed analisi dei sorgenti di OpenAFS per Windows per stabilire dei punti in comune fra i due ambienti che potrebbero essere utili per l'operazione di *porting*.

Le due routine che sembrano avere un ruolo chiave nel trasferimento del token da un sistema ad un altro sono le routine **gettok** e **puttok**, presenti nel kit LSF per AFS; per cui si è provveduto alla realizzazione di versioni prototipali di routine idonee alla lettura ed alla scrittura di un token AFS su sistemi Windows. Tale attività è descritta nel presente documento.

3. Realizzazioni di funzioni prototipali

Sulla base degli elementi acquisiti ed esposti nei capitoli 4 e 5 di [2] si è proceduto con la realizzazione di due funzioni prototipali per leggere e scrivere il token in ambiente Windows. Tale approccio consente di delineare in maniera corretta e solida, fornendo i relativi elementi di giudizio, le conclusioni che verranno espresse dal presente studio di fattibilità.

Le routine prototipali potranno trattare un solo token mentre, è opportuno ricordare, che le routine puttok e gettok del kit LSF sono capaci di impostare e leggere fino a cinque token e a causa di questa diversità e al fine di effettuare delle prove di passaggio del token da un sistema a Windows ad uno Unix, le routine prototipali create introducono, dove necessario, dei valori fittizi che saranno opportunamente documentati e commentati nel seguito.

3.1. Lettura del token tramite gettok_win

La funzione prototipale per la lettura del token in ambiente Windows è stata costruita impiegando le routine del kit OpenAFS di cui al § 5.2.1 (ktc_ListTokens) e § 5.2.2 (ktc_GetToken) di [2] ed utilizzando il sorgente tokens.c come template al fine di utilizzare i corretti file richiamati dalle direttive di include.

La lettura del codice e dei relativi commenti consente la piena comprensione dei seguenti punti chiave:

1. l'invocazione della ktc_ListTokens consente di sapere se ci sono dei token memorizzati nel sistema e, in caso positivo, è invocata la routine ktc_GetToken che restituisce in token le informazioni lette
2. la fase di stampa sullo standard output del contenuto di token è preceduta da quella dei due interi fittizi ed in ogni caso l'ordine di stampa è quello dell'ambiente LSF che prevede:

progressivo del token
lunghezza del token
lunghezza del secret token
secret token
lunghezza cleartoken
dimensione della Session Key
Session Key
kvno
username
StartTime
EndTime
Nome della Cella

Il codice della routine è:

```

/*****
 * gettok_win.c --
 * Sorgente per la lettura del token AFS in ambiente Windows
 * e la visualizzazione su stdout in formato compatibile con gettok di LSF.
 *****/
#include <afs/param.h>
#include <afs/stds.h>
#include <windows.h>
#include <winsock2.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <afs/auth.h>

struct ClearToken {
    afs_int32 AuthHandle;
    char HandShakeKey[8];
    afs_int32 ViceId;
    afs_int32 BeginTimestamp;
    afs_int32 EndTimestamp;
};

main(argc, argv)
int argc;
char **argv;
{
    int cellNum;
    int rc, i, j;
    char userName[100];
    struct ktc_principal serviceName, clientName;
    struct ktc_token token;
    struct ClearToken ct;
    WSADATA WSAjunk;

    WSStartup(0x0101, &WSAjunk);
    if (argc > 1) {
        printf("!! Don't use arguments for '%s'\n", argv[0]);
        return 0;
    }

    cellNum = 0;
    if ( (rc = ktc_ListTokens(cellNum, &cellNum, &serviceName)) != 0 )
        return rc;

    if ( (rc = ktc_GetToken(&serviceName, &token, sizeof(token), &clientName)) != 0 )
        return 0;
    strcpy(userName, clientName.name);
    /*Stampo numero di token e token size
    (per il momento settati manualmente a 1 e 100 */
    printf("1 100 ");
    /*stampo la grandezza del secret token*/
    printf("%d ", token.ticketLen);
    /*Stampo Secret token*/
    for(j = 0; j < token.ticketLen; j++) {
        printf("%x ", (unsigned char)token.ticket[j]);
    }
    /*Stampo dimensione della struttura del clear token*/
    printf("%d ", sizeof(ct));
    /*Stampo dimensione del clear token*/
    printf("%d ", sizeof(token.sessionKey.data));
    /*Stampo Clear token*/
    for (i = 0; i < sizeof(token.sessionKey.data); i++) {
        printf("%d ", token.sessionKey.data[i]);
    }
    /*Stampo il campo AuthHandle*/
    printf("%d ", token.kvno);
    /*Stampo il campo ViceId*/
    printf("%s ", userName);
    /*Stampo i campi StartTime, EndTime, CellName*/
    printf("%d %d %s", token.startTime, token.endTime, serviceName.cell);

```

```
    return(0);  
}
```

Si riporta, per completezza, anche l'output ottenuto da tale routine sulla macchina Windows impiegata per lo sviluppo, rimandando al § 6.3, il confronto di tale output con quello della routine gettok del kit LSF.

```
C:\> gettok_win
```

```
1 100 40 18 1e cf 6a af 6e e3 26 99 7d c3 d4 4a 87 e1 7f eb c2 53 b3 9c 45 ef 5b 46 32  
ab 94 8c 1b 18 a9 89 61 67 a8 c5 4c 16 2b 24 8 31 32 -3 -9 88 -118 91 31 0 romano 0  
1194001307 enea.it
```

Suddividiamo l'output in blocchi per comprendere meglio come è strutturato.

I primi due valori (1) e (100) sono stampati per compatibilità con l'output della routine del kit LSF

Il terzo valore (40) rappresenta la lunghezza del *secret token*

I 40 numeri in rosso rappresentano il *secret token*

Il valore successivo (24) è la dimensione della struttura ClearToken

Il valore seguente (8) è la dimensione della Session Key

Gli 8 numeri in verde rappresentano Session Key

Infine a seguire si hanno:

kvno (0)

Username (romano)

StartTime (0)

EndTime (1194001307)

Nome della Cella (enea.it).

3.2. Scrittura del token tramite puttok_win

La funzione prototipale per la scrittura del token in ambiente Windows è stata costruita impiegando le routine del kit OpenAFS di cui al § 5.3.1 (*ktc_SetToken*) di [2], la routine impiegata dal comando *klog* per scrivere il token.

Il sorgente *aklog.c.c* è stato preso quale template al fine di utilizzare i corretti file richiamati dalle direttive di *include*.

La routine provvede alla lettura delle informazioni dallo *standard input*, in accordo con gli output forniti dalle routine *gettok* e riempie le strutture che devono essere passate alla routine *ktc_SetToken*.

La lettura del codice e dei relativi commenti consente la piena comprensione dei seguenti punti chiave:

1. Poiche la routine *ktc_SetToken* prevede in input oltre alla struttura *token*, anche quelle relative ai *principal server* e *client* è necessario impostare correttamente tali valori prima che la stessa venga richiamata
2. Nei due *principal* viene settato il valore *cell* al *cell name* presente nell'output delle *gettok*, (*enea.it*), *server->name* è settato alla stringa costante *AFS* mentre *client->name* è impostato allo username (*romano*).

La lettura del codice e dei relativi commenti consente la piena comprensione dei seguenti punti chiave:

Il codice della routine è:

```

#include <afs/param.h>
#include <afs/stds.h>
#include <windows.h>
#include <winsock2.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <afs/auth.h>
#include <afs/kautils.h>
#include <afs/dirpath.h>
#include "cm_config.h"
#include "cmd.h"
struct ClearToken {
    afs_int32 AuthHandle;
    char HandShakeKey[8];
    afs_int32 ViceId;
    afs_int32 BeginTimestamp;
    afs_int32 EndTimestamp;
};
#define AFS_TOKEN_SECRET_LEN(t) (*(int *) (t))
#define AFS_TOKEN_SECRET(t) \
    ((char *) (&(AFS_TOKEN_SECRET_LEN(t)) + 1))
main(argc, argv)
int argc;
char **argv;
{
    int cellNum;
    int AuthHandle, ViceId, BeginTime, EndTime;
    int rc, a, b, i, j, tokenLen, tokenNum, ctsize, ctlen;
    char userName[100];
    char smbname[MAXKTCNAMELEN];
    char *cellName, *sectoken;
    struct ktc_principal server, client;
    struct ktc_token token;
    struct ClearToken ct;
    WSADATA WSAjunk;
    WSStartup(0x0101, &WSAjunk);
    if (argc > 1) {
        printf("!! Don't use arguments for '%s'\n", argv[0]);
        return 0;
    }
    /*leggo il numero di token, il token size e il secret token lenght*/
    fscanf(stdin, "%d %d %d", &tokenNum, &tokenLen, &token.ticketLen);
    /*leggo il secret token*/
    for (i = 0; i < token.ticketLen; i++) {
        fscanf(stdin, "%x", &a);
        token.ticket[i] = a;
    }
    /*leggo il size della struttura clear token*/
    fscanf(stdin, "%d", &ctsize);
    /*leggo clear token lenght*/
    fscanf(stdin, "%d", &ctlen);
    /*leggo il clear token*/
    for (j = 0; j < ctlen; j++) {
        fscanf(stdin, "%d", &b);
        token.sessionKey.data[j] = b;
    }
    /*leggo AuthHanlde, ViceId, BeginTime, EndTime, cellName*/
    fscanf(stdin, "%d %s %d %d %s", &token.kvno, &client.name, &token.startTime,
&token.endTime, server.cell);
    strcpy(server.name, "afs");
    strcpy(server.instance, "");
    strcpy(client.instance, "");
    strcpy(client.cell, server.cell);
    if ((rc = ktc_SetToken(&server, &token, &client, (KA_USERAUTH_VERSION &
KA_USERAUTH_AUTHENT_LOGON) ? AFS_SETTOK_LOGON :
0)) != 0)
        return 0;
    if (rc != 0) printf("\nerrore in ktc_setToken");
}

```

```
}    return(0);
```

3.3. Comparazione dei token letti

Esaminiamo in dettaglio gli output forniti dalle due routine di lettura del token ossia la `gettok` del kit LSF e la `gettok_win` sviluppata come prototipo. Ovviamente la prima verrà eseguita su una dei sistemi della ENEA GRID, mentre la seconda viene eseguita sul sistema Windows utilizzato per lo sviluppo.

Output della routine `gettok` in ambiente UNIX:

```
/afs/enea.it/por/user/romano> gettok
```

```
1 100 56 18 1e cf 6a af 6e e3 26 cc b3 cb 48 d3 f e5 bb 2b 6 8c 9 18 21 de 4c 2c 2e a3
47 72 bc 22 e9 49 52 2f 62 69 6e 2f 6c 6d 68 6f 73 74 69 64 20 3b 20 7d a 0 0 1 0 24 8
174 227 42 220 247 193 50 239 0 26148 1191408562 1194000561 enea.it
```

Suddividiamo l'output in blocchi per comprendere meglio come è strutturato.

Il primo valore (1) rappresenta il numero di token presenti nel sistema

Il secondo (100) rappresenta la lunghezza del token

Il terzo valore (56) rappresenta la lunghezza del *secret token*

I 56 numeri in rosso rappresentano il *secret token*

Il valore successivo (24) è la dimensione della struttura `ClearToken`

Il valore seguente (8) è la dimensione dell'`HandShakeKey`

Gli 8 numeri in verde rappresentano `HandShakeKey`

Infine a seguire si hanno:

`AuthHandle` (0)

`VicId` (26148)

`BeginTimestamp` (111408562)

`EndTimestamp` (1194000561)

Nome della Cella (enea.it).

Output della routine `gettok_win` in ambiente Windows:

```
C:\> gettok_win
```

```
1 100 40 18 1e cf 6a af 6e e3 26 99 7d c3 d4 4a 87 e1 7f eb c2 53 b3 9c 45 ef 5b 46 32
ab 94 8c 1b 18 a9 89 61 67 a8 c5 4c 16 2b 24 8 31 32 -3 -9 88 -118 91 31 0 romano 0
1194001307 enea.it
```

La struttura è essenzialmente la stessa, segue lo stesso ordine dell'output della routine per unix salvo le 4 differenze che vengono di seguito evidenziate:

1. I primi due valori ossia (1) e (100) sono valori fittizi ossia non sono letti dalla routine ma solo stampati per motivi di compatibilità anche perché mentre le routine `puttok` e `gettok` del kit LSF sono capaci di settare e leggere fino a cinque token, le routine prototipali si limiteranno a trattare un solo token e quindi, essenzialmente, la routine `gettok_win` i primi due campi, numero di token e lunghezza del token, vengono settati manualmente
2. La dimensione del *secret token* (40 invece che 56)

3. Il valore `Viceld` che in UNIX identificava l'ID dell'utente, in Windows presenta il nome dell'utente ([romano](#))
4. Il `BeginTimestamp` è settato a 0

4. I risultati

Le attività di studio dei sorgenti disponibili, quelle di sviluppo delle funzioni prototipali e le comparazioni degli output nei due ambienti hanno portato alla manifestazione di una serie di differenze e/o peculiarità dei due ambienti e quelle maggiormente rilevanti sono riportate nel § 4.1.

Nonostante difformità, fatte salvo le dovute limitazioni, è stato possibile impiegare un token prelevato da un sistema Windows su un sistema Unix (e viceversa) come descritto in § 4.2.

Il prosieguo delle attività non può prescindere dalla realizzazione di una serie di test da effettuare su sistemi Windows appartenenti alla GRID ENEA come meglio illustrato nel § 5.

4.1. Principali differenze riscontrate fra i due ambienti

Le principali differenze riscontrate fra i due ambienti (UNIX e Windows) sono relative alle strutture dati impiegate nelle routine di cui al § 4.1.1 e al *Process Authentication Group* (PAG) illustrato nel § 4.1.2.

4.1.1. Le strutture dati

In OpenAFS le strutture dati impiegate sono sostanzialmente le stesse sia per i sorgenti relativi al mondo UNIX sia per quelli relativi al mondo Windows.

Le strutture dati impiegate nel kit LSF sono invece differenti da quelle impiegate in OpenAFS.

Nelle routine prototipali sviluppate abbiamo impiegato le strutture dati di OpenAFS.

La *system call* `pioctl`, impiegata in ambiente OpenAFS per Windows, non effettua direttamente attività relative al secret token (o ticket), sia in fase di lettura che di scrittura. Le routine del kit OpenAFS delegano tale operazione a due specifiche routine che funzionano via RPC `receive_key`, per la lettura del ticket e `send_key` per la scrittura del ticket.

Nel token in ambiente UNIX il campo `ViceID` contiene il valore dell'AFS UID¹ che viene assegnato ad ogni utente UNIX che accede al sistema, mentre, sui sistemi Windows non si ha questa assegnazione ed il campo `Viceld` del token contiene il nome dell'utente. Più specificatamente nel caso specifico in ambiente UNIX il `ViceID` contiene l'intero 26148 che corrisponde all'UID dell'utente romano sul sistema mentre in Windows il `ViceID` è costituito dal valore alfanumerico romano.

La lunghezza del *secret token* sembra variare a seconda delle macchine su cui viene memorizzato. Sul sistema Windows è 40 byte, mentre per le macchine Unix varia tra 56, 150 e 166 byte. Al momento non abbiamo elementi che consentano di chiarire in maniera precisa il perché di tale comportamento.

4.1.2. Il Process Authentication Group (PAG)

Uno dei parametri del comando `klog`, ossia il comando per autenticarsi su AFS, è `setpag`. Ovviamente il fine principale del comando `klog` è quello di ottenere un token che consente di accedere al *file system*. L'invocazione del comando con tale parametro determina l'associazione del token con uno speciale numero identificativo chiamato PAG [2] che sta per *Process Authentication Group*. Nel caso non venga usato tale parametro il token risulta associato piuttosto che al PAG all'UID di Unix. Ciò significa che chiunque possa assumere l'identità Unix di un altro utente, anche senza conoscerne la password, ad esempio l'utente root tramite il comando `SU`, possa ereditare il token associato con la UID UNIX. In altre parole impiegando il parametro `setpag` la struttura che memorizza le credenziali di accesso (ossia il token) è identificata dal PAG piuttosto che dall'UID Unix.

Accedendo ai sistemi dell'ENEA GRID si è riscontrato che il token è automaticamente associato ad un PAG piuttosto che all'UID Unix, mentre sul sistema Windows il token sembra associato al nomeutente. Tali risultati sono stati riscontrati dagli output dei comandi tokens nei due ambienti.

UNIX

```
/afs/enea.it/por/user/romano> tokens
```

```
Tokens held by the Cache Manager (PAG Based Tokens):
```

```
User's (AFS ID 26148) tokens for afs@enea.it [Expires Nov 10  
12:36]
```

```
--End of list--
```

WINDOWS

¹ AFS user IDs (AFS UIDs) sono numeri interi positivi assegnati all'utente dal *Protection Server* incrementando di uno il valore `max user id` del *Protection Database*. Nel caso specifico dei sistemi dell'ENEA GRID il valore coincide con la UID in ambiente Unix e presente nel file `/etc/passwd`.

```
C:\>tokens
```

```
Tokens held by the Cache Manager:
```

```
User romano's tokens for afs@enea.it [Expires Nov 10 12:35]
```

```
--End of list --
```

L'analisi del codice OpenAFS sembra confermare che il token trattato in ambiente Windows sia obbligatoriamente un token *user based* piuttosto che *PAG based*.

4.2. Passaggio del token

Come abbiamo già avuto modo di descrivere in precedenza la routine per la lettura del token (**gettok**) visualizza lo stesso sullo **stdout** mentre quella per la scrittura (**settok**) lo legge dallo **stdin** con LSF che agisce, tramite i meccanismi di **esub** ed **eexec**, da *pipe* tra i due comandi.

Sono state effettuate varie prove per avere dei riscontri sulla validità dell'impiego del token anche su sistemi diversi da quello in cui è stato prelevato.

Le prove effettuate sono basate sulla redirectione su file dell'output delle routine di lettura (quindi la **gettok** di LSF per Unix ed il prototipo **gettok_win** per Windows), sul successivo trasferimento su altro sistema, sulla scrittura dello stesso tramite le routine di scrittura (quindi la **settok** di LSF per Unix ed il prototipo **settok_win** per Windows) nel nuovo sistema.

A causa della differenze esistenti nei token letti nei due ambienti, in particolare la diversa natura del campo **Viceld**, risulta che per poter inserire un token Windows su un sistema Unix, bisogna sostituire il valore alfanumerico con uno numerico per evitare che la routine **settok**, che si aspetta un valore numerico, vada in errore sulla lettura della stringa alfanumerica.

Nello specifico, prelevando il token dal sistema Windows tramite la **gettok_win**, sostituendo la stringa nomeutente con un qualsiasi valore numerico, e settando il token su Unix tramite la **puttok** sembra che l'accesso al *file system* continui ad essere consentito.

Se in un token viene invece variato qualche byte del *secret token* l'accesso al *file system* viene inibito e si diventa una sorta di utente anonimo.

Dopo aver settato un token Windows su un sistema Unix l'output del comando **tokens** varia passando dall'indicazione dell'utente mediante la stringa AFS UID 26148 ad un'indicazione UNIX UID valorenumerico ove il valore numerico è quello con il quale è stata sostituita la stringa nomeutente.

Altro aspetto importante è che pur avendo il *secret token* di windows una lunghezza di 40 byte, inferiore quindi a quella dei *secret token* di Unix, dopo aver settato il token Windows su Unix, una successiva **gettok** restituisce nuovamente un *secret* di lunghezza 56 byte.

Sono state effettuate anche prove nel verso opposto, ossia il passaggio di un token Unix al prototipo **puttok_win**; in tal caso non c'è necessità di variare il **Viceld** in quanto la routine lavora già con alfanumerici, e l'accesso al *file system* sembra continuare correttamente.

Dopo aver settato un token Unix su un sistema Windows l'output del comando `tokens` varia passando dall'indicazione dell'utente mediante valore alfanumerico nomeutente ad un'indicazione tramite stringa 26148

Una successiva `gettok_win` restituisce la corretta lunghezza del *secret token* che continua ad avere la lunghezza del *secret* prelevato da Unix.

Nella successiva tabella si riportano le prove di passaggio del token e le differenti lunghezze dei *secret* riscontrate dopo l'immissione.

		A			
		Windows (40)	Kleos (56)	Frascati (150)	Pace (166)
DA	Windows (40)	*	56	56	56
	Kleos (56)	56	*	56	56
	Frascati (150)	150	150	*	150
	Pace (166)	166	166	166	*

Si nota quindi come la lunghezza del *secret* sia molto variabile; sul nostro sistema Windows è di lunghezza 40, mentre in Unix varia dai 56 (`kleos.portici.enea.it`) ai 166 (`pace.bologna.enea.it`) passando per i 150 (`sp4-1.frascati.enea.it`).

Nonostante tali differenze, come già detto, sembra che l'accesso al *file system* continui ad essere garantito dopo l'immissione forzata del token proveniente da un altro sistema.

5. Conclusioni

Gli studi compiuti fino a questo momento hanno evidenziato la realizzabilità di routine che prelevino ed impostino un token AFS in ambiente Windows in modo simile a quanto fanno le routine gettok e puttok del kit LSF.

In considerazione che:

- 1) è necessario approfondire l'aspetto dell'autenticazione nel momento in cui, nella GRID, saranno effettivamente introdotte delle macchine Windows. In effetti, allo stato, l'uso di credenziali comuni, sui sistemi della GRID, risulta limitato ai soli sistemi di famiglia UNIX;
- 2) si ritiene altresì essenziale effettuare una serie di test per valutare se le routine realizzate, sia pure in versione prototipale, possano assolvere anche ai compiti che Platform LSF richiede loro;
- 3) i comportamenti dissomiglianti riscontrati in precedenza potrebbero dar luogo a problemi di integrazione

Si reputa essenziale poter svolgere un articolato piano di test che coinvolga sistemi Windows appartenenti alla GRID ENEA e preferibilmente, per diminuire i tempi relativi alla configurazione, analisi e risoluzione delle problematiche sistemistiche, congiuntamente con i tecnici ed i ricercatori dell'Ente.

Lo svolgimento di tale attività contribuirà a dare un quadro esaustivo in merito all'interazione di LSF con AFS in ambiente Windows.

Nel caso i test sul campo vadano a buon fine si potrà pianificare e procedere con la realizzazione di versioni finali di `gettok_win` e `puttok_win` che saranno, nei limiti del possibile, strutturate similmente e con pari funzionalità rispetto a quelle del kit LSF.

In caso contrario, prima di sancire la non portabilità o meglio l'impossibilità di impiego delle routine in ambiente Windows, potrebbe essere utile svolgere una ulteriore fase di analisi, tesa ad approfondire gli aspetti critici e bloccanti che saranno stati riscontrati nella fase di test.