Autenticación Biométrica On Card en el Protocolo de Kerberos

Lizama Luis, Gómez Roberto

ITESM-CEM, Dpto. Ciencias Computacionales
Edo. de México, México, 52926
00437591@academ01.cem.itesm.mx, rogomez@itesm.mx

Abstract: This paper introduces an authentication protocol based on Kerberos. We modified Kerberos to introduce biometrics with smart card hardware. Naomaru Itoi and Peter Honeyman from research center from Information Technologies Integration have integrated smart card hardware and Kerberos V5. In this work we propose to enforce the authentication module integrating a biometric device to the Itoi and Honeyman design. Our main objective is that authentication process does not be based only in the possession smart card.

1 Introducción

Kerberos fue diseñado en el MIT [1], como un medio para que los passwords (contraseñas) de los usuarios no tengan que viajar en la red, donde posibles intrusos pueden capturarlos. En su lugar, Kerberos proporciona tickets (credenciales) a los usuarios, que los obtienen de un Servidor Central de Distribución. Estos tickets le proporcionan a los usuarios acceso a los hosts (las estaciones de trabajo) y servicios de red. Los tickets se envían encriptados por la red, de tal forma que si alguien los captura no pueda usarlos.

No obstante, Kerberos contiene algunos huecos de seguridad y limitantes, debido principalmente a su dependencia sobre los passwords que son seleccionados por los usuarios. Itoi y Honeyman [2] han mejorado la seguridad de Kerberos, reemplazando los passwords con llaves generadas en forma aleatoria por Kerberos y almacenadas en una tarjeta inteligente. Desgraciadamente este enfoque sustenta el proceso de autentificación en la tenencia o posesión de dicha tarjeta, por lo que la pérdida o sustracción de la misma puede comprometer la seguridad del sistema [5], al permitir el acceso a personas no autorizadas sólo porque traen consigo la tarjeta de autentificación, ya que el NIP de la tarjeta, por sí solo, constituye un mecanismo débil de autentificación.

En el esquema que se propone reemplazamos los passwords por un lector biométrico a partir del cual se obtiene la plantilla biométrica del usuario, que es almacenada y procesada en el microchip de la tarjeta. Así, cuando un usuario intenta obtener acceso a tiempo de login es confrontado por la interfaz biométrica del sistema de autentificación. Si la lectura biométrica del usuario Pb' y la plantilla biométrica almacenada en la tarjeta Pb, son similares de acuerdo con el algoritmo de correlación biométrica, se sigue que la llave Ku, también guardada en la tarjeta, se puede usar para decriptar el ticket de autentificación de Kerberos, el cual obtiene el host Cliente tras la primera fase del protocolo de Kerberos. Por lo tanto, el usuario queda

autentificado y será autorizado para recibir algún servicio de red (ftp, telnet, rsh). Con esto queda previsto que ante pérdidas o sustracción de tarjetas, ningún intruso podrá tener acceso a los servicios de red, debido a la dificultad que implica obtener simultáneamente la tarjeta, la falsificación biométrica del usuario, o la llave de seguridad del usuario de Kerberos Ku. Adicionalmente, el usuario debe ingresar el NIP de la tarjeta antes de obtener los servicios deseados.

El presente artículo está organizado de la siguiente manera; la sección dos describe el protocolo de autentificación de Itoi y Honeyman, por medio del cual se logra la integración de hardware de tarjeta inteligente a Kerberos. La sección tres presenta la incorporación de un dispositivo biométrico dentro del escenario previo, su modelo y su validación en Promela (un lenguaje para la descripción de Protocolos). La sección cuatro explica nuestra propuesta de implementación a partir de una construcción en Java Card 2.0. La sección cinco presenta una nueva tecnología basada en un coprocesador desarrollado por IBM. La sección seis es una discusión de los resultados y por último presentamos nuestras conclusiones.

2 Kerberos On Card

Bellovin y Merrit señalan problemas inherentes a Kerberos que no pueden evitarse sin el uso de hardware de propósito especial, no importando el diseño del protocolo. El termino On Card se refiere al echo de ser ejecutado en la tarjeta. Tales requerimientos de hardware se desprenden del análisis de Itoi y Honeyman [2]. En el protocolo de Kerberos, la llave del cliente, que denotaremos $K(C)$ o simplemente Kc, es compartida por el usuario y el *KDC* (El Centro de Distribución de Llaves de Kerberos). *Kc* es derivada del password del usuario (después de aplicar una función *hash* sobre el password). La estación de trabajo lee el password, lo convierte en *Kc*, y lo usa para decriptar el ticket de autentificación *TGT*. El protocolo aparece ilustrado en la Figura 1, donde se observa la fase de autentificación de Kerberos.

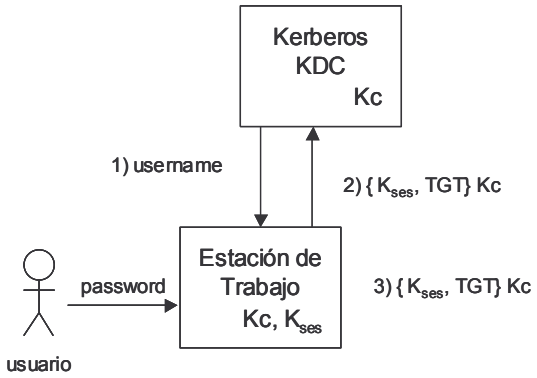

Fig. 1 El Protocolo Original de Autentificación de Kerberos

Si el ticket es decriptado correctamente, el usuario es autentificado y el acceso es permitido. En esta fase del protocolo, la llave de encripción del ticket *Kc*, queda expuesta a dos partes: el usuario y la estación de trabajo. En primer lugar, la llave es memorizada por el usuario y puede dársela a otra persona, o un adversario puede leerla mientras es ingresada en el teclado. En Segundo lugar, la llave es guardada en una estación de trabajo y puede comprometer la seguridad del sistema si la estación no está protegida adecuadamente[2].

Como primer requerimiento, Itoi y Honeyman recomiendan decriptar el ticket en un dispositivo externo de decripción. El segundo es contar con un depósito seguro para guardar la llave. Un atacante poderoso puede accesar (leer y/o escribir) el disco duro de la computadora. Los métodos de respaldo no cuentan con suficiente protección física o criptográfica (la memoria de una estación de trabajo puede ser escaneada por un adversario). El tercer requerimiento tiene que contrarrestar los ataques de diccionario**:** Cuando un usuario selecciona un password débil, la llave *Kc* puede ser objeto de un ataque de diccionario. Kerberos es vulnerable a ataques de diccionario porque es un protocolo de autentificación basado en passwords y es relativamente fácil entregarle a un adversario un par <*texto claro*, *texto cifrado*>. Por lo tanto, es preferible reemplazar los passwords con bits generados aleatoriamente y guardados en una smartcard [2].

3 Kerberos Bio Smart integrado

El diseño de Itoi y Honeyman no previene los ataques debidos a la pérdida o sustracción de la tarjeta, por lo cual se puede llegar a comprometer la seguridad del sistema. Incorporaremos al protocolo anterior una fase de pre-autentificación biométrica de acuerdo a dos posibles escenarios

Em el primero de ellos, la smart card es activada por medio de un biométrico, el PIN (Número de Identificación Personal) se reemplaza por el dato biométrico del usuario (el cual no podrá ser perdido, olvidado o transferido intencionalmente). La plantilla biométrica de un usuario legítimo se almacena en la memoria de una smart card y la llave secreta del usuario se obtiene después de su identificación biométrica. Con la plantilla biométrica el usuario no tiene que inventar un password (y no requiere cambiarlo frecuentemente), además el usuario obtiene acceso único de una sola vez al reemplazar varios passwords por su plantilla biométrica. Para activar la llave de encripción, el usuario no tiene que enviar un PIN a la smartcard, sino la lectura biométrica actual. El procesador del chipcard compara la lectura con la plantilla almacenada en la tarjeta (*Match On Card*), por medio de algún algoritmo de comparación [7].

En el segundo escenario, la smart card es activada por medio del PIN del usuario pero la plantilla biométrica del usuario queda almacenada en la smart card para su verificación *Match On Card* [8]. Con esto queda configurado un esquema de seguridad de cuatro factores: El PIN del usuario, su identificación biométrica, la smart card, y la llave del usuario de Kerberos. Igual que antes, el usuario no necesita inventar un password, basta con conservar su PIN de identificación personal, aunque de los factores mencionados, el PIN es el que con más facilidad puede quedar comprometido, no deja de ser una contención más ante la pérdida o robo de la tarjeta

y posiblemente un requerimiento legal del sistema. Además, el sistema de activación de la tarjeta no necesita ser modificado (la bioAPI y la API biométrica JC aún están en desarrollo). El sistema de activación del PIN puede ser implementado directamente en la tarjeta (para prevenir su exposición al host local), pero los *key pads* con display resultan caros, así que la alternativa es ingresar el PIN en el teclado del host local.

Por otra parte, es posible plantear una variedad de configuraciones de conexión de los puertos entre el bio reader, el smart reader y el host local: En una de ellas las tres unidades pueden estar separadas entre sí, interconectadas y comunicándose a través de una aplicación en el host local, por medio de las interfaces de soporte: BioAPI, PC/SC, API Biométrica JC. Otra alternativa consiste en una estación que incluye tanto el lector biométrico, como el lector de smart card, que se conecta al host (hasta donde sabemos Ankari Bio-Mouse Trinity, es la única excepción en este rubro del mercado). También es posible que el bio reader y/o el smart reader estén interconstruídos al host local.

Las versiones interconstruídas corren software propietario, por ello el primero de los esquemas anteriores es el más general y el que abordaremos aquí. Además los estándares que se están liberando en la actualidad, servirán en adelante como plataforma abierta de desarrollo de aplicaciones de tecnologías independientes. A pesar de esto, nunca es preferible que la estación de trabajo actúe como repetidor para el traspaso de mensajes entre las unidades interconectadas. Aquí la solución proviene de los mecanismos de autentificación interconstruídos que poseen las tarjetas criptográficas. En síntesis, asumiremos la opción de configuración del sistema de seguridad de cuatro factores, basado en la pre-autentificación biométrica con tarjeta activada por PIN de Kerberos.

Respecto a la tecnología biométrica hemos seleccionado la huella digital como la técnica biométrica más adecuada a nuestra implementación, debido a su creciente utilización, economía y sencilla adaptación y a que puede ser utlizada selectivamente, como veremos más adelante. No obstante, dejaremos en claro que la verificación de voz ocupa un lugar preferente, debido a su elevado despliege, economía y capacidad para lograr pruebas de tipo liveness[1] [10].

3.1 Diseño del Protocolo

Los objetivos de diseño del nuevo protocolo quedan de la siguiente manera: 1) Usar bits aleatorios para generar *Kc*. 2) Almacenar la llave del usuario *Kc* en una smart card. 3) Activar la smart card por medio del PIN del usuario. 4) Almacenar la plantilla biométrica maestra del usuario *Pb* en una smartcard y ejecutar el mecanismo de correlación *on card*. 5) Decriptar el *TGT* en una smart card. 6) No modificar el *KDC* de Kerberos

La Figura 2 muestra el esquema general del protocolo de pre-autentificación. Cuando un usuario intenta obtener acceso a una estación de trabajo, inserta su smart card, y también introduce su PIN.

[1] Prueba biométrica de que la persona está realmente viva

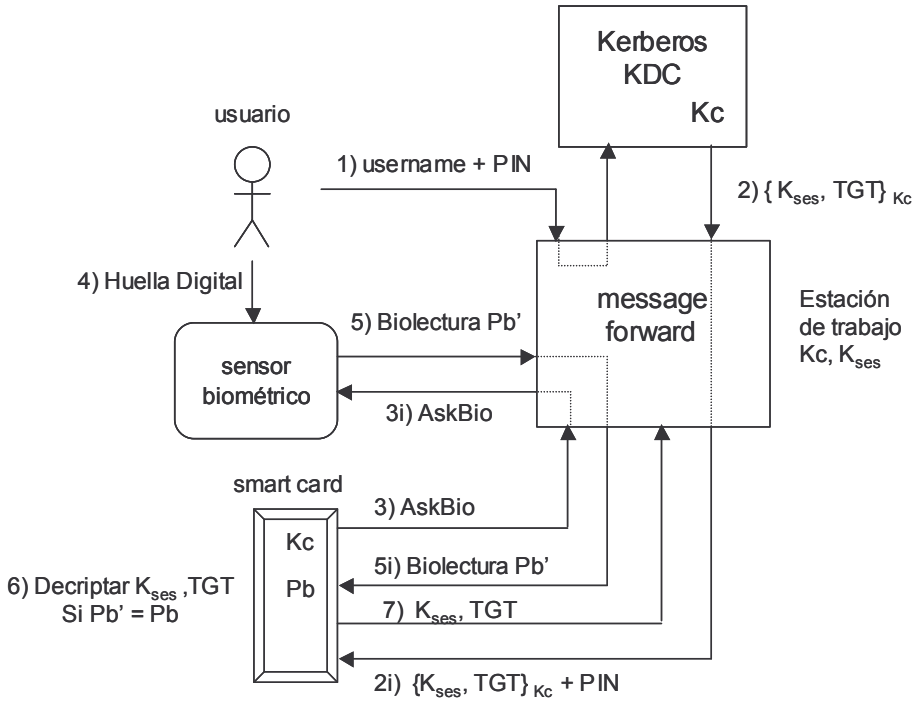

Fig. 2 El Protocolo de Pre-Autentificación Smart-PIN Biométrica

Por su parte, la estación envía una solicitud al *KDC*. 2) El *KDC* genera un *TGT* y una llave de sesión K_{ses}, los encripta con *Kc*, y los envía a la estación de trabajo, quien los reenvía a la smart card (para su decripción y almacenamiento del *TGT*) junto con el PIN del usuario. 3) La smart card verifica el PIN del usuario y solicita al usuario su identificación biométrica *Pb'*.4) El usuario ingresa su dato biométrico por medio del sensor biométrico. La estación de trabajo recibe la lectura biométrica y la reenvía a la smart card. 6) El chipcard compara la lectura *Pb'* con la plantilla biométrica del usuario *Pb* almacenada en la tarjeta. Si la validación es exitosa, el chipcard procede a decriptar el *TGT* con la llave *Kc* también guardada en la tarjeta, si el *TGT* es decriptado correctamente, el usuario es autentificado (mediante K_{ses}) y el acceso es permitido. El protocolo propuesto satisface las metas del diseño: *Kc* es una llave aleatoria, la tarjeta es activada por medio del PIN del usuario, la plantilla biométrica maestra *Pb* y la llave del usuario *Kc* son almacenadas en la tarjeta, el ticket es decriptado en la tarjeta y el *KDC* no sufre modificaciones.

3.2 El Modelo de Pre-Autentificación Biométrica con Tarjeta Activada por PIN

Para desarrollar un prototipo del diseño discutiremos primero el modelo de pre-autentificación de Kerberos derivado de una Máquina de Estados Abstractos ASM de Guverich. El modelo de pre-autentificación es una modificación del modelo de G. Bella[3] del protocolo de Kerberos, que incluye ahora la pre-autentificación biométrica con tarjeta. Para ello hemos agregado los módulos *SMART_MODULE* y

BIO_MODULE y los agentes *Bio* y *Smart* que corren a nombre de ellos, respectivamente. A su vez, los seis módulos tienen su propio subprograma y en cualquier estado la función *Mod* está definida como $Mod(A) = A\_MODULE$, $A \in AGENT$. Debido a esto, en cada regla de *A_MODULE* usamos el nombre de agente *A*, (es decir *Self*). *C_MODULE* está dividido en dos subprogramas *C_MODULE_AUTHENTICATION* y *C_MODULE_AUTHORISATION*, donde el primero contiene el bloque de pre-autentificación biométrica con tarjeta. Examinaremos únicamente el módulo de pre-autentificación.

Para ser bio-autentificado con tarjeta activada por *PIN*, *C* reenvía a *Smart* el PIN del usuario (obtenido por el programa login) y el mensaje encriptado que recibió de *KAS* (Provide_PIN & Require_BSAuthentication rule). Es necesario que la tarjeta del usuario haya sido insertada previamente en el smart reader, en caso contrario *C* solicita la tarjeta al usuario mediante la función *AskSmart*(*C*).

Cuando *Smart* recibe el PIN del usuario y el mensaje cifrado de *C*, reliza la verificación del PIN y si es correcto, almacena el mensaje cifrado en alguna localidad de memoria y responde con la solicitud de datos biométricos del usuario (Get_CryptedMssg & Require_Biometric rule). Enseguida *C* recibe de *Smart* el mensaje de solicitud de datos biométricos, por lo que *C* reenvía el mensaje a *Bio* (Forward_Require_Biometric rule). La estación biométrica recibe la petición de lectura y procede a leer la huella digital del usuario, después los envía de regreso a su interlocutor *C* (Provide_Biometric rule). Si *C* recibe los datos biométricos del usuario, los reenvía a *Smart* para su verificación *Match On Card* (Get_Biometric & Forward_Biometric rule), de otro modo *C* solicita al usuario su identificación biométrica mediante la función *AskBio*(*C*).

A continuación *Smart* recibe la plantilla biométrica del usuario y ejecuta el algoritmo de comparación entre la plantilla biométrica recibida de *C*, y la plantilla maestra del usuario almacenada en alguna localidad de memoria de la tarjeta (Provide_BSAuthentication rule). Si el procedimiento se ejecuta con éxito (el usuario ha sido *bio-autentificado*), *Smart* lee la llave del usuario guardada en el sistema de archivos de la tarjeta y procede a decriptar con esta llave el mensaje cifrado recibido y almacenado de *C*. Si la decripción retorna un mensaje en texto claro *Smart* envía este mensaje a *C* (el usuario ha sido *bio-autentificado con tarjeta*). Cuando *C* recibe el mensaje en texto claro extrae y copia para sí, el ticket y la llave de autentificación que usará ante el *TGS* durante la fase de autorización (**Get_BSAuthentication** rule).

3.3 Validación del modelo

Para la validación del modelo hemos desarrollado un programa en Promela [6], donde las variables globales tienen la forma: bool authenticated = authorised = false, por medio de estas establecemos que antes de iniciar la corrida del protocolo el cliente no está autentificado ni autorizado. Un ejemplo de variable local al proceso KAS y al proceso TGS es int K_TGS = ####, que es la llave secreta de TGS y sólo es conocida por KAS y TGS.

Un canal es un objeto que puede almacenar un conjunto de valores agrupados en estructuras definidas por el usuario [6]. Hemos definido canales de la forma: chan C_SendTo_KAS = [1] of {byte, byte, short} para la transferencia de datos del agente C al agente KAS. En nuestro programa hemos definido la recepción de mensajes cifrados de acuerdo a esta estructura:

```
C_ReceiveFrom_KAS ? K_C    (        K_C_TGS,
                                    Ticket_C_TGS  [encrypt_key],
                                    Ticket_C_TGS  [1], ..Ticket_C_TGS  [6],
                                    TGS,
                                     CT_KAS,
                                     AuthLife              );
```

En este mensaje enviado de KAS a C, K_C es la llave secreta del cliente y es usada para encriptar el mensaje que contiene la llave de autentificación K_C_TGS, el ticket de autentificación Ticket_C_TGS (que viene encriptado a su vez por encrypt_key, la llave secreta de TGS), el identificador de TGS, el timestamp de KAS designado como CT_KAS y la duración máxima del ticket AuthLife.

El valor de todos los campos de los mensajes especificados como constantes, deben coincidir con el valor de los campos correspondientes en el mensaje al inicio del búfer del canal, que hemos incorporado al prototipo bajo guardias como esta: `if :: KAS_ReceiveFrom_C ? [C] -> …..`, para modelar que KAS sólo recibirá el mensaje de C si está definido en la base de datos de KAS. Para el control de flujo la estructura *case* es usada como sigue:

```
       if
       :: Auth_C_TGS -> goto label_1 /*  si  el  Autentificador    C_TGS
existe */
       :: ! Auth_C_TGS->....         /* si el Autentificador  C_TGS no
existe */
```

Nuestra construcción emplea ciertos tipos de mensaje para designar los estados en que puede encontrarse un agente, siendo ReadyToStart el estado donde se inicia la comunicación con el servidor final que ofrece el servicio requerido por el cliente: `mtype = { ReadyToSend, ReadyToReceive, ReadyToStart}`. El siguiente monitor verifica que el Cliente no podrá quedar autorizado para obtener un servicio, si no ha sido previamente autentificado: `proctype monitor () {authorised -> assert(authenticated)}`. Los requerimientos temporales (*never*) en combinación con etiquetas *progress* pueden ser también usados para expresar la ausencia de ciclos sin progresión. Usaremos la exigencia temporal *never* para establecer que un cliente C que ha sido autentificado por KAS y no ha sido autorizado por TGS, eventualmente será autorizado por TGS para obtener el servicio deseado ante ES. La prueba de correctez, consiste en que dada una entrada aceptable por el sistema, éste produce la salida deseada:

```
       never { do:: authenticated && ! authorised -> goto accept od;
                    accept: do ::! authorised od; }
```

Las opciones usadas para la simulación en Xspin 3.4.7, así como la validación del prototipo son: Random simulation style & hash compact search verification mode. Search Mode: Exhaustive, Supertrace/Bitstate, Hash-Compact. Correctness Properties Safety (state properties), Assertions, Liveness, Acceptance Cycles, Apply Never Claim. Los resultados de la simulación y de la verificación del prototipo nos demuestran la correctez del protocolo.

Por el método manual llegamos también a la correctez de la fase de pre-autentificación. En el estado S_{19} del ASM se cumple que [*defined* (*Ticket*(*C, TGS*))] S_{19} & [*defined* (*K*(*C, TGS*))] S_{19}. Para probar que "es siempre cierto que cuando un cliente legítimo que ha logrado enviar a la smartcard *m'=encrypt*({ *K*(*C, TGS*), *Ticket*(*C, TGS*), *TGS, ts, AuthLife*}, *K*(*C*)), ésta eventualmente lo decriptará y devolverá *m'* en texto claro, esto es, *m'' = K*(*C, TGS*), *Ticket*(*C, TGS*), *TGS, ts, AuthLife* ".

4. Implementación

Para la implementación del protocolo usamos una versión simplificada de Kerberos construída usando la extensión criptográfica de Java (JCE), no obstante se instaló Kerberos V5 y el cliente smart card integrado de Keberos, sin embargo tras haber instalado repetidamente el cliente del CITI y haber recompilado Kerberos en arquitecturas distintas, podemos decir que el software se compila tal como lo prescribe el CITI, pero las opciones del programa Kinit para la obtención de tickets en bytecode y su transferencia por el puerto serial de la computadora no quedan disponibles para su uso por el momento. La prueba biométrica y su correlación On Card continúa en desarrollo debido a que la interfaz para la programación de aplicaciones biométricas de la Java Card (JC Biometric API) aún no está disponible. la Java Card en nuestra construcción del prototipo de Kerberos.

4.1 Javacard

Una Java card es una smart card que puede cargar y ejecutar programas escritos en Java, contiene CPU, ROM y memoria EEPROM, y posiblemente un co-procesador criptográfico en el chip. Esta tecnología se diseño de acuerdo a los estándares ISO 7816 [11]. El sistema operativo de la tarjeta consiste de una VM (Máquina Virtual de Java), una pieza de software distribuido que puede ejecutar programas escritos en lenguaje Java. La JCVM corre una versión especial limitada de Java. Las smart cards se comunican usando un mecanismo de paquetes llamado APDUs. Las smart cards son comunicadores reactivos, esto es, nunca inician una comunicación, sólo responden a APDUs desde su terminal. El modelo de comunicación está basado en comandos de respuesta, lo cual significa que la tarjeta recibe un comando APDU, ejecuta la solicitud y retorna una respuesta APDU [12].

Cyberflex Access emplea un subconjunto de Java denominado Java Card versión 2.0, diseñado especialmente para smart cards. Entre otras diferencias, Java Card no cuenta con colector de basura, y no contiene los tipos int, long, float, double. La compilación de un applet se realiza en dos pasos: El primero es `javac -g`, esto compila un archivo `.java` en un archivo `.class`. El segundo es `mksolo`, que convierte un archivo `.class` a un applet descargable al smart reader. Para enviar APDUs al smart reader existen diversas herramientas de comunicación que cumplen el estándar PC/SC: Cyberflex Access Developer's Kit, XCard para Linux, *pay*.

4.2 Kerberos V5

Se instaló el sistema de Kerberos V5 en dos computadoras laptop IBM A20m y Toshiba x86, Red Hat 6.1 x86, las cuales recibieron una IP fija de la red inalámbrica del Tec, por lo cual en una se instaló el KDC y el Servidor de Telnet, con el nombre de red kdc.cem.itesm.mx, la otra IP recibió el nombre server.cem.itesm.mx. Se probaron diversos comandos kerberizados como los siguientes:

```
[root@server bin]telnet –x –f –a -l luislizama kdc.cem.itesm.mx
Trying 148.241.36.238
Connected to kdc.cem.itesm.mx (148.241.36.238).
Waiting for encryption to be negotiated…
[ Kerberos V5 accepts you as “luislizama@LOCALREALM”]
[ Kerberos V5 accepted forwarded credentials]
done.
Last login: Thu Nov  1 03:19:43 from kdc
[luislizama@kdc luislizama]$
```

4.3 El prototipo de Java

Próxima a ser liberada, la Interfaz para la Programación de Aplicaciones Biométricas de la Java Card nos brinda una interfaz para correlación de datos biométricos y registro de plantillas maestras [8]. Hasta quedar disponible la API Biométrica, nos sujetaremos mientras tanto a un prototipo basado en el diseño que hemos descrito anteriormente. Para el resto de la implementación construímos un prototipo de Kerberos escrito en Java (mediante la extensión criptográfica de Java) y basado en el diseño previamente modelado y validado. Para ello se instaló el kit Cyberflex para Linux, procesador intel x86, Linux x86: RedHat 6.1 x86, así como la interfaz de comunicación PCSC-lite y los drivers del smart reader de Towitoko. Se utilizó la tarjeta Cyberflex Access 16K de SchlumbergerSema y con estos recursos se pudo levantar el demonio pcscserver para la comunicación con la tarjeta:

```
>run pcscserver  >run Xcard   >run pay
```

La versión *pay* incluída en el kit de Linux resultó no funcional pero esta herramienta viene también en otro paquete del CITI, el Cyberflex i386_linux2, con ello se probaron comandos APDU para la prueba del Cardlet. Escribimos el código Java JCE JPCSC del Emulador de Kerberos: KDC_module.java es el centro de distribución de llaves de Kerberos, C_module.java es el cliente de Kerberos y el Cardlet dentro de la tarjeta es Smart.java. KDC_module y C_module se comunican a través de un canal seguro por medio de las funciones criptográficas de JCE. A su vez, C_module se comunica con Smart por medio de la librería JPCSC de IBM. Aquí la clase Context habilita las funciones PCSC relacionadas con la conexión/desconexión de los servicios PCSC y los card readers. Una invocación a Connect() regresa una conexión Card permitiendo la transmisión de APDUs.

```
Context ctx = new Context();
```

Connect regresa un objeto Card si se conecta exitosamente:

```
Card card = ctx.Connect(sa[0], PCSC.SHARE_EXCLUSIVE,
PCSC.PROTOCOL_T1);
```

Transmit transfiere datos a Card y recibe respuesta de Card:

```
byte[]  ret  =  card.Transmit(0x00,  0xA4,  0x04,  0x00,  aid,  20);
```

Disconnect termina la sesión con Card: `card.Disconnect(PCSC.LEAVE_CARD);`

En la tarjeta debe correr la aplicación Cardlet de resguardo y decripción de tokens. Para su construcción nos ajustamos a los siguientes lineamientos [13].

Primero se importan las clases criptográficas de la Java Card :

```
import javacardx.crypto.*;
```

después se definen las constantes APDU:

```
final byte Smart_CLA              = ( byte ) 0x03;
final static byte Select   = ( byte ) 0xA4;
final static byte Decrypt  = ( byte ) 0x10;
```

y por último se define el constructor:.

```
private Smart() {
     deskey = new DES_Key((short)KEY_BASE);
     deskey.setICV (IV, (short)0);
```

El método install es invocado por el JCRE como el último paso en el proceso de instalación del applet.

```
public static void install (APDU apdu){ new Smart(); }
```

El JCRE llama el método select para informar a la tarjeta que esta applet es seleccionado.

```
public boolean select (){
       pin.reset();  return true;    }
```

El JCRE envía los APDUs de entrada al método process.

```
public void process (APDU apdu){
       apdu_bufer = apdu.getBuffer();
       switch (apdu_bufer[ISO.OFFSET_INS] ){
         …     case Select: return;
               case Decrypt: decrypt(apdu); return; } }
```

El método de decripción se construye de la manera siguiente[14]:

```
private void decrypt(APDU apdu) {
 short byteRead = (short)(apdu.setIncomingAndReceive());
 deskey.decryptCBC(apdu_bufer, (short)ISO.OFFSET_CDATA, byteRead,
                 apdu_bufer, (short)ISO.OFFSET_CDATA); }
```

Mediante el compilador de java compilamos Smart.java.

Después convertimos Smart.class en Smart.bin por medio de la herramienta *mksolo* incluída en el kit de Linux:

```
mksolo - usemap=/lib/Access.map - v - d Smart.class
```

Con el siguiente comando de *pay* cargamos el applet en la tarjeta:

```
pay > jl -p 77.77 -c 77.78 -a 536D617274 -i 1024 -s 2048
Smart.bin
```

La llave se carga en la memoria de la tarjeta de la siguiente manera:

```
pay > jk 1
ca_load_key buf=jk 1
key 0 : <- 1A2B3C4D5E6F7B8C
```

Una vez instanciado el applet en la tarjeta, podemos ejecutar el servidor KDC_module y el cliente C_module conectados en red, los cuales se comunican mediante la infraestructura de transporte de Java y la superestructura criptográfica de la JCE cuyos detalles omitimos aquí:

```
java KDC_module 100
java C_module kdc.cem.itesm.mx 100
```

La instancia C_module del host Cliente se comunica ahora con la instancia del card Smart, transfiriéndole el mensaje encriptado y recibiendo la llave de sesión como respuesta.

5. Incorporación de un coprocesador seguro en el KDC

En esta sección describiremos una tecnología reciente desarrollada por IBM basada en un coprocesador, que puede engrandecer la seguridad de trabajos futuros relacionados, sobre todo del lado del servidor de Kerberos V5, que no ha sido discutido hasta ahora. Un coprocesador seguro es un dispositivo computacional confiable para la ejecución correcta de software, a pesar de posibles ataques físicos. Por ejemplo el coprocesador IBM 4758 es un dispositivo PCI card resistente y sensitivo a infiltraciones, la tarjeta detecta intentos de apertura, intentos de penetración, ataques de temperatura y ataques de radiación. Las aplicaciones que corren en la tarjeta están escritas en C. Existen varias opciones para incorporar un coprocesador de estas características a nuestro diseño: Una opción es trasladar el KDC completo al coprocesador, pero afectaría el rendimiento y la escalabilidad del KDC. Otra opción, más adecuada, es dividir el KDC en dos partes, una en el host (KDC-host) y otra en el coprocesador (KDC-cop).

En este esquema se respeta que toda llave guardada en el coprocesador nunca lo abandone en texto claro y que todas las operaciones criptográficas sean ejecutadas en el coprocesador. La llave maestra es almacenada en la memoria de respaldo RAM del coprocesador y nunca se mueve de ahí. Debido a limitaciones de espacio, las llaves de los usuarios son almacenadas en el disco del KDC-host pero están encriptadas con la llave maestra del KDC. Cuando se necesita la llave de un usuario para encriptar un ticket, el coprocesador baja la llave (encriptada) del KDC-host y la decripta con la

llave maestra del KDC, después la utiliza para la operación criptográfica y por último la elimina de su memoria. Por su parte las llaves de sesión, que son más cortas, son generadas en el KDC-cop y guardadas en los tickets.

Por último, podemos agregar al esquema anterior la autentificación smart pin bio selectiva, esto es, la posibilidad de contar con múltiples plantillas biométricas en la tarjeta del usuario y pretender una prueba selectiva, aleatoria, parecida a una prueba de tipo liveness. Así mismo, podemos plantear la autentificación mutua bio-smart selectiva del servidor final (o sea Bob, en el análisis que sigue). Aquí, Kerberos actúa como T3P para la distribución segura de llaves de sesión, hay que recordar que Kerberos es, en principio, un protocolo de autentificación entre el cliente y el servidor final.

5.1 Análisis de Seguridad

Someteremos este último protocolo a un mayor análisis mediante un modelo simple del sistema, el cual debe cumplir con dos condiciones.

La primera consiste en que la llave maestra es almacenada en KDC-cop y en ningún otro lado, y las llaves de Alice y Bob son almacenadas en el KDC-host encriptadas con la llave maestra.

La segunda asume que la estación de trabajo del cliente es segura, porque la llave secreta de Alice K_A está guardada en su tarjeta (que es activada por medio del NIP de Alice), así como su colección de plantillas biométricas: P_{A1} ...P_{An}. Por lo cual la estación no puede perder información de Alice y no puede alterar ni modificar mensajes que Alice envía a la red. 3) El cifrador DES es fuerte, se asume imposible de romper en un tiempo corto de tiempo (Kerberos eventualmente reemplazará DES con 3DES, eliminando los ataques de fuerza bruta sobre DES).4) El atacante puede comprometer el host de Alice y Bob, y el KDC-host. El atacante puede leer y modificar cualquier información en el host de Alice y Bob, y en el KDC-host. No obstante, El atacante no puede comprometer KDC-cop. El atacante no puede leer ni modificar la información en KDC-cop. Al intentarlo, el coprocesador elimina toda la información en él. El atacante no puede influir el comportamiento del coprocesador. 5) El atacante puede leer, modificar y alterar mensajes en la red que conecta a todos los principales del reino de Kerberos, pero no puede monitorear el canal entre el host de Alice (y de Bob) y su tarjeta, porque asumimos que el canal es local al host del usuario y está libre de infiltraciones (Adicionalmente las Java Card cuentan con mecanismos interconstruidos de autentificación mutua con el host). Por último, asumimos que el atacante puede ser un principal del reino de Kerberos. Dadas las condiciones anteriores queremos averiguar si las llaves del KDC pueden ser obtenidas por un atacante, o bien si puede ocurrir alguna suplantación de usuarios, o quizás la falsificación o reenvío de un ticket usado.

En el primer caso el atacante no puede robar cualquier llave del KDC-host. La llave maestra está en el coprocesador y no es accesible al atacante (Condiciones 1 y 4). Las demás llaves están en KDC-host, pero están encriptadas con DES por la llave maestra, el cual se asume irrompible (Cond. 3). En ausencia del coprocesador el atacante podría robar todas las llaves y comprometer KDC-host (esto es posible por la Cond. 4).

Ahora probaremos que el atacante no puede suplantar a ningún usuario. Primero, el atacante no puede robar la llave privada de un usuario, porque permanece

almacenada en la tarjeta del usuario y en el coprocesador seguro del KDC. Segundo, la otra forma de suplantar a un usuario (p.e. Alice) es obtener un ticket $\{A, B, K_{AB}\}K_B$ y la llave de sesión K_{AB}. El atacante puede obtener un ticket monitoreando la red (Cond. 4), pero la llave de sesión (de corta duración) permanece encriptada en el host de Alice con su llave privada, la cual está guardada en la tarjeta de Alice, que está asegurada por medio de la autentificación biométrica, por lo que no es posible la suplantación final de Alice (Cond. 2). La llave de sesión es generada en el coprocesador y va encriptada por K_A o K_B cuando se dirije al KDC-host, pero se asume que K_A, K_B ó K_{AB} son fuertes (Cond. 3), por lo tanto la llave de sesión no puede ser obtenida por el espía Spy(M). Sin coprocesador y sin tarjetas, el atacante puede suplantar a cualquier usuario, ya sea robando la llave privada o el ticket del usuario. Finalmente, el atacante no puede generar un ticket a su favor. El coprocesador genera los tickets sólo después de que Alice muestra la posesión de su llave a través de la preautentificación. Sin coprocesador el atacante puede generar o reenviar cualquier ticket, al usar llaves robadas.

6. Resultados obtenidos

Después de obtener la especificación del modelo ASM de Bio Smart Kerberos, logramos el prototipo del modelo de Bio Smart Kerberos escrito en Promela, que contempla la interacción del Cliente con los módulos Biométrico y Smart. Posteriormente pasamos a la simulación del prototipo, en la siguiente modalidad: random simulation style & hash compact search verification mode. El método de búsqueda fue Search Mode: Exhaustive, Supertrace/Bitstate, Hash-Compact. No se encontraron errores durante la simulación ni la validación automatizada, que se realizó bajo el siguiente esquema: Correctness Properties Safety (state properties), Assertions, Liveness, Acceptance Cycles, Apply Never Claim. Se probó la exigencia temporal siguiente, que en términos simples establece que no puede ocurrir que un usuario autentificado y no autorizado, permanezca no autorizado indefinidamente:

```
never { do:: authenticated && ! authorised -> goto accept od;
        accept: do ::! authorised od;}
```

De igual forma se valida la pre-autentificación biométrica con tarjeta activada por PIN. Así mismo mediante la validación manual se probó el Teorema de Correctez de la Fase de Autentificación que incluye pre-autentificación biométrica con tarjeta, llegándose al siguiente resultado:

$[\mathit{defined}\ (\ \mathit{Ticket}(C, TGS))]\ S_{19}\ \ \&\ \ [\mathit{defined}\ (\ K(C, TGS))]\ S_{19}$

por lo cual, un cliente legítimo que solicita autentificación ante *KAS* obtiene un ticket y una llave de autentificación del módulo Smartcard. Finalmente se planteó una optimización del prototipo mediante autentificación biométrica selectiva y la incorporación de un coprocesador seguro en el KDC, aunque es posible también la autentificación biométrica mutua del servidor final. Se analizaron diversos casos de ataques y puede decirse que en protocolo presentado la pre-autentificación biométrica

es válida y completa, es decir, un usuario ilegítimo no será aceptado por el sistema de autentificación, mientras que todo usuario legítimo obtendrá el acceso deseado.

7. Conclusiones

Se presentó un esquema de integración de biométrico/smart card en el protocolo de Kerberos. La principal ventaja radica del lado de confiabilidad del protocolo. Si un intruso intercepta esta cadena de datos puede reenviarla posteriormente (lo cual no es realmente factible por tratarse de una terminal bio-inteligente local a la estación de trabajo), o bien reconstruir la llave Kb a partir de los datos obtenidos, que tampoco es fácil de lograr dada la cantidad de entropía de la que ha sido dotada la llave, haciéndola prácticamente inalcanzable por medio de una búsqueda exhaustiva, a pesar de la información que un atacante pueda disponer acerca de la llave.

A diferencia de la mayoría de los protocolos de seguridad, aquí no se asume la seguridad de componentes de bajo nivel de las computadoras como son los sistemas operativos, porque un adversario puede recargar el Kernel. Para resolver este problema, se construye hardware de propósito especial y se diseña un sistema operativo a su alrededor. Itoi y Honeyman califican su diseño como un enfoque más pragmático y experimental de rápida implementación y despliegue. Sin embargo aquí prevalece la informalidad del hardware y los sistemas operativos, de los cuales no se puede tener toda la seguridad (por ejemplo, en el caso de que un sistema operativo sea completamente reemplazado).

Como comentario final cabe mencionar que hasta donde es de nuestro conocimiento Schlumberger ha lanzado una terminal que ofrece una solución de autentificación de doble factor, que consiste de un reader PCMCIA y un Biomouse con escáner Fingerprint. Aunque el software es propietario no se menciona relación alguna con el protocolo de Kerberos.

En este momento se está trabajando en escribir un prototipo del modelo de Pre-autentificación Smart PIN Bio Selectiva y Coprocesador Integrado en el KDC, que contemple la interacción del Cliente con los módulos Biométrico y Smart, así como del KDC con el Coprocesador, para después llevar a cabo una simulación del prototipo en la siguiente modalidad: random simulation style & hash compact search verification mode. El método de búsqueda se hace en Search Mode: Exhaustive, Supertrace/Bitstate, Hash-Compact. Lo anterior permitirá validar el teorema de Correctez de la Fase de Pre-Autentificación que incluye pre-autentificación biométrica selectiva con tarjeta y coprocesador en el KDC.

Referencias

1. J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. Proceedings Winter USENIX, Conference, Dallas, February 1988, Usenix
2. Naomaru Itoi; Peter Honeyman. Smartcard Integration with Kerberos V5. CITI Technical Report 98-7, 1998.
3. Giampaolo Bella; Elvinia Riccobene. Formal Analysis of the Kerberos Authentication System. Journal of Universal Computer Science, 1997, Vol. 3, N° 12, p. 1337-1381
4. Rob Jerdoney, et al. Implementation of a Provably Secure, Smartcard-based Key Distribution Protocol. CITI Technical Report. 98-4

5. Giampaolo Bella. Modelling Security Protocols Based on Smart Cards. Computer Laboratory, University of Cambridge.
6. Luis Trejo; Carlos Sandoval. Diseño y Validación de Protocolos de Comunicación. Manual del curso. División de Ingeniería y Ciencias, ITESM, Campus Estado de México. Enero de 1997
7. Pohlmann Norbert. Forget About PINs. Chairman, Tele Trus T Association, 2002
8. Java Card ™ Biometric API White Paper (Working Document). Biometric Consortium Interoperability, Assurance, and PerformanceWorking Group. 7 August 2002, Version 1.1, Document # 02-0016
9. BioAPI Specification. The BioAPI Consortium. March 16, 2001, Version 1.1
10. Zdenek Ríha; Václav Matyás. Biometric Authentication System. Faculty of Informatics Masaryc University Report Series FIMU-RS-2000-08, 2000.
11. Jorge Ferrari, et al. Smart Cards: A Case Study. International Technical Support Organization. http://www.redbooks.ibm.com
12. Cyberflex™ Access Software Development Kit Release 3C. Cyberflex Access Programmer's Guide. 2000, C300451.
13. Cyberflex™ Cards Programmer's Guide, Cyberfex Access Software Development Kit 4.3, 2002, C300474_rev1.
14. Java Card Applet Developer's Guide. SUN Microsystems. Revision 1.10, July 17, 1998