

Autenticación Revisitada

Kerberos

Desarrollado como parte del Proyecto *Athena* en MIT, con el propósito de autenticar clientes a servidores y vice versa. Se considera que las siguientes amenazas existen:

- Manuel puede acceder físicamente a la estación de trabajo de Alicia (en su ausencia) y aparentar ser ella.
- Manuel puede alterar la dirección de red de una estación para que parezca a otra.
- Manuel puede escuchar el tráfico entre cliente y servidor y usar un ataque de *replay*.

La filosofía de Kerberos es que un servidor central provee servicios de autenticación para usuarios y otros servidores. Se asume que el servidor central está físicamente protegido.

Hay dos versiones de Kerberos: Versión 4 (ahora considerada muerta) y Versión 5. Ambas usan criptografía simétrica. V4 especifica DES, mientras V5 permite distintos sistemas.

Una variante de V5 es usado por Microsoft en su producto Active Directory.

Requisitos

Los siguientes fueron metas de Kerberos a la hora de diseño del sistema:

- Que fuera suficientemente fuerte para no constituir el punto débil del sistema Athena.
- Todo servicio que usa Kerberos para autenticación depende totalmente de su confiabilidad, por lo tanto esta debe ser alta. Debe permitir que un servidor respalde a otro.
- Con excepción de pedirle una contraseña, el sistema debe ser transparente para el usuario.
- Debe ser capaz de soportar grandes números de clientes, usuarios y servidores. Esto sugiere una arquitectura modular y distribuida.

La técnica de autenticación está basada en el protocolo de Needham y Schroeder. Sin embargo su implementación concreta es mucho más compleja. Describimos el sistema en una serie de versiones cada vez más refinadas. La última de estas corresponde a V4.

Primera Versión

Sea C un cliente (un proceso corriendo en una estación de trabajo, ej. para correo electrónico), A un servidor de autenticación, S otro servidor (ej. de correo electrónico), I_C el identificador de un usuario de C , P_C la contraseña de dicho usuario, D_C la dirección de red de la estación de C , y K_S una clave compartida entre A y S .

$C \rightarrow A$	I_C, P_C, I_S
$A \rightarrow C$	Ticket
$C \rightarrow S$	I_C, \mathbf{Ticket}

donde **Ticket** = $\{I_C, D_C, I_S\}_{K_S}$. Se incluye D_C para garantizar que el *ticket* sólo puede ser usado desde el mismo punto que lo solicitó.

Hay dos problemas principales con éste método:

- Para no guardar una copia de P_C en la estación, hay que pedirselo al usuario cada vez que C necesita contactar a S . Esto puede mejorar si los *tickets* son reutilizables, pero todavía se necesita uno para cada tipo de servicio.
- El primer mensaje ($C \rightarrow A$) requiere transmitir P_C sobre la red, lo cual es vulnerable.

Segunda Versión

Introducimos un nuevo servidor T , cuya función es emitir *tickets* (se llama un **TGS** o *Ticket Granting Server*). Ahora tenemos:

Una vez por sesión de usuario:

$C \rightarrow A$ I_C, I_T

$A \rightarrow C$ $\{\mathbf{Ticket}_T\}_{K_C}$

Una vez por tipo de servicio:

$C \rightarrow T$ $I_C, I_S, \mathbf{Ticket}_T$

$T \rightarrow C$ \mathbf{Ticket}_S

Una vez por uso del servicio:

$C \rightarrow S$ I_C, \mathbf{Ticket}_S

donde $\mathbf{Ticket}_T = \{I_C, D_C, I_T, X_1, V_1\}_{K_T}$ (o sea, un “*ticket para pedir tickets*”) y $\mathbf{Ticket}_S = \{I_C, D_C, I_S, X_2, V_2\}_{K_S}$ (o sea, un “*ticket para pedir el servicio S*”).

Ticket_T es enviado por *A* utilizando una clave K_C , derivada de P_C , que *A* conoce. Por lo tanto *C* puede decriptar la respuesta de *A* y guardar el *ticket*. **Ticket_T** es utilizado cuando *C* necesita pedir un *ticket* para algún servicio nuevo. Las siguientes veces, utiliza **Ticket_S**.

Dado que los *tickets* son reutilizables, es importante evitar que Manuel intente un ataque de *replay*. Para reducir sus posibilidades, cada *ticket* incluye un **timestamp**, X_i , y un período de validéz V_i .

Todavía quedan dos problemas:

- ¿Qué valores se deben usar para los V_i ? Si son cortos, habrá que pedirle la contraseña al usuario con mucha frecuencia. Si son largos, aumenta la posibilidad de un ataque *replay*. En efecto, se tiene un requerimiento adicional: *T* o *S* debe poder comprobar que el usuario de un *ticket* es aquello a quien se le entregó.

La manera de lograr esto es que *A* le entregue un secreto a *C* y a *T* que sólo ellos conocen. Kerberos usa una clave de sesión para esto.

- Manuel podría simular *S* (sólo tiene que recibir *tickets* y “aprobarlos”). Existe una necesidad para la autenticación de servidores a usuarios.

Kerberos Versión 4

Para Obtener un Ticket para Pedir Tickets

$C \rightarrow A$ I_C, I_T, X_1

$A \rightarrow C$ $\{K_{C,T}, I_T, X_2, V_2, \mathbf{Ticket}_T\}_{K_C}$

Para Obtener un Ticket de Servicio

$C \rightarrow T$ $I_S, \mathbf{Ticket}_T, \mathbf{Aut}_C$

$T \rightarrow C$ $\{K_{C,S}, I_S, X_4, \mathbf{Ticket}_S\}_{K_{C,T}}$

Para Obtener un Servicio

$C \rightarrow S$ $\mathbf{Ticket}_S, \mathbf{Aut}'_C$

$S \rightarrow C$ $\{X_5 + 1\}_{K_{C,S}}$

Donde $\mathbf{Ticket}_T = \{K_{C,T}, I_C, D_C, I_T, X_2, V_2\}_{K_T}$,

$\mathbf{Ticket}_S = \{K_{C,S}, I_C, D_C, I_S, X_4, V_4\}_{K_S}$,

$\mathbf{Aut}_C = \{I_C, D_C, X_3\}_{K_{C,T}}$, y $\mathbf{Aut}'_C = \{I_C, D_C, X_5\}_{K_{C,S}}$. Estos últimos se llaman **autenticadores** (*authenticators*).

Los autenticadores comprueban la identidad de quienes los generan, ej. Aut_C significa “A la hora X_3 yo conozco la clave $K_{C,T}$ ”. Cada autenticador se usa una sola vez y tiene vigencia muy corta.

El último paso del protocolo es para la mutua autenticación. X_5 es usado como un *nonce*, y S demuestra que lo conoce. Además, al final C y S comparten una clave secreta que pueden usar para privacidad de la sesión. Esto no forma parte de Kerberos.

Un servidor A más un grupo de clientes y usuarios constituyen un **reino** (*realm*) de Kerberos. Existe un protocolo adicional que permite a un reino autenticar a los usuarios de otro, bajo mutuo acuerdo.

Kerberos Versión 5

V4 tiene varias deficiencias que V5 trata de corregir:

- V4 depende de DES, lo cual era un problema por las leyes exportación de EE.UU. En V5 esto se puede parametrizar.
- V4 requiere de direcciones IPv4. V5 permite otros tipos de dirección (OSI, IPv6, ...).
- V4 permite dos ordenamientos de byte. V5 expresa los mensajes en ASN.1.
- En V4, las vigencias se expresan como un valor de 8 bits, en unidades de 5 minutos. En V5, se da la hora de comienzo y terminación del período de validéz.
- En V4, un servidor no puede pedir servicio a otro por parte de un cliente. V5 permite pasar los credenciales de un cliente entre servidores.
- V4 encripta los *tickets* dos veces (segundo y cuarto pasos). El segundo encriptamiento no hace falta y V5 lo quita.
- V4 usa un modo de encriptamiento DES no-estándar, que resultó ser vulnerable. V5 usa modo CBC.

Un problema que existe en ambas versiones es que son vulnerables de un ataque a la contraseña del usuario (por los métodos tradicionales). Además, el servidor *A* tiene que ser físicamente seguro, porque guarda las claves de todos los usuarios.

Autenticación con Criptografía Asimétrica

Los sistemas de claves públicas parecen resolver el problema de distribución de claves, pero ¿cómo hace Alicia para obtener la clave pública de Bob? Debe utilizar una base de datos pública, manejada por Arturo, pero esta tiene que ser protegida contra ataques (ej. si Manuel sustituye su clave pública por la de Bob, ¿qué pasa?).

Incluso si la base de datos está bien protegida, Manuel puede interferir con la comunicación entre Arturo y los demás. Para evitar esto, Arturo debe firmar las claves que envía con *su* clave privada. Su clave pública es *bien conocida* y por lo tanto difícil de alterar sin llamar la atención.

La información enviada por Arturo, y firmada por él, se llama un **certificado**, y Arturo se llama un **Certification Authority (CA)**. Para que no se convierte en un cuello de botella (y punto único de falla) el CA puede **delegar** parte de su responsabilidad a otro, en forma jerárquica. El CA de máximo nivel se llama el **Root CA**. Una estructura de CA's, con sus procesos de registro de identidades y emisión de certificados, se llama un **Public Key Infrastructure (PKI)**.

Certificados X.509v3

(Véase [http:](http://developer.netscape.com/docs/manuals/security/pkin/contents.htm#1047709)

[//developer.netscape.com/docs/manuals/security/pkin/contents.htm#1047709](http://developer.netscape.com/docs/manuals/security/pkin/contents.htm#1047709)).

La especificación X.509 Versión 3 es un estándar internacional recomendado por la UIT (Unión Internacional de Telecomunicaciones), y usado en la mayoría de los navegadores (Netscape, Internet Explorer etc.), además de otras aplicaciones en el Internet.

Un certificado X.509 asocia un *Distinguished Name* (DN) con una clave pública. Un DN consiste de un conjunto pares nombre–valor que identifican unívocamente a un sujeto, por ejemplo:

```
uid=doe,e=doe@netscape.com,cn=John Doe,  
o=Netscape Communications Corp.,c=US
```

La sintaxis es derivada del sistema X.500 de directorios, también usado por servidores LDAP (*Lightweight Directory Access Protocol*). El sistema es flexible, así que pueden haber otros pares, a elección del administrador.

Cada certificado X.509 consiste de dos secciones:

La **sección de datos** incluye:

- Número de versión (ej. v3).
- Número serial del certificado, asignado por la CA.
- Información sobre la clave pública del usuario, incluyendo el algoritmo usado y la clave en sí.
- El DN de la CA que emitió el certificado.
- Período de validez (comienzo y fin).
- DN del sujeto (dueño).
- Algunos datos opcionales, por ejemplo para indicar que el certificado es para firmar mensajes de correo.

La **sección de firmas** incluye:

- El algoritmo criptográfico usado por la CA para crear su propia firma digital.
- Un hash del certificado, firmado con la clave privada de la CA.

Ejemplo

Certificate:

Data:

Version: v3 (0x2)
Serial Number: 3 (0x3)
Signature Algorithm: PKCS #1 MD5 With RSA Encryption
Issuer: OU=Ace Certificate Authority, O=Ace Industry, C=US
Validity:

Not Before: Fri Oct 17 18:36:25 1997

Not After: Sun Oct 17 18:36:25 1999

Subject: CN=Jane Doe, OU=Finance, O=Ace Industry, C=US

Subject Public Key Info:

Algorithm: PKCS #1 RSA Encryption

Public Key:

Modulus:

00:ca:fa:79:98:8f:19:f8:d7:de:e4:49:80:48:e6:2a:2a:86:
ed:27:40:4d:86:b3:05:c0:01:bb:50:15:c9:de:dc:85:19:22:
43:7d:45:6d:71:4e:17:3d:f0:36:4b:5b:7f:a8:51:a3:a1:00:
98:ce:7f:47:50:2c:93:36:7c:01:6e:cb:89:06:41:72:b5:e9:
73:49:38:76:ef:b6:8f:ac:49:bb:63:0f:9b:ff:16:2a:e3:0e:
9d:3b:af:ce:9a:3e:48:65:de:96:61:d5:0a:11:2a:a2:80:b0:
7d:d8:99:cb:0c:99:34:c9:ab:25:06:a8:31:ad:8c:4b:aa:54:
91:f4:15

Public Exponent: 65537 (0x10001)

Extensions:

Identifier: Certificate Type

Critical: no

Certified Usage:

SSL Client

Identifier: Authority Key Identifier

Critical: no

Key Identifier:

f2:f2:06:59:90:18:47:51:f5:89:33:5a:31:7a:e6:5c:fb:36:
26:c9

Signature:

Algorithm: PKCS #1 MD5 With RSA Encryption

Signature:

6d:23:af:f3:d3:b6:7a:df:90:df:cd:7e:18:6c:01:69:8e:54:65:fc:06:
30:43:34:d1:63:1f:06:7d:c3:40:a8:2a:82:c1:a4:83:2a:fb:2e:8f:fb:
f0:6d:ff:75:a3:78:f7:52:47:46:62:97:1d:d9:c6:11:0a:02:a2:e0:cc:
2a:75:6c:8b:b6:9b:87:00:7d:7c:84:76:79:ba:f8:b4:d2:62:58:c3:c5:
b6:c1:43:ac:63:44:42:fd:af:c8:0f:2f:38:85:6d:d6:59:e8:41:42:a5:
4a:e5:26:38:ff:32:78:a1:38:f1:ed:dc:0d:31:d1:b0:6d:67:e9:46:a8:
dd:c4

SSL

(Véase <http://developer.netscape.com/docs/manuals/security.html#SSL>).

SSL (*Secure Sockets Layer*) es un protocolo, originalmente de Netscape pero ahora usado casi universalmente para comunicación segura entre clientes y servidores de Web. Es la base para TLS (*Transport Layer Security*), un esfuerzo de estandarización del IETF.

SSL se ubica entre TCP/IP y los protocolos de aplicación como HTTP o IMAP. Permite que el cliente y el servidor se autentican mutuamente, usando certificados X.509, y que puedan comunicar por una conexión encriptada.

Hay dos sub-protocolos: el *record protocol* define el formato de datos, mientras y el *handshake protocol* define el intercambio de mensajes necesario para establecer una conexión SSL.

SSL soporta una variedad de algoritmos criptográficos, incluyendo DES, Triple DES, DSA (*Digital Signature Algorithm*), MD5, RSA, SHA-1 y otros.

El Protocolo Inicial SSL

El propósito del protocolo inicial o “*handshake*” es:

- Llegar a un acuerdo sobre los algoritmos criptográficos a ser usados.
- Autenticar el servidor al cliente
- Autenticar el cliente al servidor (opcional).
- Usar técnicas de claves públicas para generar secretos compartidos.
- Establecer una conexión encriptada.

Los pasos a seguir son:

1. Cliente envía el número de versión de SSL que utiliza, los algoritmos que entiende, un *nonce*, etc.
2. Servidor hace lo mismo, y agrega su certificado de autenticidad. Opcionalmente, solicita el certificado del cliente.
3. Cliente autentica el servidor (ver más adelante). Si fracasa, manda una advertencia al usuario. En caso contrario, sigue.
4. Cliente utiliza los datos intercambiados hasta ahora para generar el “*pre-master secret*” (*PS*), el cual es encriptado con la clave pública del servidor y enviado a éste.

5. Si el servidor ha solicitado que el cliente se autentique, éste también firma un dato adicional que es único para esta sesión y es conocido por ambas partes. Envía el dato firmado más su propio certificado junto con el *PS*.
6. Si se requiere autenticación del cliente, el servidor intenta comprobar la misma. Si no puede, la sesión se termina. En el caso contrario, el servidor usa su clave privada para decriptar el *PS*, y lo utiliza para generar el “*master secret*” (*MS*). El cliente hace lo mismo.
7. Ambas partes usan el *MS* para generar claves de sesión simétricas, las cuales sirven para la confidencialidad e integridad de los mensajes.
8. Cliente informa a servidor que va a comenzar a usar la clave de sesión, y separadamente envía un mensaje encriptado para indicar que el protocolo inicial ha concluido.
9. Servidor hace lo mismo.
10. Ambas partes usan la clave de sesión para comunicarse.

Autenticación de las Partes

Para autenticar la asociación entre una clave pública y el servidor mencionado en el certificado X.509 correspondiente, el cliente pregunta:

1. ¿La hora y fecha actuales se encuentran dentro de la ventana de vigencia del certificado?
2. ¿Se confía en el CA emisor? El cliente tiene una lista de CA's en los cuales confía. Si el CA del emisor del certificado no se encuentra en la lista, se usa un proceso iterativo para preguntar si alguno de los que sí están lo valida, o si alguno que ellos conocen lo hace, etc.
3. ¿La clave pública del emisor convalida la firma digital del certificado?
4. ¿El nombre de dominio en el certificado corresponde al nombre de dominio del servidor? (Estrictamente, esto no es parte de SSL, pero se usa para evitar ataques tipo “*Man-In-The-Middle*”). Se supone que el DNS no ha sido atacado . . .

La autenticación del cliente por parte del servidor sigue un proceso similar, aunque con algunas diferencias de detalles.

Comentarios

- Muy pocos clientes de SSL tienen certificados propios. Para el comercio electrónico esto es poco importante, pero en otras aplicaciones sí importa.
- Los Root CA's conocidos son comerciales (Verisign, Thawte, etc.) y los navegadores populares contienen sus certificados, pero para crear un PKI local se necesitan mecanismos para convencer a los navegadores que el CA local sea válido. La forma fácil es pagando a Verisign ...
- Los vendedores de certificados asocian un nombre con una etiqueta DNS, pero no tienen autoridad sobre ninguno de los dos. Verisign dice explícitamente que no garantiza que el poseedor de un certificado determinado tiene derecho legal de usar el nombre correspondiente.
- Los mecanismos de revocación de certificados son engorrosos (hay que bajar una "lista de certificados revocados" periódicamente) y pocos clientes los implementan en forma fácil de usar.
- La situación legal no está clara: existe la presunción que un documento firmado con mi clave privada fue firmada por mí. Esto es más fuerte de lo que se presume con las firmas tradicionales.