Kerberos

Mihai Budiu — `mihaib+@cs.cmu.edu`
`http://www.cs.cmu.edu/~mihaib/`

20 iulie 1998

Subiect: Protocolul de autentificare Kerberos

Cunoştinţe necesare: cunoştinţe elementare de matematica, fundamente de Unix

Cuvinte cheie: autentificare, tichet, parola, securitate, funcţie neinversabilă

Cuprins

1 Securitate

O definiţie largă a noţiunii de "securitate" pentru calculatoare ar fi următoarea: interzicerea accesului unor persoane neautorizate la anumite date. Putem distinge două tipuri de restricţii: restricţii asupra citirii unor date secrete şi restricţii asupra modificării de către persoane ne-autorizate. Există mecanisme speciale pentru fiecare din aceste scopuri, şi există mecanisme care pot fi adaptate amîndurora.

1.1 Sisteme sigure

Înainte de a plonja în detalii să observăm că *orice schemă de securitate trebuie să înceapă cu securitatea fizică a unor dispozitive de calcul* (în sensul că accesul la aparatul însuşi este îngrădit). Dacă nu poţi avea încredere în nici un calculator, atunci nu poţi face nici un fel de comunicaţie sau stocare de date sigură. Dacă cineva are control asupra sistemul de operare al unui calculator atunci el poate intercepta toate tastele apăsate şi toate caracterele scrise pe ecran.

Atunci cînd lucraţi pe un calculator trebuie să aveţi deplină încredere cel puţin în consola la care lucraţi şi în celălalt capăt al liniei. Dacă nu puteţi garanta aceste lucruri nu trebuie să vă aşteptaţi la nici un fel de garanţii de securitate din partea sistemului. Marile bănci au calculatoarele îngropate în nişte buncăre subterane extrem de bine păzite; unul dintre motive este, desigur, acela de a nu pierde informaţii extrem de importante în cazul unei calamităţi, dar una dintre raţiunile primordiale este garantarea securităţii fizice a sistemului.

Oricare ar fi scopul pentru care vrem să protejăm date, politica pe care o aplicăm în acest sens va fi o funcţie de entitatea care vrea să acceseze datele. De pildă anumite documente militare vor fi secrete pentru gradele mici, dar vor putea fi accesate de un general.

Pentru a putea face astfel de distincţii este deci necesar să avem la dispoziţie un mecanism de *autentificare*.

2 Autentificare

Autentificarea (authentication) este procesul prin care se verifică identitatea unei instanţe (de exemplu un utilizator) care a produs nişte date. Cele mai sigure metode de autentificare sunt cele biometrice, care măsoară caracteristici fizice unice ale unui individ (cum ar fi amprentele digitale sau forma irisului). Acestea sunt foarte greu de păcălit. Din (ne)fericire sunt încă foarte costisitoare şi puţin răspîndite.

Metoda prin care în mod uzual calculatoarele identifică identitatea unei entităţi cu care comunică este verificarea că acea entitate ştie o informaţie care foarte probabil nu mai este cunoscută de nimeni altcineva. În sistemele tipice Unix acel secret este *parola* pe care utilizatorul trebuie să o tasteze înainte de a începe lucrul.

Să mai observăm că pentru a putea vorbi despre autentificare trebuie să avem o noţiune de *identitate* a utilizatorilor. Calculatorul trebuie să poată distinge cumva între diferiţii indivizi care îl folosesc. Trebuie să existe un *spaţiu de nume* pentru utilizatori; autentificarea atunci va pune în corespondenţă o entitate activă cu un astfel de nume. În Unix numele utilizatorilor autorizaţi sunt trecute într-o mică bază de date într-un fişier numit `/etc/passwd` (sistemele mai moderne au scheme mai complicate, dar înrudite); pentru un sistem Unix a autentifica un utilizator constă în a asigna unul dintre aceste nume cunoscute unui set de procese executate de acel utilizator. Dacă un individ nu are un nume în acel fişier atunci el nu există pentru calculator.

Cu alte cuvinte, pentru a putea vorbi despre autentificare trebuie ca părţile implicate în comunicare să aibă un spaţiu de nume comun pentru utilizatori.

3 Criptografie

Dacă întreg sistemul cu care lucrăm este sigur fizic atunci nu avem mare nevoie de autentificare; noţiunea de securitate fizică implică faptul că persoane neautorizate nu pot accesa sistemul. De îndată însă ce datele trebuie să traverseze porţiuni nesigure trebuie să luăm măsuri suplimentare de precauţie. Transformarea datelor în scopul de a împiedica accesul persoanelor neautorizate se numeşte *criptare*. Criptarea transformă un text *inteligibil* (plaintext) într-un *cifru* (ciphertext). Procesul de codificare se numeşte *cifrare* sau *criptare* (encryption) iar procesul invers se numeşte descifrare sau *decriptare* (decryption).

3.1 Funcţii "ne-inversabile"

Putem vedea criptarea unui mesaj ca o transformare a mesajului. În mod normal această transformare este una funcţională, în sensul că unui mesaj îi corespunde un singur cod. Pentru a putea folosi la ceva mesajele codificate, trebuie să existe şi o transformare inversă, prin care dintr-un cifru obţinem textul iniţial.

Dacă notăm funcţia de criptare cu E (de la "encryption") şi cea de decriptare cu D, pentru un mesaj x trebuie să avem relaţia $D(E(x)) = x$. De aici rezultă în primul rînd că funcţia E trebuie să fie injectivă (altfel inversa nu este bine definită), cu alte cuvinte oricare două codificări ale unor cuvinte diferite trebuie să fie diferite.

În realitate funcţiile de criptare şi decriptare mai au un parametru relativ mic (comparat cu lungimea mesajului x), numit *cheia* de criptare. Cheia este un obiect relativ uşor de descris şi manipulat, spre deosebire de funcţiile D şi E. E va fi atunci de forma $E(x, k)$, unde k este cheia. $D(x, k_1)$ este funcţia de decriptare, k_1 fiind cheia pentru decriptare.

Se deosebesc două feluri de sisteme de criptare: simetrice, în care $k = k_1$ şi asimetrice în care cheile sunt diferite.

Ceea ce este important este că funcţiile D şi E sunt cunoscute de toată lumea; atunci cînd vreau să stabilesc un canal sigur cu un alt individ noi trebuie doar să cădem de acord asupra cheilor pe care le folosim (k şi k_1).

Nu oricare două funcţii care satisfac cerinţele anterioare sunt bune pentru criptare. Dacă fixăm cheia k şi notăm funcţia $E(k, \cdot)$ cu E_k, trebuie ca din cunoaşterea valorii lui $y = E_k(x)$ (cifrul) să fie practic imposibil de calculat x. Din cauza asta funcţiile de criptare se numesc (nu foarte precis) "neinversabile" sau "greu inversabile" (one-way functions).

Ce înseamnă de fapt "imposibil de calculat"? Pentru a răspunde la această întrebare ne trebuie ceva cunoştinţe de teoria complexităţii, pe care o să le omitem din prezentarea de faţă. În principiu asta înseamnă că orice algoritm care ar calcula cheia k ştiind y, D şi E trebuie să aibă o complexitate mai mare decît polinomială (de exemplu exponenţială).

Să observăm că există întotdeauna algoritmi care pot calcula inversa, pur şi simplu iterînd prin toate cheile posibile k şi văzînd dacă $D(y, k)$ este un cuvînt din limbajul de intrare. Dar dacă cheia are n biţi, trebuie încercate 2^n combinaţii diferite. Pentru un n suficient de mare sunt mult prea multe încercări, chiar şi pentru cel mai perfecţionat supercalculator.

De altfel unul dintre cele mai populare programe pentru "spart" parole pentru Unix, Crack, se bazează pe faptul că cheile de fapt nu sunt egal probabile; utilizatorii vor alege cu mare probabilitate anumite cuvinte (pe care le vor mutila în mici feluri), pentru că sunt mai uşor de memorat. "Crack" are la dispoziţie un dicţionar enorm şi o suită de reguli pentru a modifica cuvintele, şi pur şi simplu încearcă toate variantele; deşi asta poate părea mult, o limbă umană are în jur de 200 000 de cuvinte, o bagatelă pentru un calculator (comparaţi de pildă cu 128^8=72 057 594 037 927 936, cît ar fi toate combinaţiile de 8 caractere ASCII).

3.2 P = NP?

Într-un articol mai vechi din PC Report despre algoritmi am prezentat felurite clase de complexitate, printre care şi clasele P şi NP, ale problemelor a căror soluţie se poate calcula în timp polinomial, cu un calculator determinist, respectiv nedeterminist.

Problema P=NP este probabil cea mai importantă problemă deschisă din informatica; de peste 25 de ani s-au depus eforturi susţinute pentru a vedea dacă există probleme care se pot rezolva cu maşini nedeterministe (care "ghicesc" o parte din soluţie) repede dar nu şi cu maşini deterministe (cum sunt toate calculatoarele reale).

Ei bine, pentru a fi rezonabilă, problema aflării cheii cu care este criptat un mesaj nu poate fi mai complicată decît o problemă din clasa NP. Lucrurile se pot justifica perfect riguros; ideea de bază este că cel care cunoaşte cheia trebuie să decodifice relativ rapid mesajul primit (dacă decodificarea însăşi ar dura foarte mult metoda n-ar mai fi practică). Un străin care posedă însă o maşină (deocamdată fictivă) nedeterministă ar putea folosi maşina pentru a "ghici" cheia, după care ar decodifica cu această cheie şi ar verifica corectitudinea ghicirii.

De aici rezultă o consecinţă foarte interesantă: dacă se va demonstra că P=NP *nu există criptografie*! Atunci orice cifru cunoscut ar putea fi spart în timp polinomial! Dacă nu aţi crezut pînă acum că problema P=NP merită atenţie, poate acum v-aţi schimbat opinia: rezolvarea ei (în sensul confirmării egalităţii) ar supăra o grămadă de agenţii de spionaj şi militare.

Desigur, cele spuse anterior consideră că orice problemă care are o complexitate polinomială este uşoară (chiar dacă polinomul este ceva de genul x^{10000}) şi că toate problemele exponenţiale sunt grele. Este adevărat că 1.0001^x depăşeşte cu mult pe x^{10000}, dar asta numai de la o valoare foarte mare a lui x încolo.

Dar să lăsăm pentru moment teoria complexităţii. Să presupunem că $P \neq NP$, căci altfel nu mai avem ce studia, şi să aruncăm o privire la *modul* în care poate fi folosită criptografia pentru a transmite secrete. Treaba asta este cu mult mai grea decît pare.

4 Pericole

Cînd se proiectează un protocol sigur (de pildă de autentificare) trebuie să ştim exact de ce anume vrem să ne păzim. Un model răspîndit, pe care îl vom folosi şi în textul de faţă face următoarele presupuneri:

- Agenţii finali sunt siguri: terminalul pe care lucrez eu şi serverele care conţin informaţiile sunt la adăpost în mod fizic şi nu pot fi controlate de "duşmani";

- Canalul de comunicaţii între agenţii finali este complet nesigur;
- Un agent străin poate intercepta absolut toate mesajele din reţea şi le poate studia: *ascultare pasivă*; (passive wiretapping sau eavesdropping);
- Un agent străin poate injecta mesaje noi în reţea între mesajele existente (active wiretapping, tampering). Anume inamicul poate modifica mesajele existente, poate şterge unele dintre ele, poate modifica mesajele de pe reţea sau poate stoca mesaje transmise şi le poate retransmite mai tîrziu.

Este de asemenea important de ştiut că uşurinţa spargerii unui protocol creşte cu cantitatea de mesaje avute la dispoziţie; cu cît mai multe cifruri sunt observate, cu atît mai mult succes pot fi făcute anumite atacuri statistice.

5 Protocolul Kerberos

Protocolul Kerberos a fost proiectat la Universitatea MIT (Massachusetts Institute of Techonology) în cadrul proiectului Athena, în jurul anului 1984. Scopul protocolului Kerberos este de a permite unui client să-şi demonstreze identitatea unui server aflat la distanţă, undeva dincolo de o reţea complet nesigură. Protocolul garantează de asemenea că clientul nu poate conversa cu un calculator care se dă drept server; autentificarea se face în ambele direcţii.

Protocolul în sine constă dintr-un schimb de mesaje între un client şi o serie de servere, fiecare cu o altă misiune. Idea de bază aparţine lui Needham şi Schroeder care au publicat ideea iniţială în 1978 în Communications of the ACM. Descrierea detaliată a protocolului se găseşte în documentul numit RFC 1510, care este disponibil de pe Internet[1]. Cei care administrează Kerberos pot pune întrebări pe grupul de News `comp.protocols.kerberos`.

Protocolul Kerberos indică de fapt o serie de mesaje care trebuie schimbate între părţile care doresc să comunice; unele din mesaje sunt criptate. Ce funcţie de criptare/decriptare se foloseşte teoretic nu contează prea tare, atîta vreme cît funcţia este greu inversabilă. Implementările curente folosesc un algoritm standard de criptare numit DES (Data Encryption Standard).

Kerberos este un protocol; pentru a fi util anumite aplicaţii (cele care au nevoie de comunicaţie client-server) trebuie modificate pentru a folosi autentificarea oferită de Kerberos. (Aplicaţiile modificate sunt atunci numite "kerberized"). În mod normal într-un domeniu administrativ în care se foloseşte Kerberos programe ca `telnet`, `rlogin`, POP (post-office protocol), fişiere la distanţă (AFS), etc. trebuie rescrise în aşa fel încît clientul să se autentifice serverului folosind noua metodă (toate aceste aplicaţii sunt de tip client-server).

5.1 Principii

Protocolul pare destul de complicat, dar dacă aveţi răbdare o să înţelegeţi tot ce se întîmplă; nu e mare ştiinţă la mijloc.

Premiza iniţială este că există un server central, numit serverul de autentificare (AS), care cunoaşte identităţile tuturor clienţilor posibili. De asemenea, fiecare client a stabilit o parolă

[1] Într-un articol anterior din PC Report am arătat că standardele privitoare la Internet se publică în astfel de documente numite Request For Comments.

secretă pe care şi acest server o ştie. Parola ajunge la server printr-o cale sigură, de exemplu prin poştă sau printr-un mesager uman (mă rog, sigură cel puţin din punct de vedere software). Parola asta nu este cunoscută de nimeni altcineva, nici măcar de celelalte servere cu care clientul va comunica (de la care va obţine serviciile care-l interesează de fapt). Toate celelalte servere din domeniul administrativ sunt şi ele clienţi ai AS: au o parolă unică ştiută numai de AS şi de acel server.

Niciodată parola nu va circula în clar pe reţea; atunci oricine ar putea să o citească şi să o folosească în locul clientului de drept.

Pentru a se proteja împotriva duşmanilor care vor înregistra sau şterge mesaje, mesajele vor avea informaţii ca ora la care au fost trimise şi un număr de ordine (pentru a detecta omisiunile).

O altă regulă interesantă este că cheile folosite în comunicarea dintre client şi servere se schimbă frecvent. Implementarea standard expiră o cheie după 25 de ore. În felul acesta un atac criptanalitic nu va avea prea mult succes: dacă durează mai mult de 25 de ore atunci cheia descoperită este deja inutilă (desigur, eventualele mesaje deja interceptate şi stocate vor putea fi citite, dar stricăciunile sunt limitate). Cheia pe care un client şi un server o folosesc în comun se numeşte *cheie de sesiune* şi este generată aleator.

5.2 Mesajele (protocolul Needham–Schroeder)

Să presupunem că clientul nostru vrea să vorbească cu un server de disc. Protocolul esenţial este compus din 4 mesaje (protocolul real este o simplă extensie a ideii pe care o prezint aici):

1. Un mesaj de la client spre AS, prin care se indică intenţia de a comunica cu serverul de disc;
2. Un mesaj de răspuns de la AS pentru client, prin care AS îi trimite clientului noua cheie de sesiune şi un pacheţel pentru serverul de disc;
3. Un mesaj de la client spre serverul de disc, în care este inclus pacheţelul de mai sus, pentru a garanta faptul că clientul a discutat cu AS (pacheţelul poate veni numai de la AS);
4. Un mesaj de răspuns de la serverul de disc prin care clientul este convins că serverul a putut deschide pacheţelul, deci că discută într-adevăr cu serverul de disc.

După acest schimb iniţial de mesaje clientul şi serverul de disc vor folosi cheia de sesiune generată de AS pentru a cripta cu DES toată comunicaţia dintre ei.

Vom vedea că mesajele sunt relativ încîlcite pentru a preveni toate atacurile de mai sus. Dacă veţi încerca să simplificaţi protocolul aproape sigur îl veţi face vulnerabil la anumite tipuri de atacuri.

Să notăm cele 3 entităţi care colaborează astfel: C clientul, AS serverul de autentificare şi S serverul de disc (vedeţi şi figura 1).

O tehnică importantă a protocolului, care este folosită pentru a contraataca folosirea unor mesaje vechi înregistrate este de a eticheta mesajele cu ora emiterii (într-un mod care nu poate fi contrafăcut) şi de a verifica la recepţie dacă ora este rezonabilă. Sincronizarea ceasurilor este o problema extrem de grea în sistemele distribuite, aşa că pentru Kerberos "rezonabil" înseamnă că ceasul local la recepţie arată plus/minus 5 minute de ora din mesaj (ora de transmitere).

Un alt truc interesant este folosirea a ceea ce se numeşte **nonce**: un obiect care este folosit o singură dată. Acesta este practic un număr aleator. Vom vedea mai jos cum este acesta folosit.

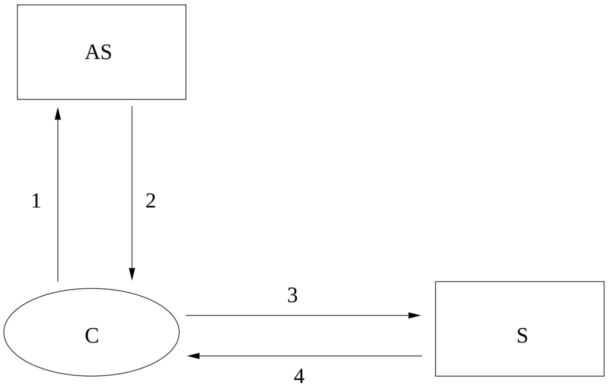

Figura 1: Protocolul de bază (simplificat).

Vom mai introduce următoarele notaţii:

- $K_{a,b}$ este cheia de sesiune pe care o folosesc pentru discuţie a şi b; de exemplu $K_{C,S}$ va fi cheia folosită pentru criptare/decriptare între client şi serverul de disc;
- Cum am spus mai sus, AS este serverul de autentificare, care ştie parola fiecărei alte entităţi din sistem;
- K_z este cheia pe care clientul z şi AS o folosesc în comun (parola clientului z); de pildă K_s este parola serverului de disc (cunoscută numai de el şi de AS);
- Voi scrie $\{\text{mesaj}_1, \text{mesaj}_2\}_K$ pentru a indica faptul că mesajele 1 şi 2 sunt puse laolaltă într-un pachet care este apoi criptat cu cheia K.
- O abreviere utilă este cea de *tichet* (ticket): $T_{a,b} = \{K_{a,b}, a, \text{ora curentă}\}$: un mesaj în care sunt împachetate 3 informaţii: o cheie de sesiune între a şi b, numele lui a şi ora curentă.

Cu aceste notaţii putem scrie complet ne-ambiguu toate mesajele schimbate între C, S şi AS. Săgeata indică traseul fiecărui mesaj:

$C \rightarrow AS$: C, S, ora expirare, N (nonce, aleator).

Clientul afirmă propria identitate, identitatea serverului cu care vrea să discute, trimite un număr aleator şi cît timp ar dori să converseze cu S. Pînă aici totul e simplu. AS va răspunde astfel:

$AS \rightarrow C$: $\{K_{C,S}, S, \text{ora expirare}, N\}_{K_C}, \{T_{C,S}\}_{K_S}$.

Acest complicat mesaj are mai două părţi mari: prima este destinată clientului, şi este criptată cu cheia clientului K_C, iar a doua este un tichet pentru S, criptat cu cheia lui S. Să vedem la ce folosesc feluritele părţi din mesaj:

- Prima parte a mesajului este criptată cu cheia K_C a lui C. Pentru că această cheie este secretă, nimeni altcineva nu poate decripta acest mesaj decît C; pentru restul lumii mesajul este gunoi. La fel stau lucrurile şi pentru partea a doua a mesajului, care fiind criptată cu K_S este inteligibilă numai pentru S.

- $K_{C,S}$ este un număr aleator generat de AS, care va fi folosit ca cheie de sesiune între C şi S. Observaţi că cheia de sesiune apare în ambele părţi ale mesajului, în aşa fel încît va fi cunoscută atît de către C cît şi de S.
- AS îi confirmă lui C identitatea serverului de disc S şi indică durata de validitate a lui $K_{C,S}$ (care poate fi diferită decît a dorit C în primul mesaj).
- Faptul că N apare în mesajul către C garantează că acest mesaj a fost criptat de AS: nimeni altcineva nu putea să-l bage pe N înauntru. De asemenea, acest mesaj nu putea fi un mesaj mai vechi dintr-o comunicaţie anterioară, pentru că N diferă.
- Partea a doua a mesajului este opacă pentru C; tot ce poate face C cu ea este să o înainteze lui S. C (sau oricine altcineva) nu o poate citi. Dacă cineva ar modifica partea a doua, S nu ar mai putea-o decodifica şi obţine un mesaj corect, deci tichetul nu poate fi contrafăcut sau modificat.

$C \to S$: $\{C, \text{ora curentă, sumă de control}\}_{K_{C,S}}, \{T_{C,S}\}_{K_S}$.

Mesajul are din nou două părţi.

- Partea a doua este exact partea a doua a mesajului 2, aşa cum a fost primită de la AS.
- Prima parte a mesajului are scopul de a demonstra că acest mesaj este "proaspăt": S va extrage $K_{C,S}$ din partea a doua a mesajului şi o va folosi pentru a citi prima parte a mesajului. De acolo află ora la care a fost trimis mesajul, şi îl rejectează dacă ora diferă prea tare de ora locală. (Asta ar putea să însemne că mesajul a fost capturat de un inamic şi re-lansat). Suma de control ne asigură că mesajul nu a fost modificat de nimeni; este practic imposibil să modifici un mesaj criptat şi să repari şi suma de control dacă nu ştii cheia.

$S \to C$: $\{\text{ora curentă}+1\}_{K_{C,S}}$.

Prin acest mesaj S în convinge pe C că a ajuns la destinaţia dorită: mesajul are ora curentă trimisă anterior de C plus 1. Dar ora putea fi extrasă numai de cel care avea $K_{C,S}$, care fiind o cheie de sesiune era ştiută numai de S. Nu era suficient să returneze aceeaşi valoare, pentru că atunci acest mesaj putea fi o copie a (unei părţi a) mesajului anterior.

La sfîrşitul acestei comunicaţii atît C cît şi S sunt siguri de identitatea celuilalt şi în plus au la dispoziţie o cheie de sesiune $K_{C,S}$ cu care pot cripta toate mesajele pe care le schimbă. Autentificarea a fost făcută.

6 Implementarea Kerberos

Între un protocol de autentificare pe hîrtie şi o implementare reală pe un calculator e o distanţă considerabilă. Secţiunea următoare, consacrată slăbiciunilor lui Kerberos va ilustra şi mai pregnant acest lucru.

Implementarea lui Kerberos încearcă să mai ţină cont de anumite particularităţi ale lumii reale care fac realizarea protocolului mai dificilă.

Prima întrebare spinoasă care se iveşte este: unde este stocată parola fiecărui client (K_C, K_S, etc.). AS trebuie să fie o maşină foarte sigură, undeva într-un subsol ferit, dar maşinile client vor fi probabil peste tot la-ndemînă. Dacă parola unui client este stocată pe disc atunci este mai la-ndemîna atacurilor asupra clientului.

Ca atare Kerberos nu memorează parola nicăieri! Utilizatorul este obligat să o tasteze de fiecare dată cînd vrea să fie autentificat. Observaţi că K_C este folosită în mesajele de mai sus numai pentru a decripta mesajul 2; în rest este inutilă. Deci procedura este următoarea: programul kerberizat al clientului va lua legătura cu AS, iar cînd răspunsul soseşte utilizatorul trebuie să tasteze parola K_C. Parola este imediat folosită pentru a decripta mesajul de la AS după care este complet ştearsă din memorie. În acest fel fereastra de vulnerabilitate este redusă la maximum.

Pe de altă parte asta poate fi foarte neplăcut, pentru că atunci utilizatorul va trebui să tasteze parola pentru fiecare nou server S pe care vrea să-l folosească (adică pentru fiecare nouă sesiune). Şi cum ştim că utilizatorii sunt leneşi, aşa ceva este inadmisibil. Kerberos a introdus atunci un al doilea server central numit "Serverul care dă tichete": Ticket Granting Server, TGS. Ideea este TGS are de fapt toate parolele K_S. Clientul C se autentifică la TGS exact în acelaşi fel ca la oricare server S: obţinînd un tichet de la AS. Odată autentificat la TGS şi folosind cheia de sesiune $K_{C,TGS}$ clientul poate solicita oricîte chei pentru alte servere S, S_1, etc.

Figura 2 şi tabela 1 arată situaţia reală: mesajele 1,2 sunt schimbate numai cînd utilizatorul face `login`. Mesajele 3,4 sunt schimbate de fiecare dată cînd utilizatorul vrea să contacteze un nou server (adică să deschidă o nouă sesiune). Mesajul 5 este folosit de client pentru a se autentifica fiecărui nou server, iar mesajul 6 este opţional, autentificînd serverul pentru client. În mesajul 5 apare un element nou, numit $K_{subsesiune}$: clientul poate alege aici o nouă cheie de sesiune care să fie folosită în locul cheii oferite de TGS, $K_{C,S}$. Asta nu schimbă prea tare natura protocolului.

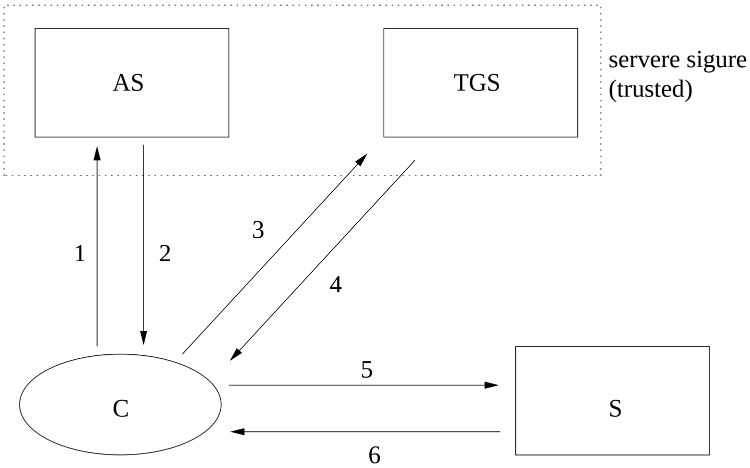

Figura 2: Protocolul Kerberos complet.

Kerberos este implementat sub forma unor procese server (AS, TGS) şi a unor biblioteci care se pot lega în programele clienţilor şi serverelor. Funcţionează sub o mare varietate de sisteme de operare: Unix şi Windows NT fiind cele mai notabile. Mai are tot felul de zorzoane, legate de pildă de autentificarea între domenii administrative diferite, transmiterea tichetelor, etc.

Nr.	Între	Conţinutul mesajului
1	C $\rightarrow$ AS	C, TGS, ora de expirare, N
2	AS $\rightarrow$ C	$\{K_{C,TGS}$, ora de expirare, $N\}_{K_C}, \{T_{C,TGS}\}_{K_{TGS}}$
h 3	C $\rightarrow$ TGS	$\{$ora locală$\}_{K_{C,TGS}}, \{T_{C,TGS}\}_{K_{TGS}}, S$, ora de expirare, N_1
4	TGS $\rightarrow$ C	$\{K_{C,S}, S$, ora de expirare, $N_1\}_{K_{C,TGS}}, \{T_{C,S}\}_{K_S}$
5	C $\rightarrow$ S	$\{$ora locală, sumă de control, $K_{subsesiune}\}_{K_{C,S}}, \{T_{C,S}\}_{K_S}$
6	S $\rightarrow$ C	$\{$ora locală$\}_{K_{C,S}}$

Tabelul 1: Schimbul complet de mesaje în protocolul Kerberos.

7 Slăbiciunile lui Kerberos

Chiar dacă în teorie Kerberos este minunat, implementarea lui în practică este cel puţin dificilă. Condiţiile ideale existente pe hîrtie sunt greu de obţinut într-o reţea de calculatoare reale.

La ora actuală nu există nici un fel de metodă complet riguroasă pentru a arăta ca un protocol criptografic nu scapă informaţii; există metode pentru a testa dacă un protocol rezistă la atacurile cunoscute, dar foarte adesea se publică algoritmi care mai tîrziu se dovedesc greşiţi. În general, raţionamentele cu astfel de protocoale sunt foarte complicate. Cercetarea în domeniu este în plină desfăşurare şi foloseşte tehnici foarte exotice, ca teoria informaţiei, teoria complexităţii, logici speciale (ex. knowledge theory), etc.

Voi ilustra aici numai unele dintre deficienţe, pentru a da o idee despre natura lor.

Să observăm că clientul trebuie să păstreze undeva cheile de sesiune pentru a putea conversa cu serverele: fiecare mesaj după cele de autentificare va fi criptat cu aceste chei. Clientul trebuie să posede deci practic permanent $K_{C,TGS}$ şi $K_{C,S}$. E adevărat că aceste chei expiră în 25 de ore, deci sunt mai puţin importante decît o parolă care teoretic este folosită luni întregi. Întrebarea este însă: unde sunt ţinute pe calculatorul clientului aceste chei?

Pe o staţie obişnuită Unix lucrează în mod normal mai mulţi utilizatori. Tichetele unuia ar trebui să fie ferite de ceilalţi. Dar pe un sistem Unix practic nimic nu poate fi adăpostit împotriva administratorului (root). Administratorul unui sistem poate citi orice fişier, şi poate inspecta memoria fizică a oricărui proces. Acesta este un călcîi al lui Ahile al lui Kerberos; toate metodele cunoscute pentru a penetra un sistem Unix ameninţă siguranţa întregului protocol. Ori securitatea unui sistem Unix, care este foarte complicat, este extrem de greu de controlat; există o sumedenie de breşe de care un atacator ar putea profita.

O altă mare problemă este cu staţiile de lucru fără disc (diskless); aceste staţii de obicei importă discuri prin reţea. Deci de îndată ce o astfel de staţie stochează un tichet pe disc, tichetul va călători prin reţea, care am stabilit că este expusă la tot felul de atacuri!

Nici păstrarea tichetului în memorie nu este neapărat mai sigură: algoritmii de paginare stochează paginile pe disc (în partiţia de swap) atunci cînd calculatorul nu are destulă memorie, deci am revenit la aceeaşi problemă.

Iată încă un exemplu: am văzut că prospeţimea unui tichet este verificată comparînd ora locală a serverului cu ora din tichet. Pentru un interval de 5 minute serverul memorează toate tichetele primite, pentru a depista duplicate, eventual rezultate dintr-un atac care re-transmite pachete vechi capturate. Un tichet mai vechi de 5 minute este considerat expirat şi ignorat. În felul acesta un server nu va primi niciodată acelaşi pachet de două ori. Asta presupune că serverul şi clientul au ceasuri relativ sincronizate.

Într-o reţea mare de calculatoare sincronizarea ceasurilor se face automat, folosind un protocol numit NTP: Network Time Protocol. Un atac foarte spectaculos este următorul: un atacator înregistrează o serie de mesaje de la un client care ştie că reprezintă o tranzacţie importantă. Peste o săptămînă atacatorul infiltrează în reţea mesaje false NTP prin care setează ceasul unui server cu o săptămînă în urmă. După asta atacatorul retransmite mesajele capturate, care vor fi re-executate, pentru că serverului îi par proaspete.

Asta face securitatea în calculatoare o problemă foarte spinoasă: adesea protocoalele propuse sunt eronate, dar nu există nici o metodă formală pentru a depista şi verifica asta. Chiar dacă un protocol este corect formal, se poate baza pe asumpţii nerezonabile asupra mediului în care operează, cum ar fi ceasurile sincronizate. Şi chiar dacă se bazează pe asumpţii rezonabile, implementarea scrisă de un programator uman poate să aibă bug-uri care o fac vulnerabilă.

8 Kerberos pentru un utilizator

Voi încheia acest articol ilustrînd cum se manifestă Kerberos pentru un utilizator obişnuit. Domeniul administrativ în care lucrez eu de obicei este complet kerberizat. Programul meu `login` a fost modificat ca imediat ce tastez parola să discute cu AS şi să obţină un tichet pentru serverul care dă tichete, TGS. Directorul meu casă este de pe un disc din reţea, aflat undeva departe, pe un server de disc (protocolul folosit este AFS: Andrew File System). Atunci cînd comunic prima oară cu serverul de AFS demonul local de pe maşina mea foloseşte tichetul obţinut de la AS pentru a obţine de la TGS un tichet pentru serverul de disc. După aceea se autentifică serverului de disc, exact ca în schema de mai sus.

După autentificarea cu serverul de disc toată comunicaţia între maşina mea şi serverul de disc se va face folosind un alt protocol, numit Secure RPC (Remote Procedure Call): apel sigur de procedură la distanţă, care foloseşte iniţial cheia de sesiune oferită de TGS.

Pentru mine ca utilizator final totul este aproape complet transparent. Singura neplăcere este că la fiecare 25 de ore tichetele pentru TGS expiră, şi atunci trebuie să-mi tastez din nou parola pentru a obţine tichete proaspete. Asta poate fi neplăcut dacă vrei să rulezi o simulare mai îndelungată.

Pot să văd în orice clipă tichetele pe care le posed cu comanda `klist`:

```
$ klist
Ticket file: /tkt/7108-0401-35ae552f
Principal: mihaib@CS.CMU.EDU

  Issued           Expires           Principal
Jul 20 10:50:34  Jul 21 12:16:55  krbtgt.CS.CMU.EDU@CS.CMU.EDU
Jul 20 10:50:34  Jul 21 12:16:55  afs@CS.CMU.EDU
Jul 20 10:51:30  Jul 21 12:17:51  zephyr.zephyr@CS.CMU.EDU
```

Tichetele sunt ţinute în fişierul `/tkt/7108-0401-35ae552f` pe discul local. Am în clipa asta 3 tichete: unul pentru TGS, unul pentru serverul de disc AFS şi unul pentru sistemul de mesagerie zephyr (care nu ne interesează prea tare acum).

Mai am la dispoziţie următoarele comenzi:

kinit prin care pot să-mi schimb identitatea Kerberos (de pildă dacă un coleg vrea să lucreze pe calculatorul meu o să tasteze `kinit numele-lui` şi apoi parola lui personală;

kdestroy prin care toate tichetele de pe maşina locală sunt distruse; foarte util dacă plec şi nu vreau ca cineva să poată lucra în numele meu;

kpasswd prin care îmi pot schimba cheia (parola) stocată pe AS. Schimbarea parolei este sigură, pentru că parola va fi trimisă criptat la un server special care face managementul parolelor şi modifică baza de date din care citeşte AS. Autentificarea la managerul de parole se face tot folosind Kerberos. Asta înseamnă că odată ce am o parolă Kerberos (care trebuie introdusă manual în baza de date a lui AS) schimbarea o pot face fără să mai bat vreun administrator la cap;

kdb_init este o comandă folosită de administrator pentru a crea o nouă bază de date Kerberos atunci cînd porneşte serviciul;

kdb_admin este comanda prin care administratorul adaugă un nou utilizator în baza de date;

kdb_edit este comanda prin care se pot adăuga noi administratori în baza de date AS: persoane care au dreptul să modifice baza de date.

În plus, în domeniul în care lucrez eu, majoritatea comenzilor care operează în reţea au fost kerberizate. De pildă demonul şi clientul de telnet (terminal virtual): cînd eu fac `telnet` pe o maşină la distanţă clientul îmi cere parola după care obţine un tichet de la TGS pentru demonul `telnetd` de pe maşina de la distanţă; în acest fel parola mea nu circulă niciodată prin reţea (cum ar fi fost cazul dacă telnet nu era kerberizat).

9 Concluzie

Securitatea în calculatoare este o problemă de mare importanţă economică, mai ales acum cînd tot mai multe tranzacţii se fac prin Internet. Proiectarea unui protocol de securitate este o treabă foarte complicată, iar implementarea nu este deloc simplă. Domeniul este în plină cercetare în continuare. Dificultăţile sunt amplificate de faptul că lanţul este tot atît de slab cît cea mai slabă verigă, iar verigi sunt destul de multe. Fiţi deci cu ochii-n patru.