Universitatea „Politehnica” Timişoara
Facultatea de Automatică şi Calculatoare

Studiu asupra nivelului de siguranță oferit de protocoalele de autentificare

Referat nr. 2

Doctorand:
ing. Valer BOCAN

Conducător ştiințific:
prof. dr. ing. Vladimir CREȚU

2004

Ultima versiune a acestui document se poate obține de la adresa *http://www.dataman.ro*
sau scriind autorului la adresa de e-mail *vbocan@dataman.ro*

Cuprins

Capitolul 1

Introducere

Autentificarea este definită ca procesul de verificare sigură a identității cuiva sau a ceva. Există o mulțime de exemple de autentificare în interacțiunea umană: persoanele cunoscute le recunoaştem după figură şi voce, un gardian ne poate identifica prin compararea fizionomiei cu poza de pe cartea de identitate sau legitimație, iar casa fiecăruia dintre noi ne „permite" accesul înăuntru dacă suntem în posesia setului de chei corespunzător.

1.1. Metode de autentificare

În cadrul autentificării deosebim două cazuri: (a) autentificarea a două sisteme de calcul şi (b) autentificarea unui utilizator către un sistem de calcul. Problema în cel de-al doilea caz este că utilizatorul trebuie să memoreze o cantitate secretă (parolă), de cele mai multe ori în format text, acesta fiind în măsură să o aleagă. Utilizatorii de cele mai multe ori vor alege parole slabe, vulnerabile la „ghicire".

De-a lungul timpului s-au imaginat diferite metode prin care identitatea unui utilizator să fie confirmată unui sistem. **Autentificarea bazată pe parole** este cea mai simplă dintre metode şi presupune transmiterea în clar a parolei de acces, acest fel de autentificare apărând ca o necesitate a terminalelor fără putere de calcul proprie. Aproximativ în aceeaşi perioadă, unele sisteme au trecut la **autentificarea bazată pe adresă**, o metodă care a stârnit controverse cu privire la avantajele sale. Odată cu evoluția sistemelor şi creşterea puterii de calcul s-a trecut la **autentificarea criptografică**. În această categorie intră protocoale cum ar fi Kerberos, SESAME, SSL, TLS, SPX şi multe altele.

1.1.1. Autentificarea bazată pe parole

Cu toții ştim ce este o parolă, dar ne este foarte greu să o definim formal. Când vorbim de *autentificare bazată pe parole*, ne gândim la o cantitate secretă (parola) pe care pretindem că o ştim. Problema acestui tip de autentificare este faptul că informația circulă în clar pe căile de comunicație şi poate fi capturată şi analizată de un atacator.

Anumite sisteme de autentificare folosesc o parolă introdusă de utilizator pe post de cheie criptografică sau o folosesc pentru generarea unor valori folosite ulterior în operații criptografice.

Deşi la bază au o parolă introdusă de utilizator, vom considera aceste sisteme ca fiind criptografice mai degrabă decât bazate pe parolă deoarece nu au neajunsul major al traficului necriptat.

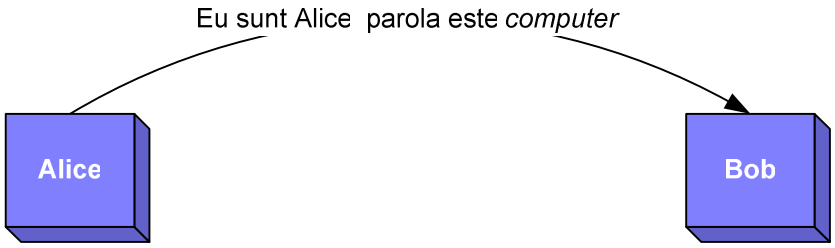


Fig. 1 Autentificarea prin parolă

Întrebarea naturală care se poate pune acum este de ce se mai foloseşte autentificarea bazată pe parole când autentificarea criptografică este mult mai sigură? Anumite categorii de persoane lucrează la aşa numitele *dumb terminals*, dispozitive care nu au putere de calcul proprie, deci nu pot efectua calcule criptografice de nici un fel. Este deci dificil să se evite autentificarea prin parolă, cu toate că în ultima vreme au apărut dispozitive inteligente[1] menite să suplinească această lipsă. Din nefericire, uneori chiar autentificarea computer – computer se bazează pe parole. Uneori protocolul a pornit ca o autentificare simplă om – maşină şi nu a evoluat corespunzător când locul omului a fost luat de altă maşină, alteori proiectanții au considerat că folosirea unui sistem criptografic ar fi mult prea scumpă din punct de vedere al timpului de execuție. Alteori sisteme criptografice sunt evitate din cauza problemelor legale.

În unele cazuri, folosirea autentificării prin parole este chiar deranjantă şi este greu de înțeles de ce proiectanții au ales-o. În Statele Unite ale Americii, la începutul telefoniei mobile, transmiterea numărului abonatului şi parola de acces în rețea se făcea în clar. Problema este că oricine putea să monitorizeze traficul şi să colecteze perechi ⟨Număr telefon; Parola⟩ pentru ca apoi să cloneze telefoane şi să efectueze convorbiri în contul abonatului clonat. Noile generații de telefoane mobile nu mai suferă astăzi de acest neajuns [KAUF02].

Autentificarea bazată pe parolă are şi alte dezavantaje. Utilizatorul unei stații de lucru poate accesa o mulțime de resurse din rețea. Acesta trebuie să se identifice în mod individual fiecărei resurse, adică să tasteze numele şi parola de fiecare dată. Acest lucru devine repede un motiv de stres pentru utilizator, mai ales că fiecare resursă ar putea avea propria parolă. O soluție convenabilă ar fi stabilirea aceleiaşi parole pentru toate resursele, dar în acest caz efortul schimbării parolelor devine disproporționat de mare în raport cu beneficiile.

1.1.2. Autentificarea bazată pe adresă

Autentificarea bazată pe adresă nu se bazează pe trimiterea parolelor prin rețea ci presupune că identitatea sursei se poate deduce din analiza adresei de rețea de la care provin pachetele. Ideea a fost adoptată de UNIX şi VMS la începuturile rețelelor de calculatoare. Ideea de bază este că fiecare computer memorează informații despre

[1] Dispozitivele de acest fel sunt chei inteligente, carduri de memorie, carduri criptografice, smart-card-uri.

conturile altor sisteme care au acces la resursele sale. Spre exemplu, presupunem că un cont numit Smith definit pe o maşină la adresa N are permisiunea de a accesa resursele sistemului C. Cererile pentru acces la resurse sunt comenzi precum *copiază fişier*, *log in*, *execută comanda*, etc. Dacă cererea soseşte de la adresa N şi pretinde că este trimisă în numele utilizatorului Smith, atunci sistemul C va onora comanda.

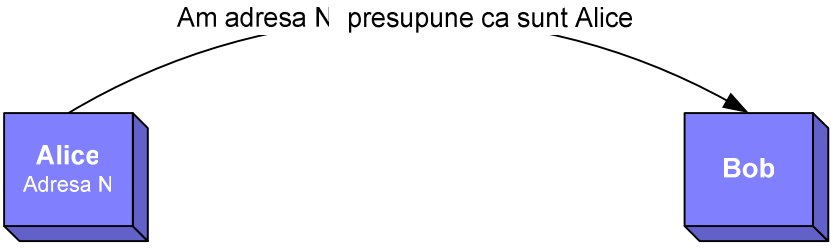


Fig. 2 Autentificarea bazată pe adresă

Deşi nu este susceptibilă la atacuri prin monitorizare, autentificarea bazată pe adresă este ameninţată în alte două feluri:

1. Un utilizator neautorizat care are acces la o staţie de lucru C, nimic nu îl va împiedica să aibă acces la toate resursele reţelei care acceptă conexiuni de la C.
2. Impersonarea adresei sursă este o tehnică uşor de implementat şi exploatat. În acest caz, resursele reţelei sunt înşelate cu privire la identitatea sursei.

Aşadar, securitatea autentificării bazate pe adresă este mai mare sau mai mică decât autentificarea prin parole, în funcţie de situaţia specifică, şi este în mod clar mult mai convenabilă din punctul de vedere al utilizatorului, fiind mecanismul standard în multe sisteme distribuite de astăzi [KAUF02].

1.1.3. Autentificarea criptografică

Ideea din spatele autentificării criptografice este că Alice îşi dovedeşte identitatea către Bob prin efectuarea unei operaţii criptografice asupra unei entităţi cunoscute de ambii participanţi sau oferită de Bob. Operaţia criptografică efectuată de Alice se bazează pe o cheie criptografică. Aceasta poate fi fie o cheie secretă sau o cheie privată dintr-un sistem cu chei asimetrice.

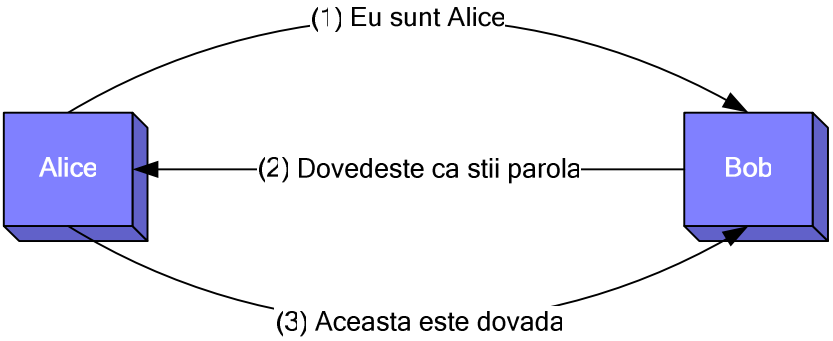


Fig. 3 Autentificarea criptografică

În general, autentificarea criptografică este mai sigură decât autentificarea bazată pe parolă sau pe adresă. În schimb, noile tehnici bazate pe dovezi *zero-knowledge* pot oferi mecanisme de autentificare chiar mai puternice. Aceste tehnici necesită calcule matematice destul de complexe dar prezintă mai multe facilităţi atractive pentru autentificare. În primul rând, permit părţii ce se autentifică să dovedească că ştie secretul

fără a transfera efectiv informația către verificator. În al doilea rând, multe dintre schemele propuse până acum folosesc aceleaşi informații publice, evitându-se astfel problema distribuției cheilor care apare în cazul mecanismelor ce folosesc DES şi RSA.

În ciuda aparentei simplități, proiectarea sistemelor reale este foarte dificilă. O serie de protocoale publicate au prezentat erori de securitate substanțiale sau subtile. În timpul ultimei decade, eforturile de cercetare s-au concentrat în crearea de utilitare pentru dezvoltarea protocoalelor de autentificare şi distribuție a cheilor cu o anumită asigurare formală a securității, realizările cele mai notabile fiind logica BAN şi logica GNY [BOCA01].

1.2. Autentificarea utilizatorilor

Autentificarea se face diferențiat, în funcție de subiectul autentificării. Cele două capacități importante sunt memorarea unei chei criptografice de înaltă calitate şi efectuarea de operații criptografice. În accepțiunea curentă, o cheie de înaltă calitate este o cantitate secretă aleasă dintr-un spațiu foarte mare în aşa fel încât o căutare exhaustivă să fie nefezabilă din punct de vedere computațional. Un computer are ambele capacități, pe când un utilizator nu are nici una dintre ele.

Autentificarea utilizatorilor constă în verificarea de către un computer că identitatea este cea declarată. Cele trei tehnici principale sunt:

- ceea ce utilizatorul ştie
- ceea ce utilizatorul are
- ceea ce utilizatorul este

Parolele sunt o metodă de autentificare ce se încadrează perfect în prima tehnică. Banalele chei se încadrează în cea două tehnică, iar cardurile bancare sunt o combinație a primelor două tehnici. Dispozitivele biometrice, cum ar fi analizoarele vocale şi de amprente se încadrează în a treia tehnică enunțată.

1.2.1. Parole

Parolele au apărut cu mult înaintea computerelor. Conceptul este simplu: dacă Alice trebuie să-i dovedească lui Bob că ea este într-adevăr Alice, iar Bob nu o cunoaşte personal pe Alice, cei doi pot stabili un salut special (o parolă) iar Bob presupune că o persoană care rosteşte acest salut este chiar Alice. Ca multe alte aspecte ale securității, primele utilizări ale parolelor au fost în armată. Membrilor unui grup li se comunică o parolă a zilei iar aceştia trebuie să o comunice la poartă, după lăsarea întunericului.

Cu siguranță că cei mai mulți dintre cititori au intrat în diferite sisteme prin introducerea numelui şi a parolei atât de des încât această acțiune nici nu mai prezintă importanță. Autentificarea prin parolă are însă o mulțime de probleme:

- Cineva poate observa parola când utilizatorul legitim o tastează.
- Un atacator poate citi fişierul de parole de pe sistemul vizat.
- Parolele pot fi ghicite şi încercate direct la consola sistemului vizat.
- Fişierul cu parole poate fi „spart” prin forță brută, dacă fişierul de parole conține o cantitate ce se poate recunoaşte ca validă.
- Încercarea de a impune folosirea de parole greu de ghicit duce la inconveniente legate de confortul utilizatorilor şi la riscul ca aceştia să scrie parolele pe diferite suporturi.

Eforturile trebuie să se concentreze în direcția limitării expunerii parolelor la vedere, îngreunarea ghicirii acestora şi limitarea numărului de ghiciri incorecte.

1.2.2. Ghicirea „on-line” a parolelor

Utilizatorul A îl poate impersona pe utilizatorul B dacă îi cunoaşte parola. Acest lucru nici nu este foarte dificil, mai ales că există anumite persoane care folosesc drept parolă numele sau prenumele lor [KAUF02]. Pe de altă parte, unele sisteme au ca măsură administrativă fixarea unei ca parolă a atributului unei persoane, cum ar fi ziua de naştere sau numărul de legitimaţie. Această metodă uşurează memorarea parolelor şi distribuţia acestora, dar creşte foarte mult riscul de ghicire a parolelor.

Chiar dacă parolele nu se aleg uşor de ghicit, un impostor poate încerca toate combinaţiile posibile până când parola este acceptată de sistem. De fapt, dacă avem în vedere un număr suficient de mare de încercări, orice parolă poate fi ghicită, într-un timp funcţie de viteza sistemului. Prima măsură care se poate aplica este împiedicarea accesului la un cont după un număr prestabilit de încercări. Acest lucru se întâmplă deja în cazul bancomatelor, acestea refuzând să elibereze cardul dacă codul PIN se introduce greşit de trei ori. O lipsă importantă al acestui fel de contracarare e ghicirii parolelor este că un vandal ar putea să încerce un anumit număr de parole pe fiecare sistem la care are acces, provocându-i astfel blocarea până când administratorul sistemului intervine. Puţin efort din partea vandalului poate provoca daune importante, lucru ce nu este de dorit în nici o situaţie.

Un altfel de abordare a problemei este încetinirea atacatorului suficient de mult ca timpul necesar ghicirii parolelor să crească foarte mult. Înaintea validării sau invalidării rezultatului se poate introduce o pauză artificială care poate trece neobservată de către utilizatorul obişnuit dar pentru atacator înseamnă o scădere considerabilă de viteză.

Nu în ultimul rând, metodele prezentate anterior trebuie coroborate cu o atentă monitorizare a jurnalelor sistemelor, unde încercările nereuşite de log-in sunt semnalate. Pentru evitarea alarmelor false (când un utilizator legitim greşeşte parola o dată sau chiar de două ori) se pot folosi filtre inteligente, care să indice administratorului o posibilă activitate suspectă. În mod uzual, sistemele indică utilizatorului data ultimului acces şi eventual adresa IP de la care s-a făcut accesul. Dacă de cele mai multe ori aceste informaţii trec neobservate de către utilizator, numărul de încercări nereuşite de log-in de la ultimul acces cu siguranţă ar fi interesant şi ar ridica imediat semne de întrebare utilizatorului.

O parolă bună în general are cel puţin 8 caractere şi conţine cel puţin un caracter special, în afară de litere şi cifre. Spaţiul cheilor în acest caz este foarte mare, o căutare exhaustivă la viteza de o mie de parole pe secundă putând să dureze în jur de 3 ani. Deşi ar putea să pară încurajator, memorarea unei astfel de parole bune este destul de dificilă pentru un utilizator, mai ales că aceasta trebuie schimbată des. Compromisul în acest caz îl reprezintă generatoarele de parole pronunţabile, care au o lungime minimă de 10 caractere, dar dat fiind faptul că setul de caractere este mai restrâns, oferă o securitate similară cu parolele bune de 8 caractere. Pentru a evita situaţiile în care se generează cuvinte existente (care sunt susceptibile la ghicire), trebuie ca parolele generate să fie validate prin căutarea în dicţionare mari. Un dicţionar bun trebuie să cuprindă în jur de 500.000 de cuvinte şi trebuie extins cu nume şi prenume uzuale.

1.2.3. Ghicirea „off-line” a parolelor

Până acum ne-am ocupat de ghicirea „on-line” a parolelor. În acest caz, o monitorizare atentă şi o procesare lentă a autentificării pot ameliora efectele atacurilor. În cazul „off-line” însă, lucrurile stau într-un mod diferit, întrucât atacatorul poate obţine prin diverse mijloace (citire directă, captură, etc.) o copie a fişierului de parole pe care îl poate analiza în intimitatea propriei case.

Parolele trebuie memorate în două locuri, adică la cele două capete ale autentificării, şi anume la utilizator şi la sistemul de calcul. Dacă în primul caz parola este „sigură", ea fiind memorată de către utilizatorul uman, serverul trebuie să memoreze cumva parola. Dacă parolele se memorează în clar, atunci fişierul în cauză trebuie protejat cel puțin la fel de bine ca şi restul sistemului. Problemele apar când sistemului i se fac copii de siguranță care trebuie protejate la rândul lor la fel de bine, lucru care devine greu de realizat. De aceea, parolele nu se vor scrie în clar pe disc, în schimb li se vor aplica o funcție de dispersie şi rezultatul acesteia se scrie în fişier. După cum se ştie, inversarea unei funcții hash este imposibilă din punct de vedere al timpului de execuție, deci metoda pare infailibilă[1]. În trecut, unele sisteme Unix sau compatibile erau atât de sigure că parolele sunt în siguranță încât fişierul cu hash-uri era public şi putea fi citit de toată lumea. Mai târziu s-a dovedit că neprotejarea la citire a fişierului de hash-uri a fost o idee proastă, întrucât cunoscându-se funcția de dispersie se pot analiza cuvintele dintr-un dicționar mare precum şi alte variante şi combinații de cuvinte care pot duce la găsirea parolelor slabe. Un avantaj al acestei tehnici este că parolele dintr-un sistem nu pot fi văzute de administratorul sistemului. În cazul în care utilizatorul reclamă pierderea parolei, noua parolă poate fi stabilită fără ca administratorul să o cunoască.

1.2.4. Lungimea parolelor

Întrebarea naturală care se pune este cat de lungă trebuie să fie o parolă pentru ca aceasta să fie sigură? Pentru a bloca un atac on-line, secretul nu trebuie ales dintr-un spațiu foarte mare, deoarece intrusul este detectat după un număr mic de încercări. Spre exemplu, codul PIN al unui card bancar are doar 4 cifre, ceea ce duce la un spațiu de 10.000 de coduri diferite. Având în vedere că bancomatele acceptă doar 3 introduceri greşite ale codului PIN, lungimea codului PIN este suficientă.

Dacă există posibilitatea unui atac off-line, parola trebuie aleasă dintr-un spațiu mult mai mare. Regula generală este ca parola să aibă în jur de 64 de biți aleatori, deoarece se consideră nefezabilă căutarea a 2^{64} posibilități. Din nefericire, oamenii nu sunt capabili să rețină numere aleatoare de 64 de biți, adică în jur de 20 de cifre. Dacă parola conține litere mari şi mici, cele 10 cifre şi câteva semne de punctuație, şi având în vedere că sunt 64 de posibilități per caracter (6 biți), atunci este nevoie de o parolă de aproximativ 11 caractere. Din nou, o parolă aleatoare de 11 caractere nu se poate memora uşor de către un utilizator.

Ideea de a genera parole pronunțabile introdusă în paragraful anterior introduce o nouă constrângere, aceea că fiecare al treilea caracter să fie o vocală. Pentru a fi memorabilă, parola trebuie să fie un şir „case-insensitive", ceea ce dă aproximativ 4½ biți per caracter şi doar 2½ biți per vocală (din moment ce sunt 5 vocale în limba engleză). Toate aceste constrângeri necesită aproximativ 4 biți aleatori per caracter, adică o parolă de 16 caractere, din nou destul de mult pentru un utilizator obişnuit.

[1] [KAUF02] menționează un sistem de operare al cărui nume nu îl dezvăluie al cărui fişier de parole are o proprietate interesantă: Dacă se aplică funcția de dispersie se aplică unui şir S, rezultatul este acelaşi cu aplicarea aceleiaşi funcții de dispersie la o anume cantitate „magică" X, în combinație cu şirul S. Proprietatea este utilă atunci când administratorul sistemului stabileşte o politică de lungime minimă a parolelor iar utilizatorul insistă să folosească parole mai scurte. Să presupunem că utilizatorul doreşte să-şi stabilească parola FOO, care este mai scurtă decât lungimea impusă de administrator. Dacă secvența magică este %#v24dR678riwe88ds/, parola care trebuie introdusă este %#v24dR678riwe88ds/FOO, aceasta fiind singura dată când trebuie introdusă această secvență complicată. De acum înainte, parola acceptată de sistem este FOO, chiar dacă administratorul a impus parole mai lungi.

În cazul în care utilizatorul îşi poate alege singur parola, nivelul de aleatorie este de aproximativ 2 biţi per caracter, ceea ce duce la o parolă de 32 de caractere, care poate fi greu de memorat.

Concluzia este că utilizatorii sunt incapabili să memoreze şi să folosească corect parole lungi care ar avea un grad satisfăcător de aleatorie, adică 64 de biţi. Drept urmare, parolele au fost şi vor fi ţinte uşoare pentru atacurile off-line.

1.2.5. Capturarea parolelor

O altă slăbiciune importantă a parolelor este că pentru a fi folosite, acestea trebuie „exprimate" în clar prin tastarea ei. Cea mai simplă metodă de captură este observarea ei în timpul tastării. Spre deosebire de birouri, există o eticheta nescrisă cu privire la operarea bancomatelor. A doua persoană în rând menţine o distanţă care să-l împiedice a vedea codul PIN tastat de persoana care operează bancomatul. Eticheta nu este aşa de obişnuită în birourile de astăzi, aşa încât parola se poate observa uşor atunci când este tastată. Singura măsură de prevenire este instruirea utilizatorilor să fie grijulii şi să folosească parole care să includă taste speciale, cum ar fi ALT şi CTRL, aşa încât să se folosească mai multe degete în tastarea lor şi parola să fie greu de urmărit.

O metodă mai evoluată de captură a parolelor este urmărirea liniilor de comunicaţie. Dacă acest lucru este simplu sau dificil depinde de mediul concret. Spre exemplu, o reţea locală în configuraţie stea şi deservită de un dispozitiv de nivelul 2 (un hub) rutează traficul către toate sistemele conectate, în acest fel urmărirea fiind posibilă. Dacă reţeaua este deservită de un dispozitiv de nivelul 3 (un switch sau router), interceptarea devine mai dificilă datorită modului de rutare al informaţiei. Nu în ultimul rând, trebuie avute în vedere programele de tip „key logger" sau dispozitivele hardware înglobate în tastaturi pentru acest scop.

Dacă suntem gata să trecem cu vederea riscul indus de parolele scrise pe hârtie, se poate folosi metoda parolelor de unică folosinţă. Odată folosită, o parolă este bifată şi nu va mai fi folosită în viitor. Deşi această metodă este rezistentă la captură prin mijloacele obişnuite, modul de folosire introduce noi probleme administrative.

1.2.6. Parole şi utilizatori neglijenţi

La un curs despre securitatea sistemelor, un profesor a întrebat care sunt avantajele parolelor faţă de sistemele biometrice de autentificare. Un student a răspuns că dacă vrei ca cineva să îţi folosească contul trebuie să mergi cu el până intră în sistem. Acest tip de răspuns face administratorii de sisteme să se gândească la utilizatori ca la inamicii lor mai degrabă decât clienţii pe care trebuie să îi protejeze. De cele mai multe ori utilizatorii vor renunţa la securitate în favoarea uşurinţei în exploatare, iar acest lucru se poate ameliora doar printr-o educaţie atentă.

1.2.7. Distribuţia iniţială a parolelor

Tot ce am discutat până acum se referă la sisteme care presupun că parolele sunt deja cunoscute de ambele părţi. Să vedem acum cum se pot distribui iniţial parolele într-o manieră sigură.

Una dintre metodele des uzitate este ca utilizatorul să se deplaseze la biroul administratorului şi să prezinte un act de identitate cu fotografie. Administratorul procedează la stabilirea informaţiilor particulare despre sistem, cu excepţia parolei, care va fi introdusă de către însuşi utilizatorul. Această metodă însă are două mai neajunsuri: în primul rând deplasarea la biroul administratorului poate fi dificilă, mai ales când acest lucru presupune transport pe distanţe mari, iar în al doilea rând este destul de periculos ca

un utilizator să petreacă fie şi numai câteva secunde la tastatura unui sistem considerat privilegiat.

O variantă a acestui procedeu este stabilirea unei parole inițiale tari care va fi scrisă pe un suport de hârtie sau electronic, iar utilizatorul va fi obligat să o schimbe odată cu prima autentificare. Riscul de securitate este minim, întrucât parola scrisă este valabilă doar o scurtă perioadă de timp.

O practică comună dar cu probleme de securitate destul de importante este stabilirea inițială a parolei la o valoare cunoscută, ca de exemplu o proprietate a utilizatorului. Unele şcoli setează parola tuturor studenților ca numărul matricol al fiecăruia, apoi anunțul este publicat la gazeta de perete, urmând ca fiecare student să-şi modifice parola la prima autentificare. O eventuală pătrundere neautorizată în aceste conturi nu va produce pagube importante deoarece conturile păstrează munca studenților care se consideră a fi la început. Totuşi, un atacator ar avea timp suficient să plaseze un cal troian sau un program de monitorizare a tastaturii.

1.2.8. Dispozitive de autentificare

Dispozitivul de autentificare este o entitate fizică a cărui posesie trebuie probată de către utilizatorul ce dorește a se autentifica. Acesta are avantaje şi dezavantaje față de alte sisteme de autentificare. În afara cazului în care dispozitivul este ataşat utilizatorului (ceea ce în cultura noastră este de neacceptat), dispozitivul poate fi subtilizat, de aceea trebuie să fie asociat cu o parolă sau un cod.

Astăzi există mai multe dispozitive de autentificare. Cel mai răspândit este cheia pe care o folosim zi de zi. Un alt dispozitiv popular este cardul de credit. Acesta conține o bandă magnetică pe care sunt imprimate informații. Avantajul major este că nu poate fi copiat uşor, iar informațiile conținute sunt mai multe decât poate un utilizator obişnuit să memoreze. Cel mai important câştig este probabil ce psihologic, deoarece oamenii sunt mai puțin tentați să „împrumute" un card de credit decât parola proprie. Dezavantajele sunt:

- Utilizarea dispozitivelor de autentificare necesită hardware specializat la fiecare punct de lucru. Aceasta presupune costuri suplimentare şi standardizare.
- Dispozitivele pot fi pierdute sau furate. Pentru securitate, dispozitivele trebuie însoțite de coduri PIN sau parole.

Utilizarea cardurilor presupune aceleaşi riscuri la captura informațiilor ca şi parolele.

O formă mai evoluată de autentificare o reprezintă cardul inteligent (smart card). Acest dispozitiv are forma şi dimensiunea unui card de credit, dar are un procesor şi memorie proprie. Când este introdus într-un aşa numit cititor, cardul poartă o conversație cu sistemul spre deosebire de un card normal care doar îşi raportează conținutul.

1.2.9. Dispozitive biometrice

Dispozitive biometrice măsoară o caracteristică a utilizatorului şi o compară cu un profil stocat anterior. Mărimile biometrice nu pot fi împrumutate sau subtilizate şi se pot falsifica destul de dificil.

În prezent există o multitudine de dispozitive biometrice, dar ele sunt mult prea scumpe pentru a fi utile în aplicații de zi cu zi. Dispozitivele includ:

- **Scanner de retină:** Aceste dispozitive examinează minusculele vase de sânge de pe fundul ochiului. Dispunerea acestora este proprie fiecărui individ în parte, aşa cum sunt amprentele digitale. Aceste dispozitive sunt

destul de scumpe şi au o interfață cu utilizatorul nu dintre cele mai prietenoase.

- **Cititoare de amprentă:** Amprentele s-au folosit pentru identificare de mulți ani şi folosirea lor ar putea părea naturală. Din diverse motive, acest fel de autentificare nu a „prins" deşi dispozitivele sunt disponibile de ceva vreme.
- **Recunoaştere facială:** O poză digitală a feței unei persoane poate fi recunoscută de un computer prin măsurarea caracteristicilor faciale. Metoda are neajunsul că micile modificări ale fizionomiei datorate unor afecțiuni cum ar fi tumefacția şi hematomul orbital pot duce la autentificări eronate.
- **Scanner de iris:** Este similar scanner-ului de retină, dar prezintă o interfață cu utilizatorul mai prietenoasă. În loc să se uite printr-un dispozitiv laser, utilizatorul poate fi examinat de la o distanță de câțiva metri.
- **Vocea:** S-a dovedit că este posibil a identifica precis pe cineva doar efectuând o analiză spectrală a vocii acestuia. Deşi relativ ieftină, tehnica nu a prins deoarece poate fi uşor păcălită cu un casetofon sau poate da alarme false dacă vocea utilizatorului a suferit modificări din cauza bolii.
- **Timpul de tastare:** Modul exact în care o persoană scrie la tastatură poate servi ca mod de identificare. Problema constă în faptul că diferite afecțiuni pot modifica puternic timpul de tastare şi acest lucru poate duce la identificări eronate.

Toate aceste metode de identificare biometrică au un neajuns comun. Datele biometrice nu se pot ascunde, şi oricine ascultă comunicația în rețea le poate reda ulterior, fără a fi nevoie de autentificarea de fapt. Soluția este folosirea unui protocol securizat de schimb de informații.

Capitolul 2

Atacuri DOS

Atacurile de tip Denial of Service[1] sunt o cauză majoră de operare defectuoasă a sistemelor în Internet şi reprezintă cea mai serioasă ameninţare de astăzi. Primul atac major a îngenunchiat reţeaua Universităţii Minnesota în august 1999, iar 6 luni mai târziu un tânăr canadian a atacat unele dintre cele mai importante site-uri Internet: Yahoo, CNN, Amazon, Buy şi eBay. De atunci atacurile par să fie în creştere.

Din nefericire, utilizatorii sunt mai interesaţi de software-ul cu mai multe facilităţi decât de cel robust, fără erori. În plus, securitatea are preţul său. Software-ul modern cheltuieşte un număr imens de cicluri maşină pentru a desena ferestre tri-dimensionale cu alpha-blending sau alte asemenea îmbunătăţiri vizuale, dar aceste lucruri nu aduc nimic din punct de vedere al funcţionalităţii. Deşi securitatea este o problemă majoră, mulţi utilizatori nu se arată dispuşi să cheltuiască o putere de calcul similară pentru securitate. De asemenea, multor utilizatori nu le pasă dacă sistemul lor este sigur sau poate fi utilizat ca ţintă sau rampă de lansare a unor atacuri de diferite feluri.

Falsul sentiment de securitate este probabil mai rău decât lipsa totală a securităţii. Încă există suficient de mulţi administratori care lasă sistemele lor neprotejate, neaplicând ultimele patch-uri şi neconformându-se procedurilor de bună practică adoptate de fiecare organizaţie. Dacă avem în vedere faptul că numărul de locuinţe, şcoli, biblioteci şi alte locuri publice conectate la Internet a crescut exponenţial în ultima vreme, dimensiunea problemei apare în toată măreţia sa.

Ameninţările de securitate se pot categorisi după cum urmează [MEAD99]:

- Scurgeri de informaţii (confidenţialitate)
- Autentificări nereuşite
- Denial of Service (DOS)

În timp ce primele două atacuri au fost analizate extensiv în literatura de specialitate, atacurile DOS nu au primit atenţia cuvenită până de curând, şi anume după evenimentele din februarie 2000 [CERT00].

[1] Termenul se traduce aproximativ prin „interzicerea serviciului”, dar am ales să folosesc în această lucrare terminologia originală din limba engleză deoarece este un termen de referinţă în articolele de specialitate.

2.1. Definiția unui atac DOS

Un atac Denial of Service poate lua una din cele două forme posibile. Un atacator poate cauza netransmiterea de către rețea a mesajelor pe care ar trebui să le transmită în mod normal clienților săi. De cealaltă parte se află rețelele care trimit mesaje pe care nu ar trebui să le trimită. De departe cel mai cunoscut atac DOS este cauzarea de trafic fals (inundarea rețelei) în direcția un server particular, lucru care în final va duce la împiedicarea clienților legitimi să obțină serviciul pe care îl cer de la acel server.

Comunitatea Internet cunoaşte o serie întreagă de atacuri DOS, care se pot împărți în două categorii: atacuri prin inundare (flooding) şi atacuri prin pachete modificate.

2.2. Cauza atacurilor DOS

O cauză evidentă a atacurilor TCP SYN este că dialogul inițial între părți are loc înaintea unei minime autentificări. Serverul este incapabil de a distinge traficul legitim de cel fals, fapt ce are toate şansele să rămână aşa. Impunerea necesității de a autentifica toate cererile ar însemna un atac DOS în sine, din moment ce serverul ar fi ocupat cu verificarea semnăturilor digital, indiferent dacă acestea sunt sau nu valide. Această nouă cale de atac ar fi la fel de periculoasă ca şi simpla umplere a tabelei TCP.

O cauză mai puțin cunoscută a atacurilor DOS este contabilizarea resurselor, mai precis lipsa acestora. Spatscheck şi Peterson [SPAT99] consideră că sunt trei ingrediente cheie pentru protejarea de atacuri DOS:

[1] *contabilizarea* tuturor resurselor consumate de un client;
[2] *detecția* momentului când resursele alocate unui client depăşesc un prag stabilit apriori;
[3] *constrângerea* – capacitatea de a revendica resursele blocate după detectarea unui atac prin alocarea de resurse minime unui atacator, lucru ce înseamnă automat evitarea unui atac DOS ulterior;

În perioada când Internetul însuşi era proiectat, contabilizarea resurselor era scopul cu prioritatea cea mai mică, lucru ce ne afectează astăzi cel mai mult. În contrast cu rețelele de telefonie omniprezente unde folosirea resurselor era atent controlată, cei care au proiectat rețeaua Internet nu au părut să acorde importanță acestui aspect. Astfel, serverele alocă aceeaşi putere de calcul tuturor cererilor care sosesc la un moment dat ceea ce împiedică o degradare elegantă a performanței în cazul unui atac sau în cazul unei încărcări excesive.

Scenariul anterior este oarecum similar cu mecanismul rudimentar de procesare a pachetelor de intrare datorită arhitecturii bazate pe întreruperi a subsistemului de rețea [DRUS96]. Toate sistemele de operare implementează acest tip de arhitectură care s-a dovedit neadecvat în condiții de încărcare mare. Pachetele de la intrare sunt procesate cu prioritatea maximă pentru ca apoi să fie distruse pentru că nu există nici o aplicație care să le deservească. Această situație se numeşte *receiver livelock.* Mai mult, chiar dacă există o aplicație care să deservească aceste pachete, prioritatea procesului nu este luată în calcul. Astfel, aplicațiile cu prioritate mică primesc aceeaşi cantitate de trafic ca cele cu prioritate mai mare. În lucrarea lor, Druschel şi Banga propun o arhitectură cu de procesare întârziată care se bazează pe demultiplexarea timpurie a pachetelor şi procesarea cu prioritatea destinatarului. Ei susțin că noua arhitectură ar îmbunătăți stabilitatea, justețea şi capacitatea sistemelor în condiții de încărcare mare în timp ce performanța nu ar avea de suferit în condiții normale.

Crosby şi Wallach [CROS] au descris o nouă modalitate de atac bazată pe design-ul intrinsec al protocoalelor. Noua clasă de atacuri de bandă îngustă exploatează deficiențele structurilor de date în diferite aplicații. Structurile de date folosite în mod

uzual au un timp de rulare în cazul mediu mult mai bun decât în cazul cel mai defavorabil. Spre exemplu, atât tabelele de dispersie cât şi arborii binari degenerează în liste înlănțuite atunci când la intrare se prezintă date alese corespunzător. Folosind banda tipică a unui modem, autorii citați au adus un server Bro în pragul colapsului; în şase minute de la începutul atacului, serverul ignora 71% din trafic şi consuma întreaga putere de calcul disponibilă.

2.3. Importanța luptei împotriva atacurilor DOS

Având în vedere tendința globală a piețelor de a se muta on-line, atacurile DOS se dovedesc mult mai periculoase decât s-a prevăzut la început întrucât acestea pot bloca victimele pe durate lungi de timp. De la momentul la care a început atacul şi până când acesta este detectat şi eliminat, victima este paralizată şi nu poate răspunde la cererile legitime. Pentru site-urile comerciale mari aceasta se traduce prin pierderi de ordinul miliardelor de dolari.

Deşi atacurile DOS nu amenință datele în mod direct, nu avem nici un motiv să credem că un altfel de atac nu ar putea uram cu exact acest scop. Aceste atacuri de urmare pot distruge date critice, cauzând astfel mai multe daune decât atacul DOS în sine, ceea ce nu este de dorit.

Acest fel de atacuri în lanț pot avea loc dacă protocoalele continuă dialogul cu atacatorul chiar şi după detectarea unor anomalii în dialogul purtat între părți. Ideea de bază în spatele protocoalelor rezistente (numite protocoale *fail-stop* sau *fail-safe*) este că schimbul de mesaje să înceteze cu orice client care nu urmează cursul firesc al protocolului.

2.4. Atacuri asupra protocoalelor de autentificare

2.4.1. Atacuri prin inundare (flooding)

Atacurile prin inundare sunt comune şi sunt lansate cu intenția de a satura legăturile de rețea pentru a prăbuşi router-ele şi switch-urile sau inundarea cu trafic peste posibilitățile de prelucrare. Din nefericire, uneltele necesare pentru un asemenea atac sunt disponibile pe Internet şi chiar utilizatorii fără experiență le pot folosi cu succes.

Smurf Flood

Smurf Flood este un atac DOS cunoscut şi sub numele de *reflector*. Un atacator trimite un număr mic de pachete echo ICMP la o adresă de broadcast care defineşte mai multe gazde. Răspunsurile tuturor acelor gazde sunt trimise simultan către victimă, epuizând toată banda de comunicație şi posibil puterea de calcul.

TCP SYN

Atacul de tip TCP SYN este posibil datorită schimbului de mesaje de la începutul protocolului TCP. Un client trimite o cerere (SYN) către un server, anunțându-şi intenția de a porni o conversație. La rândul său, serverul desemnează o intrare în tabela cu conexiuni pe jumătate deschise şi trimite înapoi un mesaj de încuviințare (SYN ACK), semnalizând astfel disponibilitatea sa. În acest moment clientul trebuie să răspundă cu un pachet SYN ACK ACK pentru a putea începe comunicația de fapt. Un atacator ar putea să nu trimită niciodată această confirmare, cauzând umplerea tabelei de conexiuni, cererile legitime ulterioare fiind astfel blocate. Daca un atacator trimite o rafală de astfel de cereri, acesta poate paraliza activitatea unui server de 100 MIPS care poate deservi 2000 de

conexiuni pe secundă [SPAT99], dimensiunea tipică a tabelei TCP fiind de 2048 de intrări [DEC96].

UDP Flood (Fraggle)

Acest atac este posibil datorită naturii protocolului UDP care nu este orientat pe conexiune. Din moment ce nu este necesar nici un dialog în prealabil, un atacator poate trimite pachete către porturi aleatoare ale sistemului vizat. Victima va aloca resurse pentru determinarea aplicațiilor care ascultă porturile pe care sosesc date, iar când își dă seama că nici o aplicație nu face acest lucru, va trimite ca răspuns un pachet ICMP. Dacă numărul de pachete aleatoare este suficient de mare există posibilitatea ca sistemul să aibă probleme.

ICMP Flood

Acest atac constă din trimiterea unui număr mare de pachete ICMP către victimă. Aceasta nu poate ține pasul cu volumul de informație primit şi poate observa o degradare a performanței.

E-mail bombing

„E-mail bombing" înseamnă trimiterea unui număr mare de mesaje electronice către un server cu scopul de a epuiza spațiul de pe disc şi lățimea de bandă.

Cu excepția atacului UDP, restul se pot evita prin măsuri luate la nivelul sistemului de operare. Atacul UDP este dificil de contracarat întrucât există o multitudine de aplicații care ascultă la o multitudine de porturi. Filtrarea cu ajutorul firewall-urilor ar avea un impact puternic asupra funcționalității iar acest preț nu îl vor plăti foarte mulți utilizatori.

2.4.2. Atacuri prin pachete modificate

Atacul prin pachete modificate este un alt fel de atac DOS răspândit. Scopul acestui tip de atac este exploatare greşelilor de proiectare a codului care prelucrează pachete din diferite sisteme de operare. Efectele merg de la degradarea performanței până la căderea sistemului.

Aproximativ jumătate din totalul problemelor pe Internet are ca şi cauză depăşirile de buffer. Acestea se cunosc de mai bine de 40 de ani şi cam din aceeaşi perioadă datează şi soluția. Limbajul Algol 60 a introdus limitele obligatorii la tablouri dar programatorii încă refuză să folosească instrumente mai bune. Acest lucru este comparabil cu un fabricant de maşini care înzestrează maşinile cu rezervor din hârtie cerată [DEWA01].

Principalele atacuri cu pachete modificate sunt:

Ping of Death

Acest atac constă în trimiterea unui pachet ICMP mult mai mare decât pachetul maxim IP, şi anume 64 KBytes. La destinație, unele implementări nu pot decodifica pachetul, cauzând prăbuşirea sau reboot-ul sistemului. Două implementări binecunoscute care au acest comportament sunt Windows 95 şi unele versiuni timpurii de Windows NT.

Chargen

Acest atac este o variantă a atacului de tip UDP Flood şi foloseşte portul 19 (chargen) al unui sistem intermediar folosit ca amplificator. Atacatorul trimite un pachet UDP fals către un sistem intermediar care la rândul său răspunde cu un şir de caractere

victimei, pe portul său echo. Victima trimite înapoi un ecou al şirului primit şi bucla creată consumă rapid banda dintre victimă şi sistemul intermediar.

Teardrop

Datorită implementării defectuoase, unele sisteme nu pot asambla fragmente de pachete care au deplasamente eronate. În loc să ignore elegant aceste pachete, aceste implementări blochează sau reboot-ează sistemul.

Land

Deşi este greu de crezut, unele sisteme se blochează când primesc pachete având aceeaşi adresa ca sursă şi destinație.

WinNuke

Acest tip de atac este specific sistemelor de operare Windows. Atacatorul trimite date aleatoare la un port anume, ceea ce cauzează blocarea sau reboot-ul sistemului.

O altă clasificare a atacurilor DOS este după numărul de participanți:

- **Atacuri uni-sursă**: un singur atacator are ca țintă o singură victimă.
- **Atacuri multi-sursă**: mai multe gazde (numite „zombie") participă fără să ştie în rolul atacatorilor, fiind compromise de capul operațiunii. Deşi mai greu de pus în practică, acest tip de atac este şi cel mai periculos şi cel mai greu de luptat împotriva lui. Mai este cunoscut şi sub numele de atac Distributed Denial of Service (DDOS).

[RAZM00] dă un echivalent al atacului DOS în lumea reală: Alice nu îl place pe Bob şi în consecință telefonează mai multor companii de pizza, comandând câte o pizza de la fiecare şi cerând ca acestea să fie trimise la domiciliul lui Bob la o oră anume. Când timpul soseşte, Bob este copleşit de mulțimea de distribuitori de pizza care ajung la domiciliul său, fiecare pretinzând banii cuveniți. Simplu şi eficient, acest atac poate să rămână cu autor necunoscut dacă Alice a sunat de la un telefon public (în esență ascunzându-şi identitatea).

2.4.3. Atacuri la design şi implementare

Atacurile criptografice sunt o categorie aparte de atacuri. Acestea se deosebesc de restul atacurilor prin aceea că se exploatează design-ul şi implementarea de fapt ale protocoalelor de autentificare, fără a se face uz de vreo slăbiciune descoperită întâmplător. Lucrul cel mai grav în cazul acestor atacuri este faptul că parcul de software care înglobează astfel de protocoale este imens iar țintele sunt numeroase. O eventuală modificare pe scară largă a parcului de software în scopul eliminării vulnerabilităților este extrem de costisitoare, nepractică şi de lungă durată.

Atacuri prin complexitate algoritmică

Atacurile prin complexitate algoritmică exploatează deficiențele intrinsece ale diferitelor structuri de date. Este binecunoscut faptul că timpul mediu de execuție în cazul mediu este mult mai bun decât în cazul cel mai defavorabil (spre exemplu tabelele de dispersie şi arborii binari degenerează în liste înlănțuite atunci când datele de intrare prezintă anumite particularități) ceea ce deschide o nouă modalitate de atac. Dacă un atacator are cunoştințe despre implementarea efectivă a unui serviciu (spre exemplu un

server web), folosind date de intrare alese cu grijă poate afecta puternic funcționarea acestuia, chiar având la dispoziție o bandă îngustă de comunicație.

Atacuri prin epuizarea resurselor

Atacurile care exploatează design-ul protocoalelor de autentificare sunt probabil cele mai subtile şi mai greu de combătut. Cel mai cunoscut tip de atac este supraîncărcarea procesorului prin solicitarea de calcule (de regulă inutile), spre exemplu verificarea de semnături digitale necorespunzătoare. Fie că este validă sau nu, verificarea unei semnături digitale este o operație costisitoare care trebuie evitată cu orice preț în lipsa unei minime autentificări prealabile.

Capitolul 3

Tehnologii de apărare

Breşele în securitate prezentate până acum au un număr de remedii propuse. Aici putem diferenția trei abordări ale problemei: eliminarea completă a atacului, ameliorarea efectelor unui atac şi descurajarea atacatorului în sine. Aceste abordări nu pot fi şi nu trebuie utilizate de sine stătător ci trebuie folosite în conjuncție oriunde acest lucru se impune.

3.1. Eliminarea posibilității de atac

Aceasta este cea mai de dorit care de apărare împotriva unui atac DOS din moment ce atacul nu are loc şi efectele asupra victimei sunt nule. Din nefericire, lucrurile nu stau aşa de simplu şi amenințările nu pot fi eliminate complet.

3.1.1. Accesul selectiv la resurse

Mediile închise (spre exemplu intranet-urile companiilor, obiectivele militare) pot profita de pe urma accesului selectiv la resurse, adică doar clienții autentificați pot comunica cu serverele. Această configurație este în mod evident nepotrivită pentru un mediu deschis cum este Internetul. [PING00] citează o problemă cunoscută a mediilor închise, şi anume că intruziunile şi atacurile sunt neaşteptate şi neanticipate. Astfel, nivelul de pregătire în cazul unui eventual atac, în caz că se întâmplă, este foarte scăzut iar pierderile sunt proporțional mai mari.

3.1.2. Semnalizarea în afara benzii

Ideea din spatele semnalizării în afara benzii este că datele şi informațiile de control călătoresc pe canale fizice diferite, evitându-se astfel confuzia şi interferențele. Acest scenariu se aseamănă cu comunicația în rețele de telefonie mobilă (spre exemplu GSM) unde traficul de voce şi semnalizările se transmit în sloturi de timp diferite. [PING00] citează cazul telefoniei terestre din anii 1960 când se folosea semnalizarea în interiorul benzii. O persoană putea literalmente fluiera în microfon iar pentru o anume frecvență şi amplitudine semnalul era interpretat ca semnalizare şi putea oferi utilizatorului un avantaj, ca de exemplu un apel gratuit.

Fără să se bazeze pe experimente de vreun fel, Schneier susține că semnalizarea în afara benzii poate ameliora problemele legate de atacurile de tip DOS. Rămâne de văzut dacă afirmația sa este sau nu adevărată [SCHN00].

3.2. Ameliorarea efectelor atacului asupra victimei

Am văzut că acțiunile pentru împiedicarea atacurilor sunt puține. În aceste condiții este de dorit să avem mecanisme de răspuns care să permită sistemelor să furnizeze un serviciu chiar şi când acestea sunt sub atac. Există mai multe metode pe care le enumerăm mai jos:

3.2.1. Securizarea tuturor calculatoarelor într-o rețea

Aceasta ar însemna sfârşitul sistemelor „zombie", adică cele aflate sub controlul unui conducător de atac. Atacatorii ar fi forțați să lanseze atacuri uni-sursă, micşorând magnitudinea atacului şi uşurând sarcina administrativă de prindere a infractorului. Această abordare este de o importanță practică mică din moment ce securizarea unui sistem este o noțiune relativă, iar îmbunătățirea şi menținerea aceluiaşi nivel de securitate într-un sistem imens cum este Internetul ar fi o sarcină imposibilă.

3.2.2. Măsuri împotriva atacurilor TCP SYN

În cazul atacurilor de tip TCP SYN au fost propuse mai multe abordări, conform [FERG98]:

- *Timeout:* Zona tampon alocat pentru conexiuni TCP deschise pe jumătate este eliberată după un timp determinat. Deşi acest mecanism este simplu de implementat, serverul trebuie să țină seama de conexiunile legitime lente. Dacă un atacator deschide conexiuni mai rapid decât cel mai lent dintre utilizatorii legitimi, atunci acest mecanism nu este eficient.
- *Random Dropping:* Ideea de bază este ca la depăşirea unui anumit prag în alocarea resurselor, serverul închide aleator conexiuni. Problema constă în faptul că şi utilizatorii legitimi au de suferit iar atacul nu este oprit în nici un fel. Dacă atacatorul deschide conexiuni cu o rată ridicată, implementarea acestui algoritm nu aduce o îmbunătățire semnificativă.
- *SYN Cookie:* Această metodă este citată ca cea mai eficientă împotriva atacurilor SYN. Serverul trimite o valoare „V" care este hash-ul unor parametri specifici conexiunii şi o valoare secretă cunoscută numai de server. Serverul nu va aloca nici intrare în zona tampon până nu primeşte înapoi de la client aceeaşi valoare „V" în mesajul SYN ACK. Această metodă presupune că adresa de lansare a atacului este falsă şi atacatorul nu va putea niciodată primi valoarea. Deşi se spune că este o metodă puternică, implementarea presupune că protocolul trebuie să stocheze informații referitoare la fiecare conexiune în parte, ceea ce nu este de dorit întrucât apar alte probleme.

3.2.3. Filtrarea intrărilor

Un atacator ar putea falsifica adresa sursă de la care lansează atacul, ceea ce face ca victima să trimite mesaje SYN ACK la adrese eronate, ceea ce la rândul său duce la imposibilitatea primirii mesajului de confirmare ACK. În RFC 2267 se descrie filtrarea intrărilor, ceea ce previne un atacator să folosească adrese false pentru lansarea atacului.

3.2.4. Filtrarea ieşirilor

Filtrarea ieşirilor asigură că doar pachetele cu adrese IP valide părăsesc reţeaua. Această abordare este efectivă când este implementată aproape de utilizatorul final, dar nu este potrivită pentru implementarea în Internet deoarece deseori furnizorii trebuie să ruteze pachete care nu fac parte din spaţiul lor de adrese.

3.2.5. Împiedicarea cererilor *ICMP echo*

Împiedicarea cererilor ICMP echo de a intra în reţeaua locală ajută la ameliorarea problemei amplificatorului, unde răspunsurile la cereri sunt trimise la o adresă de broadcast. Acest lucru cauzează trafic inutil şi degradarea performanţei.

3.2.6. Dezactivarea serviciilor de reţea nefolosite

Dezactivarea oricăror servicii de reţea nefolosite însemnă în esenţă îngustarea ţintei DOS prin micşorarea numărului de posibilităţi. De asemenea, regulat se descoperă vulnerabilităţi iar atacatorul le poate folosi pentru a produce pagube înainte ca utilizatorul să fie conştient de pericol şi înainte să aplice patch-urile disponibile.

3.2.7. Client Puzzles

Utilizarea client puzzles este una dintre cele mai citate abordări. Ideea din spatele acesteia este încetinirea atacatorului în aşa fel încât atacul DOS să nu mai fie efectiv. Înainte de a aloca resursele, serverul trimite clientului un puzzle. Dificultatea acestuia poate fi uşor schimbată de la 0 la infinit pentru a se potrivi cu încărcarea curentă a serverului. Puzzle-ul propus în mod uzual este inversiunea prin forţă brută a unei funcţii de dispersie cu sens unic cum ar fi MD5 sau SHA1. Această propunere este practică din moment ce funcţiile hash sunt uşor de implementat pe o mare varietate de platforme hardware, iar testarea succesivă a tuturor combinaţiilor de intrare e cea mai eficientă metodă de a inversa o funcţie de dispersie.

3.2.8. Autentificarea progresivă

Autentificarea progresivă este o abordare bazată pe „promiterea” alocării resurselor sistemului în funcţie de nivelul de încredere acordat clientului care face cererea [JUEL99]. Evitarea autentificării puternice de la început este pertinentă deoarece aceasta ar însemna o cale de atac în sine. Plecând de la o autentificare sumară (spre exemplu un cookie) şi primind răspuns pozitiv de la client (spre exemplu clientul urmează paşii ceruţi) se poate trece la autentificare mai puternică şi în final verifică semnătura digitală înainte de alocarea resurselor.

3.2.9. Monitorizarea performanţei

Observarea performanţei sistemului şi stabilirea unor modele de utilizare normală este o chestiune de importanţă relativă. Performanţa sistemului (măsurători combinate ale utilizării CPU, activităţii reţelei şi variaţiei spaţiului de pe disc) este un canal lateral a cărui monitorizare continuă poate duce la detecţia acceselor neautorizate.

3.2.10. Rezoluţia securizată a numelui

Rezoluţia securizată a numelui este o tehnologie nouă propusă de Dewan, Dasgupta şi Karamcheti [DEWA01]. Această tehnică face atacurile DOS dificile prin comunicarea locului unde se furnizează un anumit serviciu doar anumitor clienţi preînregistraţi. În plus, un serviciu static este transformat într-unul dinamic care se mută

ori de câte ori se află sub atac. Protocolul DNS (Domain Name Service) este modificat în așa fel încât să furnizeze într-un mod criptat locația serviciului către toți clienții și se introduce un server de chei care să furnizeze cheile de decriptare doar clienților înregistrați.

3.3. Descurajarea atacatorului

În cazul în care se produce un atac, este de dorit să avem la îndemână o metodă de a trasa atacul înapoi spre inițiatorul său cu scopul de a-l prinde. Acesta este un efort eminamente administrativ și ar trebui folosit complementar cu tehnicile de apărare împotriva atacurilor DOS.

3.3.1. Trasarea adreselor prin coordonarea furnizorilor

Trasarea adreselor prin coordonarea furnizorilor înseamnă cooperarea strânsă între furnizorii a căror rețea a fost traversată de atac. Acest lucru este foarte greu de realizat întrucât politica variază de la furnizor la furnizor, iar o singură ruptură în lanț poate compromite întreaga acțiune. Razmov [RAZM00] propune formarea unui control administrativ și obligarea furnizorilor să cooperez, asemănător cu cooperarea guvernelor în capturarea unui infractor. Pentru a fi cu adevărat efectiv, furnizorii trebuie să aibă o motivație pentru participarea într-un astfel de efort, cum ar fi crearea unei liste de furnizori nesiguri și nepăsători care nu sunt recomandați publicului larg.

3.3.2. Trasarea adreselor prin marcarea probabilistică a pachetelor

Marcarea probabilistică a pachetelor pare cea mai promițătoare abordare pentru descurajarea atacatorilor. Este robustă, implementabilă incremental și compatibilă înapoi, fiind totodată rezistentă la falsificarea adresei și la atacuri de tip denial of service [SAVA00].

Ideea în spatele marcării probabilistice este să se atașeze informație parțială a căii la pachetele care traversează routerele (la care atacatorii nu au acces). În cazul unui atac DOS, ruta pachetelor se poate reconstitui la victimă cu condiția să existe un număr suficient de pachete, ceea ce nu este în general o problemă. Implementarea acestei tehnici ar însemna o realizare remarcabilă din moment ce nu sunt necesare schimbări de infrastructură. Cooperarea strânsă cu autoritățile este însă obligatorie.

Capitolul 4

Client puzzles

Una dintre cele mai promițătoare soluții pentru contracararea efectelor atacurilor de tip *denial of service* este folosirea aşa numitelor client puzzles. Ideea de bază în spatele acestui mecanism este că un eventual client să-şi aloce resursele înaintea serverului ale cărui servicii foloseşte, iar în orice punct al execuției protocolului de autentificare, costul rulării pentru client să fie mai mare decât pentru server. Costul pentru client poate fi crescut artificial prin solicitarea rezolvării unui puzzle al cărui grad de dificultate poate fi stabilit cu uşurință de către server. În acelaşi timp, verificarea corectitudinii soluției nu trebuie să fie o povară pentru server deoarece acest lucru ar anula beneficiile acestei tehnici.

Într-o lucrare din anul 1978, Merkle [MERK78] a fost primul care a venit cu ideea de puzzle-uri criptografice dar a aplicat ideea doar pentru schimbul de chei şi nu în autentificare. Client puzzle-urile au fost aplicate în inundarea TCP SYN de către Juels şi Brainard [JUEL99] care menționează că SSL are aceeaşi slăbiciune şi dau o demonstrație riguroasă a caracteristicilor de securitate. Aura, Nikander şi Leiwo au aplicat puzzle-urile în protocoalele de autentificare în general [TUOM00] iar Dwork şi Naor [DWOR92] au propus măsuri de reglementare pentru mesajele nesolicitate. Rivest, Shamir şi Wagner [RIVE96] discută o problema conexă a criptografiei blocată în timp.

4.1. Descriere

Înainte să aloce resurse pentru deservirea unei conexiuni serverul trebuie să se asigure că pe întreaga durată a execuției protocolului costul pentru client este mai mare decât pentru server. Costul pentru client se poate mări artificial prin solicitarea rezolvării unui puzzle care trebuie să aibă anumite proprietăți., după cum urmează [TUOM00, SHON01]:

- Crearea unui puzzle şi verificarea soluției nu necesită resurse importante din partea serverului.
- Costul rezolvării puzzle-ului este uşor de a fi modificat de la 0 la infinit.
- Puzzle-ul poate fi rezolvat pe majoritatea platformelor hardware.
- Precalcularea soluției puzzle-ului este imposibilă.

- În timp ce clientul rezolvă puzzle-ul, serverul nu trebuie să memoreze soluția sau alte informații specifice clientului.
- Acelaşi puzzle poate fi distribuit mai multor clienți. Cunoscând soluțiile calculate de unul sau mai mulți clienți nu ajută în calcularea unei noi soluții.
- Un client poate reutiliza un puzzle prin crearea mai multor instanțe ale sale.

Puzzle-ul propus în mod uzual este inversiunea prin forță brută a unei funcții de dispersie cu sens unic cum ar fi MD5 sau SHA1. Această propunere este potrivită din moment ce funcțiile hash sunt uşor de implementat pe o mare varietate de platforme hardware, iar testarea succesivă a tuturor intrărilor este cel mai probabil cea mai eficientă metodă de a inversa o funcție de dispersie. Principiul general este arătat în figura 4.

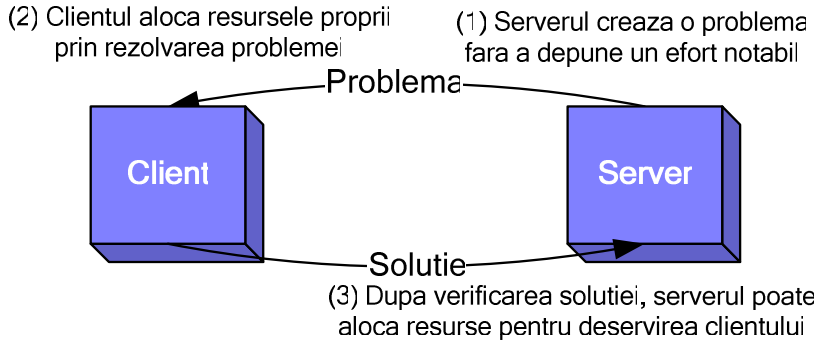


Fig. 4 Principiul *client puzzle*

4.2. Crearea unui puzzle

Periodic (să spunem o dată la câteva minute), serverul generează o valoare aleatoare $\mathbf{N_S}$. Pentru a împiedica atacurile prin ghicire, valoarea trebuie să aibă o entropie de 64 de biți şi să nu fie o valoare predictibilă în vreun fel, ca de exemplu amprenta de timp. Această entropie ar trebui să fie suficientă pentru a împiedica un atacator să calculeze perechi «N_S, rezultat». Potrivirile ocazionale rezultate din atacuri de tip „zi de naştere” nu au efecte prea importante în acest caz.

Serverul trebuie să decidă de asemenea şi nivelul de dificultate **k** al puzzle-ului pe baza unui cumul de măsurători ale condițiilor curente. În rezumat, puzzle-ul trimis clienților arată astfel:

$$\mathbf{« N_S, k »}$$

- N_S – valoarea aleatoare generată de server (de obicei o cantitate cu mărimea de 64 biți)
- k – nivelul de dificultate a puzzle-ului

4.3. Rezolvarea puzzle-ului

Pentru a rezolva puzzle-ul, clientul generează la rândul său o valoare aleatoare $\mathbf{N_C}$. Scopul acestei valori este dublu. În primul rând, dacă clientul refoloseşte valoarea $\mathbf{N_S}$ generată de server, poate construi un nou puzzle prin generarea unei noi valori N_C. În al doilea rând, fără această valoare, un atacator ar putea calcula puzzle-ul înaintea clientului şi ar trimite rezultatul serverului. 24 de biți de entropie ar trebui să fie îndeajuns pentru a împiedica atacatorul de a epuiza întreaga plajă de valori ale N_c dat fiind faptul că N_S se schimbă frecvent.

Clientul trebuie să aplice în mod repetat o funcție de dispersie unei cantități iar puzzle-ul se consideră rezolvat când primii k biți ai cantității Y sunt egali cu 0.

$$\mathbf{h(C, N_S, N_C, X) = Y}$$

1. h – funcție criptografică de dispersie, cum ar fi MD5 or SHA
2. C – identitatea clientului
3. N_S – valoarea aleatoare generată de server
4. N_C – valoarea aleatoare generată de client
5. X – soluția puzzle-ului

Din moment ce serverul schimbă N_S periodic, atât timp cât acest N_S se consideră recent, serverul trebuie să mențină o listă de perechi N_S – N_C în aşa fel soluțiile vechi să nu poată fi refolosite.

Din moment ce nu se cunoaşte nici o metodă ocolitoare pentru a determina X, singura posibilitate este ca această cantitate să se determine secvențial, adică prin forță brută. Nivelul de dificultate k (adică numărul de biți 0 de la începutul lui Y) dictează cât de mult va lua rezolvarea puzzle-ului. Dacă k este egal cu 0 nu este necesar nici un efort, iar dacă k este egal cu 128 (pentru MD5) sau 192 (pentru SHA), clientul trebuie să inverseze o întreagă funcție de dispersie, lucru imposibil computațional.

4.4. Dificultatea puzzle-ului

Parametrul k reprezintă dificultatea puzzle-ului. Sarcina de a stabili această valoare este destul de dificilă deoarece nu există nici o metrică ce s-ar putea folosi într-o implementare reală. În conformitate cu [DEAN], cea mai potrivită abordare ar fi luarea în calcul a numărului de operații RSA alocate deja. Încărcarea curentă a procesorului şi numărul de conexiuni la intrare sunt metrici care depind de un număr de factori care le fac nepotrivite în implementările reale.

Din nefericire, timpul de rezolvare în funcție de dificultatea puzzle-ului urmează o curbă exponențială, aplicabilitatea practică fiind astfel limitată. Pentru a rezolva un puzzle de dificultate k, clientul trebuie să efectueze în medie aproximativ $2^k - 1$ operații. În [TUOM00], Aura, Nikkander şi Leiwo susțin că valorile rezonabile pentru k sunt între 0 şi 64. Prin experiment am aflat că plaja rezonabilă este mult mai îngustă, iar pentru valori mici ale nivelului de dificultate, timpul necesar pentru rezolvarea nivelului k poate fi uneori mai mare decât pentru nivelul k + 1.

Astăzi (începutul anului 2004), un client web tipic este capabil de aproximativ 4500 – 5000 MIPS ceea ce duce la 0,02 milisecunde per operație criptografică. Timpii de execuție pentru fiecare nivel de dificultate sunt dați în tabelul 1:

Tabelul 1 Timpul mediu de execuție pentru diverse niveluri de dificultate

Nivel de dificultate	Timp de execuție (ms)	Nivel de dificultate	Timp de execuție (ms)
1	0	11	37
2	0	12	155
3	0	13	245
4	1	14	461
5	1	15	1022
6	1	16	1968
7	3	17	3232

Nivel de dificultate	Timp de execuție (ms)	Nivel de dificultate	Timp de execuție (ms)
8	1	18	7228
9	15	19	13646
10	37	20	31962

Aşa cum observăm din figura 5, timpul de execuție urmează o curbă exponențială:

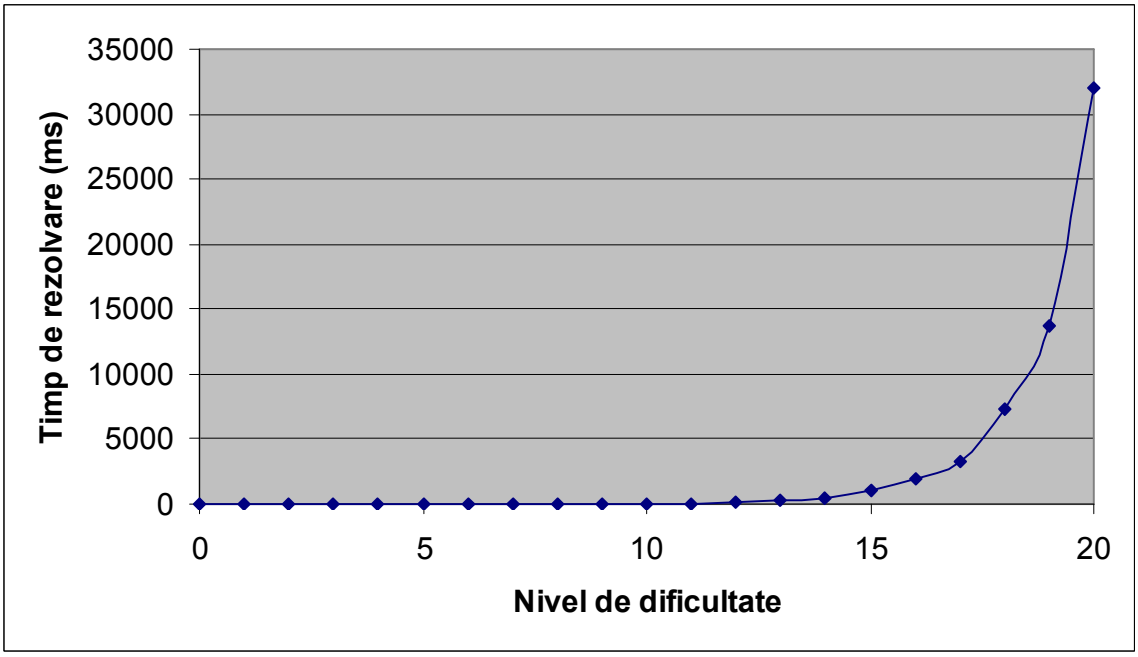


Fig. 5 Curba timpului de execuție pentru diferite niveluri de dificultate

Pentru niveluri de dificultate peste 20, timpul de rezolvare al puzzle-ului devine prohibitiv, de aici aplicabilitatea practică redusă.

Pentru a obține o scală mai precisă pentru parametrul de dificultate al puzzle-ului, Juels şi Brainard [JUEL99] au propus împărțirea problemei în puzzle-uri mai mici de dificultate egală care ar urmau să fie rezolvate separat iar rezultatul total să fie obținut din combinarea rezultatelor parțiale. Aura, Nikkander şi Leiwo [TUOM00] susțin că aceeaşi granularitate poate fi obținută din combinarea sub-problemelor de dificultate diferită dar la un cost mai redus pentru server, fără a demonstra practic viabilitatea soluției.

4.4.1. Algoritm de liniarizare a timpului de rezolvare

Pentru a facilita implementările practice, natura exponențială a timpului de execuție pentru un nivel de dificultate dat trebuie transformată într-o natură liniară. Cea mai potrivită soluție este împărțirea puzzle-ului de dificultate k în puzzle-uri mai mici, de dificultăți diferite, k_1, k_2, ... k_n, după un algoritm de tip greedy. Algoritmul în pseudocod arată în felul următor:

INTRARE:

`Difficulty` – Nivelul de dificultate liniar (între 0 şi 100)

`MaxSolvingTime` – Timpul maxim de rezolvare al puzzle-ului (uzual 20000 – 25000 ms)

`ExecTimes[]` – Timpul mediu de rezolvare pentru fiecare nivel de dificultate în parte (tablou)

IEŞIRE:

`ExpDifficulties[]` – Lista de niveluri de dificultate exponențiale (a căror rezolvare cumulată se face aproximativ în timpul asociat nivelului de intrare)

- Estimează timpul total de rulare: `RunningTime = Difficulty * MaxSolvingTime`
- Cât timp `RunningTime > 0`
  - Găseşte cel mai mare nivel de dificultate din tabelul `ExecTimes` pentru care timpul de execuție este mai mic decât valoarea `RunningTime` actuală.
  - Adaugă nivelul găsit în tabelul `ExpDifficulties`
  - Decrementează valoarea `RunningTime` cu timpul de execuție găsit
- Returnează tabloul `ExpDifficulties`.

Aplicarea acestui algoritm (cu limitarea superioară a timpului de execuție total la 20000 ms) duce la rezultatele din tabelul 2, parametrul ExecTimes având valorile prezentate în tabelul 1.

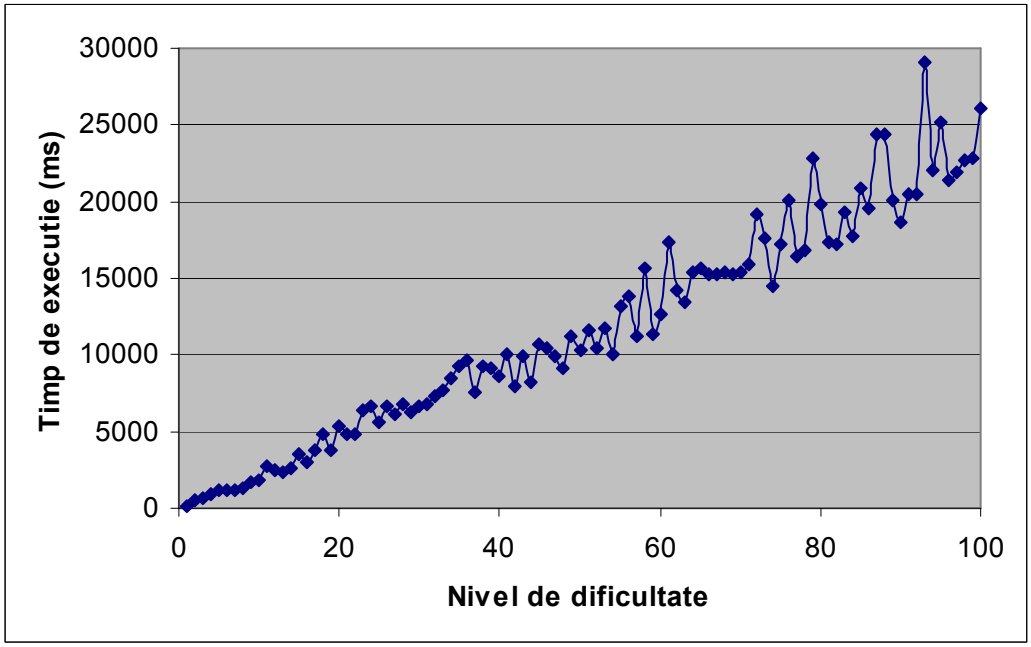
Tabelul 2 Timpul mediu de rezolvare al puzzle-ului după aplicarea algoritmului de liniarizare

Nivel de dificultate	Timp de rezolvare (ms)	Nivel de dificultate	Timp de rezolvare (ms)
1	191	29	6319
2	500	30	6671
3	680	31	6827
4	896	32	7289
5	1132	33	7728
6	1194	34	8489
7	1235	35	9260
8	1247	36	9611
9	1640	37	7534
10	1792	38	9295
11	2695	39	9137
12	2414	40	8598
13	2333	41	9981
14	2575	42	7969
15	3568	43	9888
16	3015	44	8174
17	3749	45	10751
18	4869	46	10426
19	3773	47	9975
20	5360	48	9191
21	4771	49	11217
22	4782	50	10272
23	6439	51	11590
24	6662	52	10417
25	5576	53	11724
26	6606	54	10078
27	6156	55	13180
28	6796	56	13829

Nivel de dificultate	Timp de rezolvare (ms)
57	11198
58	15678
59	11396
60	12660
61	17351
62	14234
63	13433
64	15363
65	15692
66	15198
67	15256
68	15430
69	15277
70	15362
71	15885
72	19129
73	17599
74	14527
75	17173
76	20026
77	16478
78	16770

Nivel de dificultate	Timp de rezolvare (ms)
79	22769
80	19774
81	17384
82	17197
83	19294
84	17678
85	20905
86	19540
87	24382
88	24432
89	20094
90	18692
91	20436
92	20539
93	29126
94	22005
95	25211
96	21430
97	21921
98	22642
99	22847
100	26115

Timpul mediu de execuție este ilustrat grafic în figura următoare. Se observă că acesta urmează o evoluție cvasi-liniară, liniaritatea perfectă neputând fi atinsă deoarece timpul de execuție al unui puzzle poate varia în limite largi pentru un nivel de dificultate dat.



4.5. Threshold puzzles

Client puzzles s-au dovedit eficace atât teoretic cât şi practic în condiţii normale. Totuşi, dificultatea puzzle-ului este stabilită după o metrică ce ia în calcul doar angajamentul serverului şi nu ia în calcul puterea clientului care poate varia în limite largi. Aşa cum am văzut anterior, puzzle-urile sunt generate la momente precise, momente la care se ia în calcul angajamentul instantaneu al serverului. Această „măsură universală" nu este cea mai potrivită în toate scenariile.

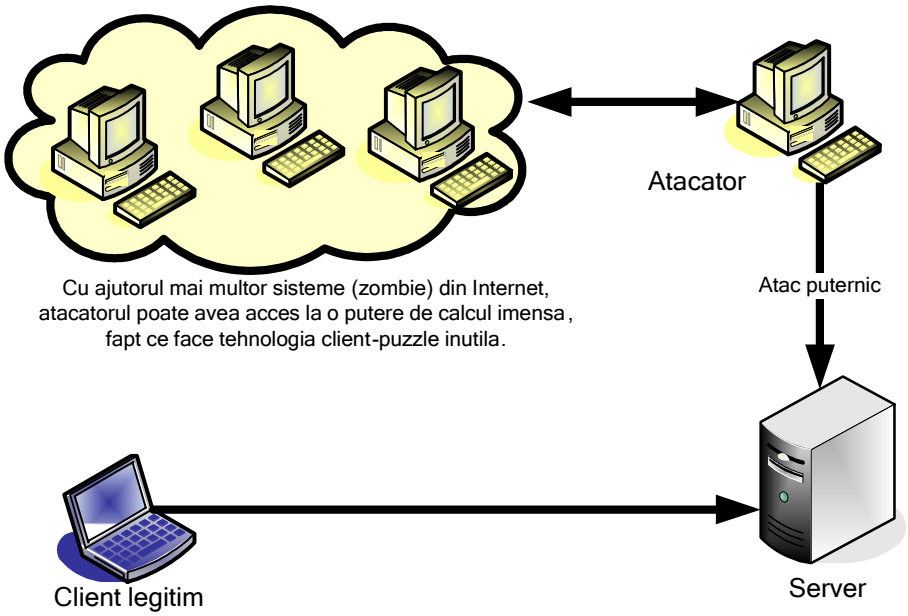


Fig. 6 Schema unui atac puternic

Am remarcat că puzzle-urile sunt vulnerabile la o formă particulară de atac (numită de aici înainte **atac puternic**) datorită naturii paralele a puzzle-ului. Un atac puternic se defineşte ca un atac de tip *denial-of-service* pus la cale de un atacator cu acces la o putere de calcul imensă. În acest caz, atacatorul este capabil de a rezolva puzzle-urile într-un timp mult mai scurt decât un client legitim, după cum se poate vedea în figura 6.

Să presupunem că un server autentifică un număr de clienţi legitimi iar dificultatea iniţială a puzzle-ului este zero. Când un atac puternic este în desfăşurare, serverul are tendinţa de a creşte gradual nivelul de dificultate până la valori mari pentru a ţine pasul cu cantitatea importantă de procesare necesară pentru deservirea cererilor atacatorului. Deşi nivelul de dificultate poate fi crescut până la infinit, acest lucru înseamnă un atac DOS în sine îndreptat asupra clienţilor legitimi care nu ajung niciodată să rezolve un puzzle atât de dificil.

Deşi nu foarte probabil, un atac puternic este posibil. Dacă un atacator ar avea acces la N calculatoare (cu N suficient de mare ca să vorbim de putere de calcul imensă), atunci timpul necesar rezolvării puzzle-ului de dificultate k ar fi împărţit la N. Programul SETI [SETI] şi efortul de a sparge algoritmul RSA [DIST] sunt exemple reale de cum se pot pune la muncă sute de mii de sisteme pentru acelaşi scop comun. Puterea cumulată a reţelei *distributed.net* a depăşit echivalentul a 160000 de calculatoare PII la 266MHz, lucru ce arată clar că atacurile puternice sunt posibile.

Propun două modificări la design-ul actual al tehnologie client puzzles:

- Limitarea superioară a nivelului de dificultate în aşa fel încât puzzle-ul rămâne în limite utile.

- Adăugarea unui timp minim de răspuns definiției puzzle-ului.

4.5.1. Limitarea superioară a nivelului de dificultate

Deşi design-ul curent al puzzle-ului aşa cum este el descris în [TUOM00] specifică o plajă a dificultății între 0 (nici un efort necesar) şi 128 sau 192 (imposibil, în funcție de funcția de dispersie folosită), o implementarea reală a mecanismului are mari şanse să aleagă o plajă mai rezonabilă, să spunem între 0 şi 25 datorită scalei exponențiale care duce la margine îngustă a utilității. Niveluri mai ridicate ar putea să ducă la atacuri DOS îndreptate asupra clienților legitimi, lucru chiar mai grav decât atacul asupra serverului.

4.5.2. Stabilirea unui timp minim de răspuns

Ideea de bază constă în adăugarea unei amprente de timp la care serverul a generat valoarea sa aleatoare la lista «N_S, N_C, X, k». Când serverul primeşte soluția, acesta poate calcula timpul exact de care a avut nevoie clientul pentru a rezolva puzzle-ul. Acest timp nu ar trebui să fie mai mic de o estimarea serverului bazată pe gradul de dificultate. Dacă este, atunci rezultă că serverul se află sub un atac puternic şi ar trebui să înceteze imediat comunicarea cu clientul în cauză. În medie, pentru rezolvarea unui puzzle de dificultate k este nevoie de $2^k - 1$ operații, deci formula de calcul al timpului estimativ este:

$$\mathbf{T_{estimat} = (2^k - 1) * T_{operație}}$$

- $T_{estimat}$ – timpul estimat pentru rezolvarea puzzle-ului
- k – nivelul de dificultate al serverului
- $T_{operație}$ – timpul minim de efectuare a unei operațiuni criptografice (curent în plaja de 0.01 – 0.02 milisecunde, trebuie determinată experimental sau trebuie aplicată legea lui Moore la momentul implementării de fapt)

Timpul estimativ reprezintă pragul de acceptare pentru un client puzzle. Un client puzzle care încorporează modificările menționate poartă numele de **threshold puzzle**.

4.6. Autentificarea rezistentă la atacuri DOS folosind threshold puzzles

Client puzzles au fost folosite pentru conferirea rezistenței la atacuri DOS protocoalelor de autentificare în [TUOM00]. Folosirea threshold puzzles pentru acelaşi scop nu implică schimbări importante, scenariul fiind similar.

Protocolul începe în mod normal prin cererea clientului pentru o conexiune, în forma unui mesaj **ClientHello**. Serverul generează puzzle-ul (parametrii N_S şi k) şi îl trimite clientului printr-un mesaj **ServerHello**. Opțional, mesajul poate fi marcat în timp şi semnat pentru a împiedica atacatorul de a falsifica aceste puzzle-uri. Dacă mesajul ClientHello lipseşte din design-ul protocolului de autentificare, serverul poate transmite mesaje ServerHello cu aceeaşi valoare aleatoare. Această valoare trebuie schimbată periodic.

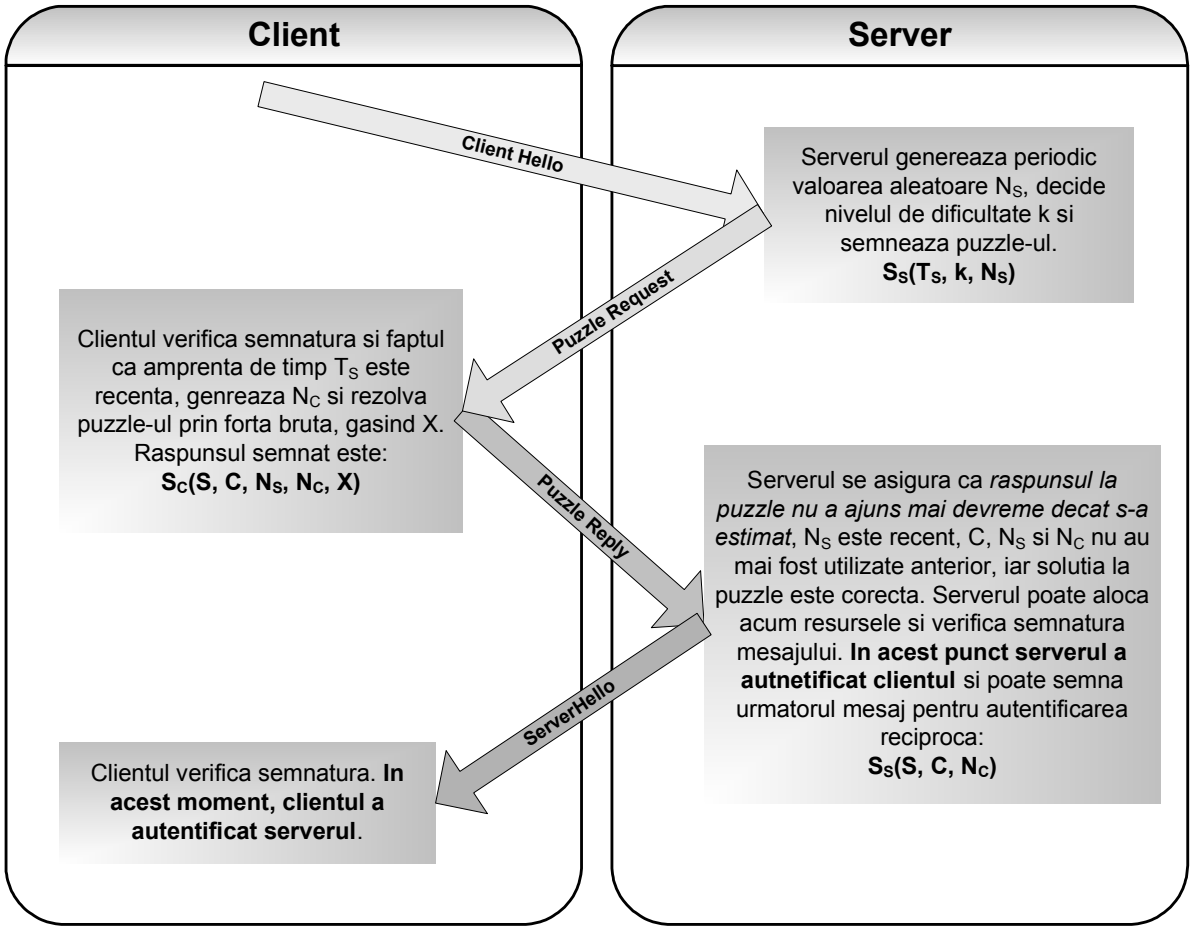


Fig. 7 Schema unui protocol de autentificare baza pe tehnologia threshold puzzle

Orice client care doreşte să comunice cu serverul trebuie să genereze o valoare aleatoare N_C şi trebuie să rezolve corect puzzle-ul iar apoi să furnizeze parametrii C, N_C şi X pentru verificare. În cazul în care clientul doreşte să iniţieze mai multe conexiuni la acelaşi server, acesta poate reutiliza puzzle-ul prin generarea altor valori aleatoare N_C. La recepţionarea rezultatului, serverul verifică dacă clientul C a mai furnizat o soluţie cu aceiaşi parametri N_S şi N_C, verificare care asigură că soluţiile nu sunt refolosite. În acest punct protocolul pe partea de server este diferit faţă de versiunea prezentată în [TUOM00], deoarece serverul efectuează un pas suplimentar. Serverul verifică dacă puzzle-ul a fost rezolvat într-un timp mai scurt decât timpul estimat. Dacă suntem în acest caz, serverul se află sub atac puternic şi încetează comunicaţia cu clientul în cauză, fără a aloca resurse. Dacă timpul depăşeşte estimarea, serverul continuă cu calcularea dispersiei, verifică semnătura şi procedează la executarea protocolului în mod normal. Figura 7 arată schema unui protocol de autentificare protejat de threshold puzzles.

Aşa cum am văzut anterior, pentru a evita refolosirea puzzle-urilor şi a contracara efectele atacurilor puternice, serverul trebuie să menţină o listă de forma următoare:

$$\mathbf{« N_S, N_C, k, C, t_{NS} »}$$

- N_S – valoarea aleatoare generată de server
- N_C – valoarea aleatoare generată de client
- k – nivelul de dificultate al serverului

- C – identitatea clientului
- t_{NS} – Amprenta de timp la care s-a generat valoarea aleatoare N_S

Acest lucru contravine întrucâtva principiului de a nu stoca pe server informații referitoare la clienți deoarece acest lucru deschide o nouă cale de atac, prin epuizarea intrărilor în tabelă, similar atacului TCP SYN. Intrările în tabelă trebuie să se mențină doar atât timp cât N_S se consideră recent.

4.7. Încărcarea serverului

Am văzut anterior că un puzzle constă din două cantități: o valoare aleatoare N_S şi un nivel de dificultate k. Dacă prima cantitate nu pune probleme deosebite, riscurile folosirii unui generator de numere pseudo-aleatoare nefiind important, în cazul determinării nivelului de dificultate ne confruntăm cu o problemă. Nivelul de dificultate este o mărime strâns legată de încărcarea serverului. Cu cât încărcarea este mai mare, cu atât nivelul de dificultate este mai mare, intr-o creştere liniară. Gradul de încărcare este o noțiune complexă şi greu de estimat deoarece ne lovim de dificultatea găsirii unei metrici potrivite.

Dean şi Stubblefield [DEAN] propun ca metrică numărul de operații RSA angajate. Contorul este incrementat când sistemul decide să nu trimită un puzzle sau când un client furnizează o soluție corectă a unui puzzle generat anterior. Contorul se decrementează după ce operația RSA respectivă s-a încheiat sau dacă conexiunea s-a întrerupt înainte ca operația RSA să înceapă. Această metrică măsoară numărul de operații RSA care urmează să fie efectuate în perioada imediat următoare. Metrica are o creştere liniară, funcția de calcul pentru parametrul k fiind:

$$\mathbf{k = N_{RSA} / Total_{RSA} * 100}$$

- N_{RSA} – Numărul total de operații RSA angajate până în prezent
- $Total_{RSA}$ – Numărul maxim de operații RSA simultane pe care sistemul le poate executa fără degradarea performanței
- k – nivelul de dificultate al puzzle-ului (între 0 şi 100)

Nivelul k calculat se aplică funcției de liniarizare prezentată anterior în lucrare pentru a obține combinația optimă de puzzle-uri pentru client.

Pentru a acomoda şi sisteme distribuite (cum ar fi fermele de servere web) metrica ar trebui să ia în calcul încărcarea procesorului, capacitatea sistemului de I/O, schimbările de context, memoria liberă, numărul de întreruperi, etc. Elaborarea unei astfel de metrici este însă dincolo de scopul acestei lucrări.

Capitolul 5

Implementare

Ideile şi conceptele din această lucrare au la bază experimente. Platforma pe care am ales-o pentru implementare este .NET, mai exact limbajul C#. În mod natural, mediul de dezvoltare ales a fost Microsoft Visual Studio .NET[1] rulând sub sistemul de operare Windows XP Professional. Experimentele legate de SSL/TLS au fost realizate pe biblioteca open-source Mentalis.org Security Library[2].

5.1. Puzzle Challenge

Aşa cum am văzut în capitolul anterior, problema propusă spre rezolvare (puzzle-ul) constă din nivelul de dificultate k şi valoarea aleatoare N_S. Această valoare este o cantitate de 64 de biţi. Definiţia în limbajul C# este:

```
/// <summary>
/// This class is used to send the puzzle challenge from the
/// server to the client
/// For brevity, the challenge is not signed
/// </summary>
[Serializable]
public class PuzzleChallenge
{
      /// <summary>
      /// Time stamp for the puzzle challenge (it must be recent
      /// in order to be accepted
      /// by the client)
      /// </summary>
      public DateTime         TimeStamp;

      /// <summary>
      /// Difficulty of the puzzle which ranges from 0
      /// (no work to do) to 128 (impossible)
      /// </summary>
      public int              Difficulty;
```

[1] Versiunea 2003 Enterprise Architect
[2] www.mentalis.org

```
        /// <summary>
        /// Server generated value (64-bit value)
        /// </summary>
        public ulong            ServerNonce;

        public PuzzleChallenge(int Difficulty)
        {
              this.Difficulty   = Difficulty;
              TimeStamp         = DateTime.Now;

              Random rand = new Random();
              ulong a = (ulong)rand.Next();
              ulong b = (ulong)rand.Next();
              ServerNonce = a + 65536 * b;
        }
}
```

Clasa are atributul `public`, ceea ce înseamnă că este accesibilă oricui, în interiorul sau exteriorul assembly-ului. Membrul `TimeStamp` conține timpul la care puzzle-ul a fost creat. Deşi nu face parte din definiția de fapt a unui puzzle, membrul a fost adăugat pentru a putea verifica vechimea acestuia care în mod tipic nu trebuie să depăşească 60 de secunde. Membrul `Difficulty` memorează nivelul de dificultate pentru puzzle, o valoare întreagă între 0 şi 128 (sau 192) care depinde de funcția de dispersie folosită. Valoarea aleatoare generată de server este memorată în variabila `ServerNonce`.

Constructorul clasei `PuzzleChallenge` primeşte ca parametru doar dificultatea dorită pentru puzzle, apoi o memorează în variabila locală destinată acestui scop. Timpul curent este furnizat de proprietatea statică `Now` a obiectului `DateTime` (care face parte din .NET Framework). Deoarece clasa `Random` generează doar valori aleatoare pe 32 de biți, valoarea necesară de 64 de biți se realizează în doi paşi, prin combinarea acestora.

Clasa este marcată cu atributul `[Serializable]` pentru a permite serializarea acesteia şi transmiterea printr-un flux HTTP.

5.2. Puzzle Solution

Soluția unui puzzle este memorată de clasa `PuzzleSolution`, care are ca membri identitatea clientului, valorile aleatoare generate de server şi de client şi soluția de fapt a puzzle-ului, adică o valoare care să satisfacă o condiție stabilită prin convenție. Definiția în limbajul C# este următoarea:

```
/// <summary>
/// This class represents the solution to the puzzle challenge sent by
the client to the server
/// </summary>
[Serializable]
public class PuzzleSolution
{
        /// <summary>
        /// Client identifier (usually the IP address)
        /// </summary>
        public string     ClientID;

        /// <summary>
        /// Server generated value generated in the challenge
        /// (64-bit value)
        /// </summary>
```

```
        public ulong      ServerNonce;

        /// <summary>
        /// Client generated value (32-bit value)
        /// </summary>
        public uint       ClientNonce;

        /// <summary>
        /// Solution to this puzzle
        /// </summary>
        public ulong      Solution;

        /// <summary>
        /// Number of operations performed to compute the solution
        /// </summary>
        public int        NumberOfOperations;

        public PuzzleSolution(string ClientID)
        {
              this.ClientID     = ClientID;
              ClientNonce       = (uint)(new Random()).Next();
              NumberOfOperations = 0;
        }
}
```

Ca şi în cazul clasei `PuzzleChallenge`, această clasă are atributul `public` pentru a fi accesibilă de oriunde. Membrul `ClientID` memorează identitatea clientului sub formă de şir de caractere, în mod uzual adresa sa IP. Membrul `ServerNonce` este cantitatea aleatoare generată de server (copiată din `PuzzleChallenge`),. iar membrul ClientNonce este cantitatea aleatoare de 32 de biţi generată de client. Soluţia puzzle-ului se memorează în membrul `Solution`. Ultimul membru (`NumberOfOperations`) memorează numărul total de operaţii efectuate de sistem pentru a rezolva puzzle-ul. Această cantitate nu are însemnătate criptografică.

Constructorul clasei primeşte ca parametru identitatea clientului pe care o memorează într-o variabilă locală. De asemenea se generează valoarea aleatoare de 32 de biţi. Clasa este marcată cu atributul `[Serializable]` pentru a permite serializarea acesteia şi transmiterea printr-un flux HTTP.

5.3. Puzzle Solver

Această clasă rezolvă un puzzle dat. Implementarea în limbajul C# este următoarea:

```
/// <summary>
/// This class solves the puzzle challenge
/// </summary>
public class PuzzleSolver
{
      public PuzzleSolver()
      {

      }

      /// <summary>
      /// Computes formula h(C, Ns, Nc, X), where:
      /// h - the MD5 cryptographic function
      /// C - client identity
      /// Ns - server nonce
```

```
      /// Nc - client nonce
      /// X - (possible) solution of the puzzle
      /// </summary>
      /// <param name="ClientID">Client Identity</param>
      /// <param name="ClientNonce">Client Nonce</param>
      /// <param name="ServerNonce">Server Nonce</param>
      /// <param name="X">(Possible) solution of the puzzle</param>
      /// <returns>Hash value</returns>
      private static ulong ComputeHash(string ClientID, uint
ClientNonce, ulong ServerNonce, ulong X)
      {
            // Build MD5 cryptographic provider
            MD5CryptoServiceProvider md5 = new
MD5CryptoServiceProvider();

            byte []buff1 =
      BitConverter.GetBytes(ClientID.GetHashCode());
            byte []buff2 = BitConverter.GetBytes(ClientNonce);
            byte []buff3 = BitConverter.GetBytes(ServerNonce);
            byte []buff4 = BitConverter.GetBytes(X);

            int pos = 0;
            byte []buffer = new byte[buff1.Length + buff2.Length +
buff3.Length + buff4.Length];
            Array.Copy(buff1, 0, buffer, pos, buff1.Length);
            pos += buff1.Length;
            Array.Copy(buff2, 0, buffer, pos, buff2.Length);
            pos += buff2.Length;
            Array.Copy(buff3, 0, buffer, pos, buff3.Length);
            pos += buff3.Length;
            Array.Copy(buff4, 0, buffer, pos, buff4.Length);
            pos += buff4.Length;
            // Compute hash
            ulong longhash =
BitConverter.ToUInt64(md5.ComputeHash(buffer), 0);

            return longhash;
      }

      public static PuzzleSolution Solve(PuzzleChallenge Puzzle)
      {
            // TODO: Change client identification
            PuzzleSolution ps = new PuzzleSolution("ClientID");
            // Copy over the server nonce
            ps.ServerNonce = Puzzle.ServerNonce;
            // Verify whether the timestamp is newer than 1 minute
            if(Puzzle.TimeStamp.AddSeconds(60) < DateTime.Now) throw new
Exception("Puzzle is older than 1 minute.");

            for(ulong x = System.UInt64.MinValue; x <=
System.UInt64.MaxValue; x++)
            {
                  ulong longhash = ComputeHash(ps.ClientID,
ps.ClientNonce, Puzzle.ServerNonce, x);
                  ps.NumberOfOperations++;
                  if(BitCounter(longhash) == Puzzle.Difficulty)
                  {
                        ps.Solution = x;
                        break;
                  }
            }
```

```
            return ps;
      }

      /// <summary>
      /// Verifies whether the puzzle solution is correct, given the
difficulty
      /// </summary>
      /// <param name="ps">Puzzle Solution</param>
      /// <param name="Difficulty">Difficulty</param>
      /// <returns>TRUE if the solution is correct</returns>
      public static bool VerifySolution(PuzzleSolution ps, int
Difficulty)
      {
            ulong longhash = ComputeHash(ps.ClientID, ps.ClientNonce,
ps.ServerNonce, ps.Solution);
            return (BitCounter(longhash) == Difficulty);
      }

      /// <summary>
      /// Returns the number of contiguous zero bits from the left
      /// </summary>
      /// <param name="val">Value</param>
      /// <returns>Number of zero bits</returns>
      private static int BitCounter(ulong val)
      {
            int bits = 0;
            for(int i = 0; i < 64; i++)
            {
                  if((val & 0x8000000000000000) == 0)
                  {
                        bits++;
                        val = val << 1;
                  }
                  else
                  {
                        break;
                  }
            }
            return bits;
      }
}
```

Metoda `ComputeHash` evaluează cantitatea h(C, N_S, N_C, X). Fiecare parametru este transformat în reprezentarea sa binară. Sumei acestor parametri (obținută printr-o simplă concatenare) i se aplică funcția MD5. Funcția `BitCounter` primește ca parametru o valoare numerică de 64 de biți şi returnează numărul de biți semnificativi consecutivi cu valoarea 0.

Funcția `Solve` primește ca parametru un obiect de tip `PuzzleChallenge` şi încearcă aflarea soluției puzzle-ului. Primul pas constă din verificarea faptului că puzzle-ul este recent (nu mai vechi de un minut). Găsirea efectivă a soluției se face prin forță brută şi anume prin încercarea succesivă a tuturor valorilor din plaja [System.UInt64.MinValue, System.UInt64.MaxValue] ale parametrului X. Dacă numărul de biți 0 consecutivi este egal cu dificultatea cerută pentru puzzle, valoarea X se consideră soluția puzzle-ului şi se construiește un obiect de tip `PuzzleSolution` care returnat.

Clasa mai pune la dispoziție şi funcția `VerifySolution` de verificare a corectitudinii rezolvării puzzle-ului.

5.4. Mentalis.org Security Library

Biblioteca de securitate Mentalis.org este un add-on pentru .NET Framework. Scopul acestui add-on este oferirea de instrumente privitoare la securitate dezvoltatorilor în C# şi VB.NET. Biblioteca constă curent din următoarele subproiecte:

- Biblioteca SecureSockets (cu suport pentru SSL şi TLS)
- Biblioteca CertificateServices (cu suport pentru managementul certificatelor)
- Biblioteca Crypto (cu suport pentru operații criptografice)

Ca bază în experimentele pe care le-am efectuat am folosit unul dintre exemplele oferite în biblioteca Mentalis.org, WebClient / WebServer. Pentru a se potrivi scopului propus, modulul WebServer a fost modificat în aşa fel încât să accepte conexiuni simultane. Pentru fiecare conexiune care soseşte se porneşte un nou fir de execuție.

```
/// <summary>
/// Starts listening for incoming server connections.
/// </summary>
/// <param name="ep">The EndPoint on which to listen.</param>
/// <param name="sp">The protocol to use.</param>
/// <param name="pfxfile">An optional PFX file.</param>
/// <param name="password">An optional PFX password.</param>
public void StartServer(IPEndPoint ep, SecureProtocol sp, Certificate
cert)
{
      // initialize a SecurityOptions instance
      SecurityOptions options = new SecurityOptions(sp, cert,
ConnectionEnd.Server);
      // create a new SecureSocket with the above security options
      SecureSocket s = new SecureSocket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp, options);
      // from here on, act as if the SecureSocket is a normal Socket
      s.Bind(ep);
      s.Listen(10);
      Console.WriteLine("Listening on " + s.LocalEndPoint.ToString());

      ArrayList Threads = new ArrayList();
      while(true)
      {
            SecureSocket ss = (SecureSocket)s.Accept();
            Console.WriteLine("Socket accepted, starting new thread.");
            // Socket accepted, spawn thread
            ServerThreadDataContainer stdc = new
ServerThreadDataContainer(ss);
            try
            {
                  Thread x = new Thread(new
ThreadStart(stdc.ThreadProc));
                  x.Start();
                  Threads.Add(x);
            }
            catch(Exception)
            {
                  Console.WriteLine("Error spawning new process.");
                  return;
            }

            Console.WriteLine("Waiting for another connection...");
      }
}
```

```
internal class ServerThreadDataContainer
{
      // Since .NET does not directly support parametrized thread
working functions,
      // I chose to use this class to encapsulate thread parameters

      private SecureSocket ss = null;
      private ServerMonitor sm = null;

      public ServerThreadDataContainer(SecureSocket ss)
      {
            this.ss = ss;
      }
      public void ThreadProc()
      {
            sm = new ServerMonitor(ss);
            Thread.Sleep(Timeout.Infinite);
      }
}
```

Deoarece .NET nu suportă în mod direct transmiterea de parametri către firele de execuție, soluția este crearea unei clase auxiliare (`ServerThreadDataContainer` în cazul concret) care să creeze firul de execuție şi care să păstreze local parametrii necesari.

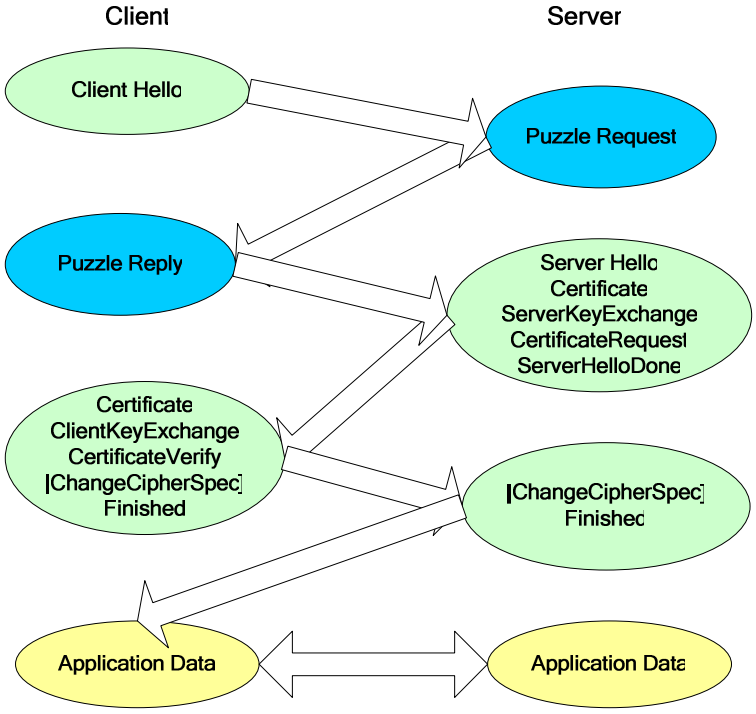


Fig. 8 Schimbul de mesaje SSL în varianta protejată a protocolului

Pentru a implementa efectiva mecanismul client puzzle trebuie să intervenim în schimbul de mesaje SSL. In mod normal, un protocol neprotejat va efectua verificări de certificate şi alte operații costisitoare fără o autentificare prealabilă a clientului, doar la primirea unui mesaj de tip ClientHello. Pentru a încetini un eventual atacator, sistemul de puzzle trebuie plasat înainte ca serverul să efectueze operații scumpe din punct de vedere al timpului de execuție. În acest scop, şirului normal de mesaje SSL i-au fost adăugate

două mesaje noi, PuzzleRequest şi PuzzleReply ale căror implementare este arătată mai jos.

```
/// <summary>
/// Creates a new puzzle and returns the serialized version
/// </summary>
/// <returns>Serialized puzzle</returns>
protected byte[] CreatePuzzle()
{
      // Calculate the difficulty level based on current load
      int Difficulty = ServerLoad.GetDifficultyLevel();
      PuzzleChallenge pc = new PuzzleChallenge(Difficulty);
      // Store the server nonce and the difficulty level so we can later
verify the solution from the client
      NonceList.AddNonce(pc.ServerNonce, Difficulty);
      MemoryStream ms = new MemoryStream();
      IFormatter formatter = new BinaryFormatter();
      formatter.Serialize(ms, pc);
      byte []buffer = ms.ToArray();
      ms.Close();
      return buffer;
}
```

Pentru a crea un puzzle, se apelează inițial funcția `GetDifficultyLevel` care returnează dificultatea calculată pentru puzzle în funcție de condițiile curente de încărcare ale serverului. Odată creat puzzle-ul se memorează perechea de valori `ServerNonce` şi `Difficulty` care să permită sistemului să evite refolosirea parametrilor vechi de către clienți. Pentru a putea fi încapsulat într-un mesaj SSL, puzzle-ul trebuie serializat binar.

La primirea unui puzzle, clientul deserializează mesajul pentru a recupera un obiect de tip `PuzzleChallenge` trimis de server şi procedează la rezolvarea puzzle-ului. Soluția acestuia (un obiect de tip `PuzzleSolution`) este serializat din nou şi încapsulat într-un nou mesaj SSL. Implementarea acestei secvențe este dată mai jos:

```
/// <summary>
/// Function to process the puzzle request from the server
/// </summary>
/// <param name="message"></param>
/// <returns></returns>
protected SslHandshakeStatus ProcessPuzzleRequest(HandshakeMessage
message)
{
      byte[] puzzlesolution = SolvePuzzle(message.fragment);
      MemoryStream ms = new MemoryStream();
      HandshakeMessage hm = new
HandshakeMessage(HandshakeType.PuzzleReply, null);
      hm.fragment = puzzlesolution;
      byte []buffer = m_RecordLayer.EncryptBytes(hm.ToBytes(), 0,
hm.fragment.Length + 4, ContentType.Handshake);
      UpdateHashes(buffer, HashUpdate.All); // output message
      ms.Write(buffer, 0, buffer.Length);
      buffer = ms.ToArray();
      ms.Close();

      return new SslHandshakeStatus(SslStatus.MessageIncomplete,
buffer);
}

/// <summary>
```

```
/// Resolves the puzzle
/// </summary>
/// <param name="PuzzleChallenge">Serialized puzzle challenge</param>
/// <returns>Serialized puzzle solution</returns>
protected byte[] SolvePuzzle(byte[] PuzzleChallenge)
{
      // Decode challenge from array
      MemoryStream ms = new MemoryStream(PuzzleChallenge);
      IFormatter formatter = new BinaryFormatter();
      PuzzleChallenge pc = (PuzzleChallenge)formatter.Deserialize(ms);
      ms.Close();
      // Solve puzzle
      PuzzleSolution ps = PuzzleSolver.Solve(pc);
      // Encode puzzle solution back to array
      ms = new MemoryStream();
      formatter = new BinaryFormatter();
      formatter.Serialize(ms, ps);
      byte []buffer = ms.ToArray();
      ms.Close();
      return buffer;
}
```

5.5. Scenarii de execuție

În paragrafele anterioare am văzut modificările aduse bibliotecii SSL de la Mentalis.org prin care s-a adăugat rezistența la atacuri DOS. Pentru a arăta eficiența tehnologiilor client puzzle şi threshold puzzle în protejarea serverelor care autentifică clienții prin SSL am creat mai multe module de program după cum urmează [BOCA04b]:

- **Client legitim** – un client normal care urmează secvența normală de execuție SSL, incluzând noile mesaje PuzzleRequest şi PuzzleResponse.
- **Client atacator** – un client care are acces la o putere de calcul importantă. Puterea de calcul este simulată prin nerezolvarea de fapt a puzzle-ului din partea clientului şi neverificarea soluției din partea serverului.
- **Server normal** – un server SSL protejat de tehnologia client puzzles.
- **Threshold server** – un server SSL protejat de tehnologia threshold puzzles.

Utilizând două sisteme Pentium IV conectate printr-o rețea Ethernet de 100Mbps am efectuat o serie de teste cu modulele descrise anterior. Modulele client au fost rulate pe Windows 2000 Professional iar modulele server au fost rulate pe Windows 2003 Server Standard. Pentru fiecare test individual am măsurat timpul mediu de serviciu pentru o cerere de acces.

5.5.1. Clienți legitimi care se conectează la un server normal

În condiții normale clienții au suferit doar întârzieri minimale datorate protocolului SSL şi ale transferului prin rețea. Mecanismul de protecție cu client puzzles nu a fost practic folosit deoarece sistemul a fost perfect capabil de a deservi toate cererile, timpul mediu de răspuns fiind de 4545 ms.

5.5.2. Clienți neautorizați care atacă un server normal

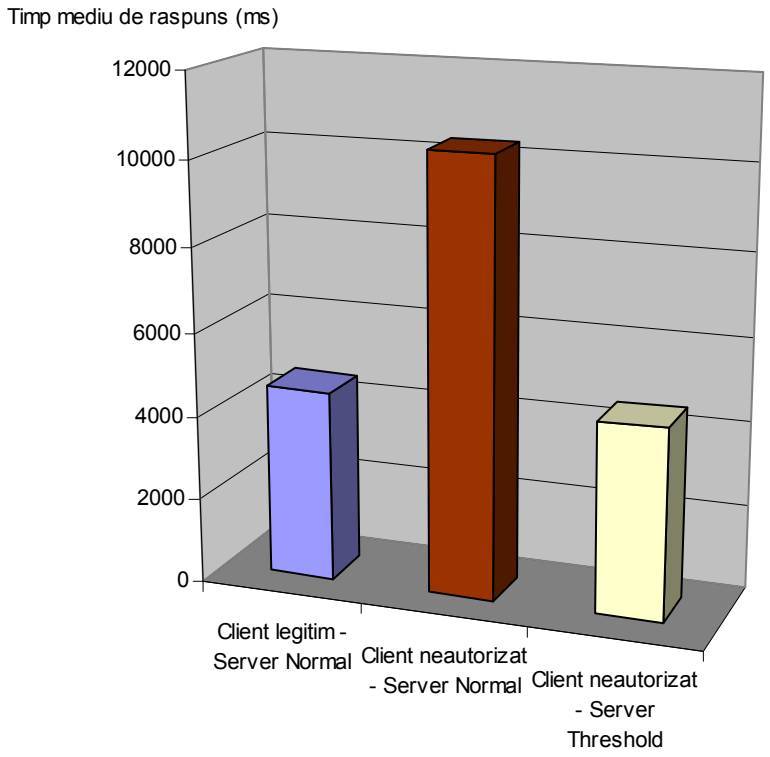
Când cel puțin unul dintre clienți atacă serverul (rezolvând puzzle-ul primit de la server într-un timp mult prea mic față de ceea ce se aşteaptă) încărcarea pe server creşte foarte mult. În consecință serverul creează puzzle-uri din ce în ce mai dificile, un proces

în cascadă care duce la timpi de rezolvare neacceptabil de mari pentru clienții legitimi, fapt ce înseamnă un atac DOS asupra clienților înşişi. În acest caz timpul mediu de răspuns a fost 10339 ms.

5.5.3. Clienți neautorizați care atacă un server threshold

În cazul în care serverul utilizează tehnologia threshold puzzles, clienții care atacă serverul sunt detectați imediat datorită faptului că ei rezolvă puzzle-urile mult mai rapid decât timpul estimat de către server. Serverul nu alocă pentru aceştia resurse semnificative, oprind dialogul normal al protocolului SSL. Timpul mediu de răspuns este în acest caz 4553 ms.

Cele trei scenarii arată clar vulnerabilitatea tehnologiei client puzzles în fața unui atac special. Mai mult decât un simplu atac DOS asupra unui server, atacul se transformă în acest caz într-un atac îndreptat împotriva clienților înşişi, ceea ce înseamnă un pericol deloc de neglijat. Timpii medii măsurați în cele trei experimente sunt cuprinşi în histograma următoare:



Capitolul 6

Concluzii

6.1. Concluzii

Referatul de față face parte din programul de doctorat cu tema *„Protocoale de autentificare în rețele de înaltă siguranță”*, sub coordonarea ştiinţifică a domnului prof. dr. ing. Vladimir CREȚU, de la Facultatea de Automatică şi Calculatoare din cadrul Universității Politehnica din Timişoara.

În cadrul lucrării a fost abordată problematica protocoalelor de autentificare din perspectiva protejării acestora la atacuri de tip DOS (Denial-of-Service). În capitolul 1 s-a definit autentificarea ca fiind *procesul de verificare sigură a identității cuiva sau a ceva* luând ca modele exemplu interacțiunea umană. Considerând definiția, deosebim două cazuri:

- Autentificarea a două sisteme de calcul
- Autentificarea unui utilizator către un sistem de calcul

Fiecare dintre aceste categorii comportă un set unic de detalii şi riscuri pe care le-am evidențiat în lucrare. Procesul de autentificare se face diferențiat, deoarece subiectul autentificării are capacități diferite. Spre exemplu, un sistem de calcul poate memora o cheie criptografică de calitate şi poate efectua operații criptografice. În contrast, un subiect uman are probleme cu memorarea parolelor şi nu poate efectua operații criptografice. Cele trei tehnici principale de autentificare ale utilizatorilor sunt:

- Verificarea a ceea ce utilizatorul **ştie**
- Verificarea a ceea ce utilizatorul **are**
- Verificarea a ceea ce utilizatorul **este**

Parolele sunt o metodă de autentificare ce se încadrează perfect în prima tehnică. Banalele chei de yală se încadrează în cea două tehnică, iar cardurile bancare sunt o combinație a primelor două tehnici. Dispozitivele biometrice, cum ar fi analizoarele vocale şi de amprente se încadrează în a treia tehnică enunțată.

Autentificarea criptografică se pretează a fi folosită între două sisteme de calcul. De-a lungul timpului au apărut o serie de protocoale cum ar fi Kerberos, SESAME, SSL, TLS, SPX şi multe altele. Deşi există diferențe notabile între acestea, ideea de bază din

spatele autentificării este un schimb de mesaje la sfârşitul cărora cele două părți sunt sigure cu privire la identitatea interlocutorului.

Cu toate că domeniul protocoalelor de autentificare a beneficiat de o atenție deosebită din partea comunității academice şi industriale în ultimele decenii, există însă o serie de probleme nerezolvate (capitolul 2) cum ar fi vulnerabilitatea la atacuri de tip DOS. Acest tip de probleme sunt comune celor mai multe dintre protocoalele de autentificare, deoarece resursele disponibile sunt alocate fără o judicioasă evaluare a riscurilor asociate cu fiecare client în parte.

Un atac Denial of Service poate lua una din cele două forme posibile. Un atacator poate cauza netransmiterea de către rețea a mesajelor pe care ar trebui să le transmită în mod normal clienților săi. De cealaltă parte se află rețelele care trimit mesaje pe care nu ar trebui să le trimită. De departe cel mai cunoscut atac DOS este cauzarea de trafic fals (inundarea rețelei) în direcția un server particular, lucru care în final va duce la împiedicarea clienților legitimi să obțină serviciul pe care îl cer de la acel server.

Cele trei ingrediente cheie pentru protejarea de atacuri DOS:

- *contabilizarea* tuturor resurselor consumate de un client;
- *detecția* momentului când resursele alocate unui client depăşesc un prag stabilit apriori;
- *constrângerea* – capacitatea de a revendica resursele blocate după detectarea unui atac prin alocarea de resurse minime unui atacator, lucru ce înseamnă automat evitarea unui atac DOS ulterior;

În capitolul 3 am scos în evidență tehnologiile de ameliorare şi de protejare împotriva atacurilor DOS. Acestea se concentrează pe trei fronturi paralele: eliminarea posibilității de atac (accesul selectiv la resurse, semnalizarea în afara benzii), ameliorarea efectelor atacului asupra victimei (securizarea calculatoarelor într-o rețea, măsuri împotriva atacurilor TCP SYN, filtrarea intrărilor, filtrarea ieşirilor, etc.) şi descurajarea atacatorului (trasarea adreselor prin coordonarea furnizorilor, trasarea adreselor prin marcarea probabilistică a pachetelor).

Tehnologia client puzzle este prezentată pe larg în capitolul 4. Ideea de bază în spatele acestui mecanism este că un eventual client trebuie să-şi aloce resursele înaintea serverului ale cărui servicii foloseşte, iar în orice punct al execuției protocolului de autentificare, costul rulării pentru client să fie mai mare decât pentru server. Costul pentru client poate fi crescut artificial prin solicitarea rezolvării unei probleme a cărei grad de dificultate poate fi stabilit cu uşurință de către server. În acelaşi timp, verificarea corectitudinii soluției nu trebuie să fie o povară pentru server deoarece acest lucru ar anula beneficiile acestei tehnici.

Puzzle-ul trebuie să aibă următoarele proprietăți:

- Crearea unei puzzle şi verificarea soluției nu necesită resurse importante din partea serverului.
- Costul rezolvării puzzle-ului este uşor a fi modificat de la 0 la infinit.
- Puzzle-ul poate fi rezolvat pe majoritatea platformelor hardware.
- Precalcularea soluției puzzle-ului este imposibilă.
- În timp ce clientul rezolvă puzzle-ul, serverul nu trebuie să memoreze soluția sau alte informații specifice clientului.
- Acelaşi puzzle poate fi distribuit mai multor clienți. Cunoscând soluțiile calculate de unul sau mai mulți clienți nu ajută în calcularea unei noi soluții.
- Un client poate reutiliza un puzzle prin crearea mai multor instanțe ale sale.

Puzzle-ul propus în mod uzual este inversiunea prin forță brută a unei funcții de dispersie cum ar fi MD5 sau SHA1. Propunerea este potrivită din moment ce funcțiile

hash sunt uşor de implementat pe o mare varietate de platforme hardware. În plus, literatura de specialitate citează testarea succesivă a tuturor combinațiilor de intrare ca fiind cea mai eficientă metodă de a inversa o funcție de dispersie, în acest moment neexistând o metodă mai eficientă decât forța brută.

Cu toate acestea, tehnologia client puzzle este imperfectă. Operația de inversare a funcției de dispersie prezintă un grad înalt de paralelism astfel încât timpul de rezolvare al puzzle-ului scade liniar cu resursele folosite. Un client cu acces la resurse de calcul importante ar putea să rezolve puzzle-ul într-un timp mult mai scurt decât timpul mediu de rezolvare obținut de clientul tipic. Acest lucru îi permite să lanseze aşa numitele *atacuri puternice*, care pot anula avantajul oferit de tehnologia client puzzle. Pentru a elimina şi această (improbabilă) cale de atac, tehnologiei client puzzle i-am adus o serie de modificări menite să-i confere rezistență şi la atacurile puternice:

- Limitarea superioară a nivelului de dificultate în aşa fel încât efectul puzzle-ului rămâne între limite utile.
- Adăugarea unui timp minim de răspuns definiției puzzle-ului.

În cadrul acestei lucrări s-a introdus conceptul de *threshold puzzle* ca fiind modelul *client puzzle* cu cele două modificări propuse.

Capitolul 5 prezintă implementarea sub platforma .NET (în limbajul C#) ale conceptelor prezentate în lucrare. Au fost descrise în detaliu modulele (puzzle challenge, puzzle solution, puzzle solver, etc.) şi rezultatele rulărilor care confirmă considerațiile teoretice din cadrul prezentei lucrări.

În cele din urmă, anexa 1 prezintă mecanismul de funcționare al protocolului SSL/TLS.

6.2. Rezumatul contribuțiilor

Referatul de față avansează următoarele idei principale:

- Studiile de securitate asupra protocoalelor de autentificare au omis sistematic analiza comportamentului în cazul atacurilor DOS, lăsând deschisă această breşă de securitate.
- Utilizarea tehnologiei client puzzle în protejarea protocoalelor de autentificare s-a dovedit o soluție viabilă, dovedită atât teoretic cât şi practic.
- Tehnologia client puzzle este vulnerabilă la aşa numitele atacuri puternice, deci se impune o revizuire a acesteia şi modificarea ei.
- Timpul de rezolvare al unui puzzle creşte exponențial cu nivelul de dificultate, de unde limitarea în aplicațiile practice.

Având la bază aceste principii, lucrarea aduce următoarele contribuții în domeniul securizării sistemelor de autentificare:

- Definirea conceptului de *threshold puzzle* – Un puzzle cu un nivel de dificultate limitat superior şi cu un timp de rezolvare limitat inferior.
- Definirea conceptului de *atac puternic* – Un atacator care are acces la un număr mare de sisteme interconectate (în aşa fel încât putem vorbi de putere de calcul masivă) poate încerca un atac de tip DOS asupra unui sistem de autentificare protejat cu client puzzle.
- Stabilirea unui nivel superior de dificultate pentru un puzzle – creşterea sa liberă până la valori înalte poate bloca execuția pe partea de client pentru un timp proporțional exponențial cu dificultatea.
- Stabilirea unui timp estimativ minim de răspuns pentru puzzle – un client tipic rezolvă un puzzle de o dificultate dată într-un timp care se poate estima.

Rezolvarea acestuia într-un timp semnificativ mai mic este o indicație că un client încearcă un atac puternic.

- Propunerea unei metrici de evaluare a încărcării unui sistem care să ia în calcul mai multe variabile, cum ar fi încărcarea procesorului, capacitatea sistemului de I/O, schimbările de context, memoria disponibilă, numărul de întreruperi, etc. Propunerile existente în literatura de specialitate iau în calcul doar numărul de operații RSA acceptate.
- Stabilirea unui algoritm de tip greedy pentru liniarizarea timpului de rezolvare al unui puzzle. Modelul utilizat curent urmează o curbă exponențială care împiedică un reglaj fin al nivelului de dificultate, aplicabilitatea practică fiind astfel afectată.
- Avertizarea asupra posibilității de atac asupra clienților care încearcă să se conecteze la un server neprotejat cu threshold puzzle, aflat sub atac puternic. Dacă serverul aflat sub atac creşte nivelul de dificultate al puzzle-ului până la valori superioare, timpul de rezolvare al puzzle-ului pentru clienții legitimi devine prohibitiv, ceea ce se traduce printr-un atac DOS îndreptat împotriva clienților înşişi.
- Avertizarea asupra posibilității de atac asupra implementării client puzzle, similar cu TCP SYN. Lista de puzzle-uri deja rezolvate de clienți trebuie memorată la nivelul serverului, acest lucru comportând un risc în cazul unui atac puternic.

6.3. Perspective de cercetare şi dezvoltare

Preocupările curente şi de viitor din cadrul activității de doctorat cuprind următoarele subiecte:

- Analiza sistemelor de tip *single sign-on* existente şi identificarea problemelor cu care se confruntă acestea.
- Testarea sistemelor single sign-on în condiții de atac DOS.
- Proiectarea şi implementarea unui sistem single sign-on cvasi-universal care să unifice autentificarea în rețele neomogene (de exemplu Intranet – Internet), cu suport pentru metode de autentificare multiple şi cu rezistență nativă la atacuri DOS.

Bibliografie

[BOCA01] Valer Bocan – *Stadiul actual al dezvoltării sistemelor de securitate pentru rețele de calculatoare de înaltă siguranță*, Referat doctorat nr. 1, Universitatea „Politehnica" Timişoara, Facultatea de Automatică şi Calculatoare, 2001

[BOCA04a] Valer Bocan – *Developments in DOS Research and Mitigating Technologies*, Periodica Politehnica, Transactions on Automatic Control and Computer Science, Vol. 49 (63), CONTI 2004

[BOCA04b] Valer Bocan – *Threshold Puzzles: The Evolution of DOS-resistant Authentication*, Periodica Politehnica, Transactions on Automatic Control and Computer Science, Vol. 49 (63), CONTI 2004

[CERT00] Computer Emergency Response Team - *CERT advisory CA-2000.01 Denial of service developments*, 2000 (http://www.cert.org/advisories/CA-2000-01.html)

[CROS] Scott A. Crosby, Dan S. Wallach – *Denial of Service via Algorithmic Complexity Attacks*, Proceedings of the12th USENIX Security Symposium, 2003

[DEAN] Drew Dean, Adam Stubblefield – *Using Client Puzzles to Protect TSL*, http://www.csl.sri.com/users/ddean/papers/usenix01b.pdf, Proceedings of the10th USENIX Security Symposium, 2001

[DEC96] Digital Equipment Corporation – *Performance tuning tips for Digital Unix*, iunie 1996, http://www.abdn.ac.uk/local/apache/manual_1.3.4/misc/perf-dec.html

[DEWA01] Prashant Dewan, Partha Dasgupta, Vijay Karamcheti – *Defending Against Denial of Service Attacks Using Name Secure Resolution*, The 23rd International Conference on Distributed Computing, 2003

[DIST] The Distributed.net Organization, http://www.distributed.net

[DRUS96] Peter Druschel, Gaugrav Banga – *Lazy receiver processing (LRP): a network subsystem architecture for server systems*, Proceedings of the 2nd USENIX Symposium on OSDI, Seattle, 1996

[DWOR92] Cynthia Dwork, Moni Naor – *Pricing via Processing or Combating Junk Mail*, Proceedings of CRYPTO '92, Springer Verlag, 1992

[FERG98] P. Ferguson, D. Senie – *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*, RFC 2267, 1998

[FREI96] Alan O. Freier, Philip Karlton, Paul C. Kocher – *The SSL Protocol, Version 3.0 (Internet Draft)*, Transport Layer Security Working Group, 1996

[JUEL99] Ari Juels, John Brainard – *Client puzzles: A cryptographic defense against connection depletion attacks*, Proceedings of the NDSS 1999

[KAUF02] Charlie Kaufman, Radia Perlman, Mike Speciner – *Network Security. Private Communication in a Public World,* Prentice Hall, 2002

[MEAD99] Catherine Meadows – *A formal framework and evaluation method for network denial of service*, Proceeding of the 1999 IEEE Computer Security Foundations Workshop, Mordano, Italy, 1999

[MERK78] R. C. Merkle – *Secure Communications Over Insecure Channels*, Communications of the ACM, 1978

[OPPL96] Rolf Oppliger – *Authentication Systems for Secure Networks*, Artech House, Inc., 1996

[PATR94] Victor-Valeriu Patriciu – *Criptografia şi securitatea reţelelor de calculatoare*, Editura Tehnică, 1994

[PING00] Ping-Herng Denny Lin – *Survey of the Denial of Service Countermeasures*, California State University, Fullerton, 2000

[RAZM00] Valentin Razmov – *Denial of Service Attacks and How to Defend Against Them*, University of Washington, 2000

[RIVE96] Ronald R. Rivest, Adi Shamir, David A. Wagner – *Time-lock Puzzles and Timed-release Cryptography*, 1996, http://lcs.mit.edu/~rivest/RivestShamirWagner-timelock.pdf

[SAVA00] Stefan Savage, David Wetherall, Anna Karlin, Tom Anderson – *Practical network support for IP traceback*, technical report UW-CSE-00/02/0, SIGCOMM '00, 2000

[SCHN00] Bruce Schneier – *Distributed denial of service attacks*, Crypto-gram newsletter, 2000

[SETI] SETI @home Program, http://setiathome.ssl.berkely.edu

[SHON01] Shon Harris – *DOS Defense*, Information Security Magazine, 2001

[SPAT99] Oliver Spatscheck, Larry Peterson – *Defending against denial of service in Scout*, Proceedings of 3rd USENIX/ACM Symposium on OSDI, p. 59-72, 1999

[TUOM00] Tuomas Aura, Pekka Nikander, Jussipekka Leiwo – *DOS-resistant authentication with client-puzzles*, Proceeding of the Cambridge Security Protocols Workshop 2000, LNCS, Cambridge, UK, 2000

Anexa 1

SSL/TLS

Conform definiției din documentele oficiale, protocolul SSL/TLS este un protocol de securitate care oferă comunicare secretă prin Internet. Protocolul permite aplicațiilor client / server să comunice într-un fel în care se împiedică capturarea, modificarea sau falsificarea mesajelor.

A.1. Scurt istoric

SSL (Secure Sockets Layer) versiunea 2 a fost implementată în Netscape Navigator 1.1 de către Netscape în 1995. Versiunea 1 nu a fost implementată niciodată. Microsoft a eliminat unele probleme de securitate din SSL v2 şi în buna sa tradiție, a introdus un protocol proprietar numit PCT (Private Communications Technology). Urmarea a fost că Netscape a modificat substanțial SSL şi aşa a ieşit versiunea 3. IETF a realizat faptul că trei protocoale incompatibile cu funcțiuni similare dăunează industriei şi drept urmare a introdus un al patrulea protocol, TLS. Astăzi protocolul *de facto* este SSL v3 şi rămâne de văzut dacă TLS va fi adoptat pe scară largă. În cele ce urmează, când se menționează SSL se va înțelege implicit şi protocolul TLS, iar diferențele se vor evidenția acolo unde se impune.

A.2. Prezentare

Scopul principal al protocolului SSL este de a oferi confidențialitate şi încredere între două aplicații care comunică. Protocolul constă din două straturi: La cel mai de jos nivel, plasat deasupra unui protocol sigur de comunicație (spre exemplu TCP) se află protocolul *SSL Record*. Acesta este folosit pentru încapsularea protocoalelor de nivel mai înalt, cum ar fi protocolul *SSL Handshake* care permite serverului şi clientului să negocieze un algoritm de criptare şi cheile criptografice corespunzătoare înainte ca aplicațiile să schimbe vreun mesaj. Un nivel superior al protocolului ar putea sta deasupra acestuia în mod transparent.

Protocolul SLL oferă securitatea conexiunii, având trei proprietăți:

- Conexiunea este privată. Criptarea se utilizează după un înțelegerea inițială pentru definirea cheii secrete. Criptografia simetrică se utilizează pentru codificarea datelor (DES, RC4, etc.)

- Identitatea părților se autentifică utilizând criptografie asimetrică (RSA, DSS, etc.)
- Conexiunea este de încredere. Transportul mesajului include verificarea acestuia cu un MAC parametrizat. Pentru calcularea MAC-ului se folosesc funcțiile de dispersie SHA, MD5, etc.

Scopurile protocolului SSL, în ordinea priorităților sunt:

- **Securitatea criptografică**: SSL ar trebui să se utilizeze pentru conexiune sigură între două părți.
- **Inter-operabilitate**: Programatori independenți ar trebui să fie capabili de a dezvolta aplicații SLL care să funcționeze cu succes fără ca aceştia să aibă cunoştință de codul scris de altcineva.
- **Extensibilitatea**: SSL încearcă să furnizeze un cadru în care să se integreze metode noi de criptare (simetrică sau asimetrică), acest lucru contribuind la evitarea creării unui protocol nou (riscând implicit să se introducă noi slăbiciuni) şi evitarea scrierii unei biblioteci de securitate noi.
- **Eficiența relativă**: Operațiile criptografice tind să fie puternic procesor intensive, în particular criptografia cu chei publice. Din acest motiv, SSL a înglobat un mecanism de caching al sesiunii pentru a reduce numărul de conexiuni ce trebuie stabilite în totalitate.

SSL este un protocol stratificat. Pe fiecare strat, mesajele pot conține câmpuri pentru lungime, descriere şi conținut. SSL primeşte mesajele de transmis, fragmentează informația în blocuri prelucrabile, opțional comprimă datele, aplică un MAC, codifică şi în final transmite rezultatul. Datele recepționate sunt decriptate, verificate, decomprimate şi reasamblate, apoi oferite protocolului superior ierarhic.

O sesiune SSL are stare. Este responsabilitatea protocolului SSL Handshake să coordoneze stările clientului şi ale serverului, permițând ca sistemele să funcționeze în mod consistent, în ciuda faptului că stările nu sunt exact paralele. În mod logic, starea este reprezentată de două ori, o dată ca starea curentă în operare şi (în timpul protocolului de handshake) ca starea în aşteptare. În plus, se mențin stări separate ale scrierii şi citirii. Când clientul sau serverul recepționează un mesaj de schimb a specificației cifrului, se copiază starea în aşteptare în starea curentă de citire. Când negocierea este completă, serverul şi clientul interschimbă mesajele de specificare a cifrului şi comunică prin noul cifru asupra căruia s-a convenit.

O sesiune SSL poate include mai multe conexiuni sigure, în plus participanții pot avea mai multe sesiuni simultane. Starea sesiunii poate cuprinde următoarele elemente:

- **Identificator de sesiune**: O secvență arbitrară de octeți aleasă de server pentru a identifica o sesiune activă sau reluabilă.
- **Certificat al egalului:** Certificat X.509 al părții cu care se comunică. Acest element poate să fie nul.
- **Metoda de compresie:** Algoritmul utilizat pentru a comprima datele înainte de codificare.
- **Specificația cifrului:** Specifică algoritmul de codificare a datelor (cum ar fi nimic, DES, AES, etc.) şi un algoritm MAC (cum ar fi MD5 sau SHA). De asemenea defineşte atributele criptografice cum ar fi dimensiunea tabelei de dispersie.

- **Secretul master:** O valoare secretă de 48 de octeți cunoscută de client şi de server.
- **Capacitatea de reluare:** Un indicator care arată dacă o sesiune se poate utiliza pentru a iniția noi conexiuni.

Starea conexiunii poate include următoarele elemente:

- **Valori aleatoare pentru client şi server**: Secvențe de octeți alese de client şi de server pentru fiecare conexiune.
- **Secret MAC pentru scrierea pe server:** Secretul utilizat în operații MAC pe datele scrise de server.
- **Secret MAC pentru scrierea pe client:** Secretul utilizat în operații MAC pe datele scrise de client.
- **Cheia de scriere a clientului:** Cheia pentru datele codificate de server şi decodificate de client.
- **Cheia de scriere a serverului:** Cheia pentru datele codificate de client şi decodificate de server.
- **Vectori de inițializare:** Când se utilizează un cifru bloc în modul CBC, se păstrează un vector de inițializare pentru fiecare cheie. Acest câmp este inițializat de protocolul SSL Handshake.
- **Numere de secvență:** Fiecare participant menține numere de secvență pentru mesajele transmise şi recepționate pentru fiecare conexiune. Când un participant trimite sau recepționează un mesaj de modificare a specificațiilor cifrului, acest număr se pune pe zero. Numere de secvență sunt de tipul *uint64* şi nu trebuie să depăşească valoarea $2^{64} – 1$. [FREI96]

A.3. Arhitectură

Protocolul SSL foloseşte o combinație de criptări simetrice şi cu chei publice. Criptarea cu chei simetrice este mult mai rapidă decât cea cu chei publice, dar aceasta din urmă oferă metode de autentificare mai bune. O sesiune SSL începe întotdeauna cu un schimb de mesaje numit SSL Handshake. Acesta permite serverului să se autentifice clientului folosind criptografia cu chei publice, apoi permite clientului şi serverului să coopereze în crearea cheilor simetrice utilizate pentru codificarea şi decodificarea rapidă precum şi detecția modificării mesajelor. Opțional, protocolul permite şi clientului să se autentifice serverului.

A.3.1. SSL Handshake

În cele ce urmează vom descrie paşii urmați de un sistem în timpul protocolului SSL Handshake, fără a intra în detalii de programare:

- Clientul trimite serverului versiunea sa SSL, setările cifrurilor, date generate aleator şi alte informații necesare în cadrul viitoarei comunicații securizate SSL.
- Serverul trimite clientului versiunea sa SSL, setările cifrurilor, date generate aleator şi alte informații necesare în cadrul viitoarei comunicații securizate SSL. De asemenea, serverul trimite propriul certificat şi dacă clientul cere o resursă care necesită autentificarea clientului, cere certificatul acestuia.

- Clientul utilizează unele dintre informațiile primite pentru autentificarea cu serverul. Dacă serverul nu poate fi autentificat, utilizatorul este atenționat asupra faptului că nu se poate stabili o comunicație autentificată şi securizată.
- Utilizând toate informațiile deținute până acum, clientul (în cooperare cu serverul, în funcție de cifrul utilizat) creează un **secret premaster** pentru sesiune, îl criptează cu cheia publică a serverului (obținut din certificatul serverului în pasul 2) şi trimite acest secret serverului.
- Dacă serverul a cerut autentificarea clientului (un pas opțional în cadrul protocolului), clientul mai semnează o informație unică cunoscută de cele două părți. În acest caz, clientul trimite atât informația semnată cât şi certificatul său serverului, precum şi secretul premaster codificat.
- Dacă serverul a cerut autentificarea clientului, acesta încearcă să autentifice clientul. Dacă clientul nu poate fi autentificat, sesiunea se încheie. Dacă clientul se autentifică cu succes, serverul decriptează secretul premaster folosind cheia sa privată apoi efectuează o serie de paşi (ca de altfel şi clientul) pentru a genera **secretul master.**
- Atât clientul cât şi serverul folosesc secretul master pentru a genera chei de sesiune, care sunt chei simetrice folosite la codificare şi decodificarea informației precum şi la verificarea integrității datelor în timpul sesiunii SSL.
- Clientul trimite un mesaj către server informându-l că mesajele viitoare de la acesta vor fi codificate cu cheia de sesiune. Apoi trimite un mesaj codificat prin care se specifică faptul că partea de client a protocolului de handshake s-a încheiat.
- Serverul trimite un mesaj către client informându-l că mesajele viitoare de la acesta vor fi codificate cu cheia de sesiune. Apoi trimite un mesaj codificat prin care se specifică faptul că partea de server a protocolului de handshake s-a încheiat.
- Protocolul SSL Handshake este acum încheiat, iar sesiunea SSL a început. Clientul şi serverul folosesc cheile de sesiune pentru a codifica şi decodifica datele care se schimbă între cele două părți.

Autentificarea serverului

Software-ul client cu capabilități SSL necesită întotdeauna autentificarea serverului. Aşa cum s-a arătat în pasul 2, serverul trimite clientului certificatul pentru autentificarea sa. Clientul foloseşte acest certificat în pasul 3.

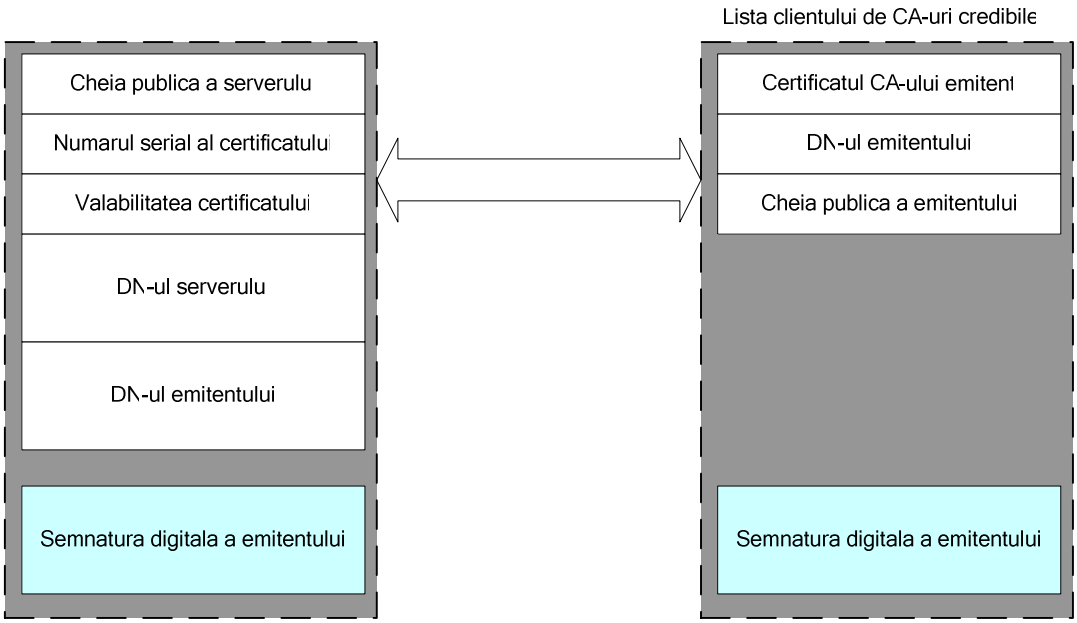


Fig. 9 Autentificarea certificatului unui client

Pentru a autentifica legătura dintre o cheie publică şi un server identificat de un certificat care conţine o cheie publică, clientul SSL trebuie să recepţioneze un răspuns „da” la toate cele patru întrebări ilustrate mai jos. Deşi a patra întrebare nu este în mod tehnic parte din protocolul SSL, cade în responsabilitatea clientului să suporte această cerinţă care oferă asigurarea identităţii serverului şi protejează împotriva unei forme de atac numită „omul din mijloc”.

Un client SSL urmează paşii descrişi în continuare pentru a autentifica identitatea serverului:

- **Data de astăzi este în perioada de valabilitate?** Clientul verifică perioada de validitate a certificatului serverului. Dacă data şi timpul curent sunt în afara perioadei de valabilitate, procesul de autentificare nu mai continuă, altfel se trece la pasul 2.
- **CA-ul emitent este credibil?** Fiecare client menţine o listă de CA-uri credibile, reprezentată în partea dreaptă a figurii 18. Această listă determină care dintre certificate vor fi acceptate de client. Dacă numele (DN – distinguished name) al CA-ului emitent se potriveşte cu numele unui CA de pe listă, răspunsul la întrebare este „da” şi se trece la pasul 3. Altfel, serverul nu este autentificat până când clientul nu verifică faptul că CA-ul se află într-un lanţ credibil, aflat pe listă.
- **Cheia publică a CA-ului emitent verifică semnătura digitală a emitentului?** Clientul utilizează cheia publică din certificatul CA-ului pentru a valida semnătura digitală a acestuia pe serverul în discuţie. Dacă informaţia în certificatul serverului s-a modificat de la momentul în care a fost semnat de CA sau dacă cheia publică din certificatul CA-ului nu corespunde cu cheia publică folosită pentru semnarea certificatului serverului, clientul nu va autentifica identitatea serverului. Dacă semnătura digitală a CA-ului se poate verifica, serverul consideră certificatul utilizatorului ca o „scrisoare de introducere” validă de la CA şi continuă. În acest punct clientul a determinat că certificatul serverului este valid.

- **Numele de domeniu din certificatul serverului se potriveşte cu numele serverului în sine?** Acest pas confirmă că serverul este localizat la adresa de rețea specificată de numele de domeniu în certificatul serverului. Deşi cest pas nu face parte – în mod tehnic – din protocolul SSL, acesta oferă singura protecție împotriva atacului „omul din mijloc". Clienții trebuie să efectueze acest pas şi să refuze să autentifice un server dacă numele de domeniu nu se potrivesc.
- **Serverul este autentificat.** Clientul continuă cu protocolul SSL handshake. Dacă pentru orice motiv nu se ajunge până în acest punct, utilizatorul este informat că nu se poate crea o conexiune autentificată şi codificată. Dacă serverul cere autentificarea clientului, acesta va efectua paşii specificați în paragraful următor.

Autentificarea clientului

Serverele cu capabilități SSL pot fi configurate să ceară la rândul lor autentificarea clientului sau validarea criptografică a identității acestuia. Când serverul astfel configurat cere autentificarea clientului (pasul 6), clientul trimite serverului atât certificatul cât şi o informație separată semnată digital pentru identificarea sa. Serverul utilizează datele semnate pentru validarea cheii publice din certificat şi pentru autentificarea identității pretinse de certificat.

Protocolul SSL cere clientului să creeze o semnătură digitală dintr-o dispersie cu sens unic aplicată unor date aleatoare generate în timpul handshake-ului şi cunoscute doar de client şi de server. Dispersia datelor este apoi codificată cu cheia privată care corespunde cheii publice din certificatul prezentat serverului.

Pentru a autentifica legătura dintre cheia publică şi persoana sau altă entitate identificată de certificatul conține cheia, serverul trebuie să primească răspunsul „da" la primele patru întrebări de mai jos.

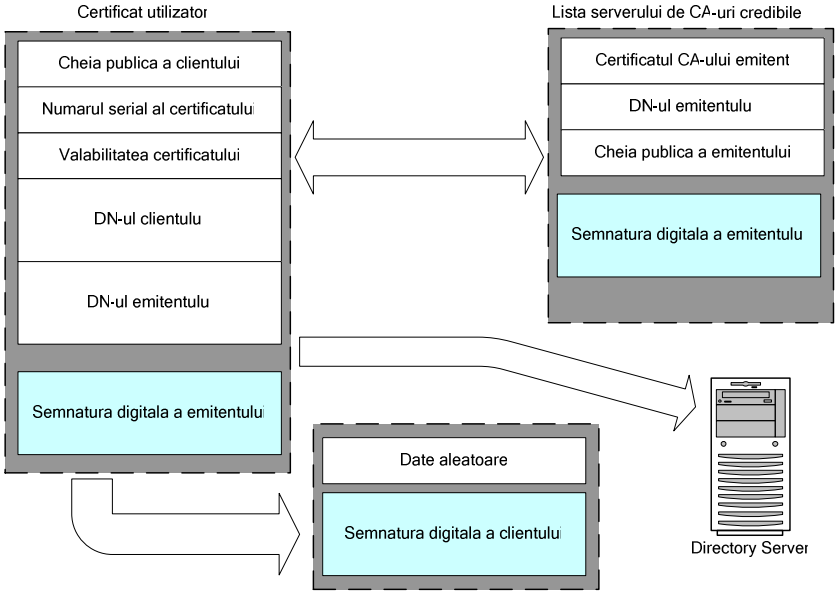


Fig. 10 Modalitatea de autentificare a certificatului unui client

Serverul parcurge următorii paşi pentru autentificarea identității clientului:

1. **Cheia publică a utilizatorului validează semnătura sa digitală?** Serverul verifică dacă semnătura digitală a utilizatorului poate fi validată

cu cheia publică a certificatului. Dacă este aşa, serverul a stabilit că cheia publică a utilizatorului în cauză se potriveşte cu cheia privată folosită la crearea semnăturii şi datele nu au fost modificate de la momentul semnăturii.

2. **Data de astăzi este în perioada de valabilitate?** Serverul verifică perioada de validitate a certificatului clientului. Dacă data şi timpul curent sunt în afara perioadei de valabilitate, procesul de autentificare nu mai continuă, altfel se trece la pasul 3.
3. **CA-ul emitent este credibil?** Fiecare sever menține o listă de CA-uri credibile, reprezentată în partea dreaptă a figurii 19. Această listă determină care dintre certificate vor fi acceptate de server. Dacă numele (DN – distinguished name) al CA-ului emitent se potriveşte cu numele unui CA de pe listă, răspunsul la întrebare este „da" şi se trece la pasul 4. Altfel, clientul nu este autentificat până când clientul nu verifică faptul că CA-ul se află într-un lanț credibil, aflat pe listă. Administratorii pot controla care CA-uri sunt incluse pe listă.
4. **Cheia publică a CA-ului emitent verifică semnătura digitală a emitentului?** Serverul utilizează cheia publică din certificatul CA-ului pentru a valida semnătura digitală a acestuia. Dacă informația din certificat s-a modificat de la momentul în care a fost semnat de CA sau dacă cheia publică din certificatul CA-ului nu corespunde cu cheia publică folosită pentru semnarea certificatului, serverul nu va autentifica identitatea clientului. Dacă semnătura digitală a CA-ului se poate verifica, serverul consideră certificatul utilizatorului ca o „scrisoare de introducere" validă de la CA şi continuă. În acest punct serverul a determinat că certificatul clientului este valid.
5. **Clientul autentificat are acces la resursele solicitate?** Serverul verifică ce resurse poate accesa clientul în conformitate cu listele de control al accesului (ACL) şi stabileşte o conexiune cu drepturile potrivite. Dacă serverul nu ajunge la pasul 5 pentru orice motiv, utilizatorul identificat de certificat nu poate fi autentificat şi utilizatorului nu i se permite acces la resurse.

A.4. Protocoale criptografice

Schimbul de chei, autentificarea, criptarea şi algoritmii MAC sunt determinate de suita de cifruri selectate în mesajul *hello* de la server. În cadrul SSL se folosesc două tipuri de sisteme de criptografie şi anume simetrică şi asimetrică. Înainte însă de a le prezenta, ne vom referi mai întâi la protocolul Handshake, care se află la baza SSL.

A.4.1. Protocolul Handshake

Parametrii criptografici ai sesiunii sunt produşi de protocolul SSL Handshake, care operează deasupra stratului SSL Record. Când un client şi un server SSL comunică pentru prima dată, aceştia convin asupra unei versiuni a protocolului, selectează algoritmii criptografici, opțional se autentifică reciproc şi utilizează tehnici de criptare cu chei publice pentru a genera secrete. Aceste procese se realizează în cadrul protocolului Handshake, care se rezumă după cum urmează:

Clientul trimite un mesaj **client hello** la care serverul trebuie să răspundă cu un mesaj **server hello**, altfel survine o eroare fatală şi conexiunea eşuează. Valorile din aceste două mesaje sunt folosite pentru stabilirea caracteristicilor de securitate între client şi server, şi anume atributele: versiunea de protocol, identificatorul de sesiune, suita de

cifruri şi metoda de compresie. În plus se generează şi se interschimbă două valori**:** **ClientHello.random** şi **ServerHello.random**. Ca răspuns la mesajele hello, serverul va trimite certificatul său dacă urmează a fi autentificat. În plus, se poate trimite un mesaj de schimb ale cheilor dacă este necesar (spre exemplu dacă serverul nu are certificat sau certificatul este numai pentru semnătură). Dacă serverul este autentificat, acesta poate cere un certificat de la client, dacă suita de cifruri aleasă cere acest lucru. Acum serverul va trimite un mesaj **server hello done** care indică faptul că această fază a protocolului s-a încheiat. Serverul aşteaptă acum un răspuns de la client. Dacă serverul a emis un mesaj de cerere a certificatului, clientul trebuie să trimită fie un mesaj cu certificatul său fie o alertă de lipsă a acestuia. Acum se trimite mesajul de schimb al cheilor de către client, conținutul acestuia depinzând de algoritmul cu chei publice selectat. Dacă clientul a trimis un certificat cu capacități de semnare, se trimite un mesaj de verificare semnat digital.

În acest moment clientul trimite un mesaj pentru schimbarea specificațiilor criptografice, clientul copiind specificațiile în aşteptare peste cele curente. Imediat după aceea, clientul trimite mesajul de terminare codificat cu algoritmul selectat. Ca răspuns, serverul va trimite propriul mesaj de schimbare a specificațiilor criptografice şi va trimite mesajul de terminare codificat cu algoritmul selectat de specificații. În acest moment, strângerea de mână este completă, iar clientul şi serverul pot începe schimbul securizat de date la nivel de aplicație. Întregul proces este descris schematic în figura 20:

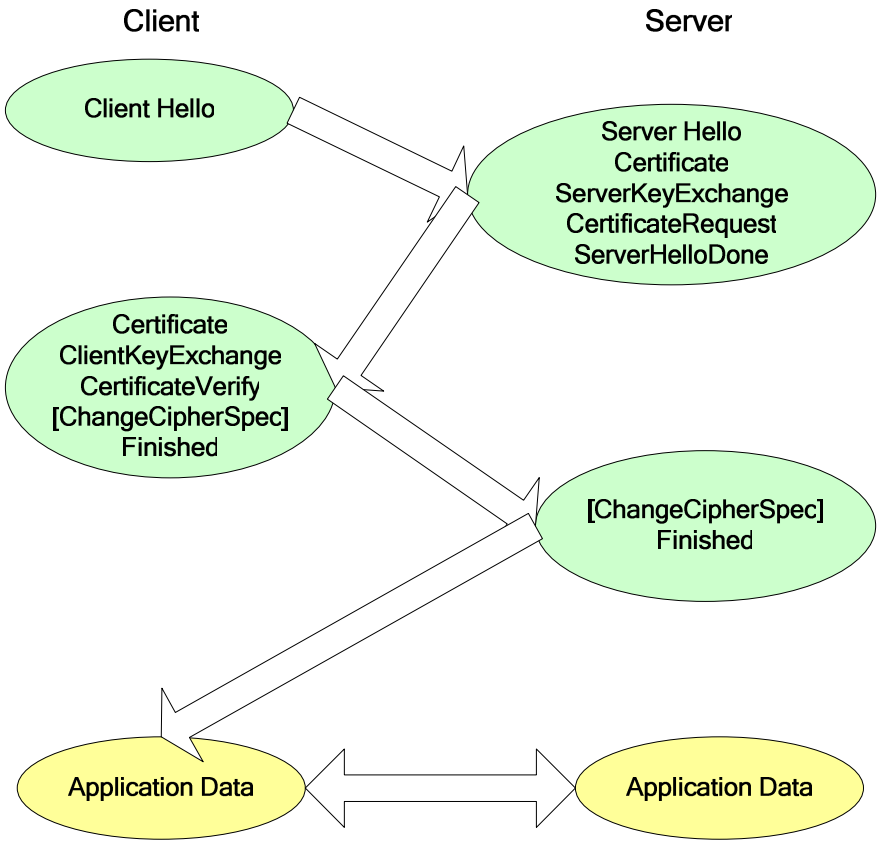


Fig. 11 Schimbul de mesaje în protocolul SSL Handshake

Când clientul şi serverul decid să reia o sesiune anterioară sau doresc să duplice o sesiune existentă (în loc să negocieze noi parametri de securitate), mesajele schimbate sunt următoarele:

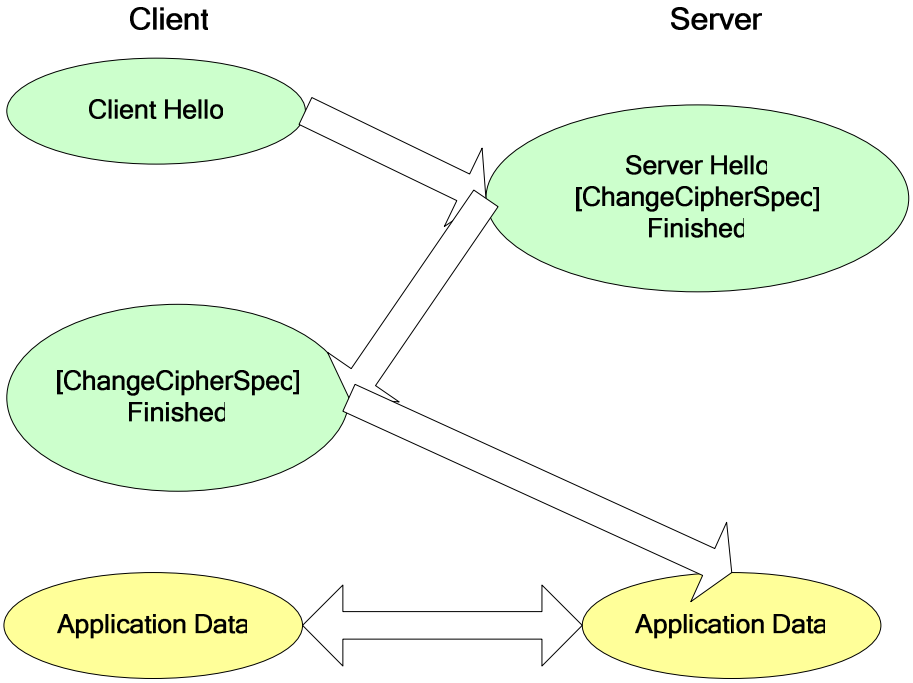


Fig. 12 Reluarea unei sesiuni SSL

Clientul trimite un mesaj ClientHello cu identificatorul sesiunii de reluat. Serverul verifică cache-ul iar dacă identificatorul este găsit şi se doreşte reluarea sesiunii, se trimite înapoi mesajul ServerHello cu acelaşi identificator de sesiune. În acest punct, atât clientul cât şi serverul trebuie să-şi trimită mesaje de specificare a cifrurilor şi continuă direct cu mesajele de terminare. Odată restabilirea încheiată, clientul şi serverul pot schimba date la nivelul de aplicații. Dacă identificatorul de sesiune nu este găsit în cache, serverul generează un nou identificator de sesiune, iar clientul şi serverul efectuează un handshake complet. Conținutul şi semnificația fiecărui mesaj se găseşte în [FREI96].

A.4.2. Criptografie asimetrică

Algoritmii asimetrici sunt utilizați în cadrul protocolului Handshake pentru autentificarea părților şi pentru generarea cheilor şi secretelor.

Pentru Diffie-Hellman, RSA şi FORTEZZA se foloseşte acelaşi algoritm pentru a converti secretul premaster în secretul master. Primul ar trebui eliminat din memorie odată ce secretul master a fost calculat.

```
      master_secret =
      MD5(pre_master_secret + SHA('A' + pre_master_secret +
ClientHello.random + ServerHello.random)) +
      MD5(pre_master_secret + SHA('BB' + pre_master_secret +
ClientHello.random + ServerHello.random)) +
      MD5(pre_master_secret + SHA('CCC' + pre_master_secret +
ClientHello.random + ServerHello.random));
```

RSA

Când se foloseşte RSA pentru autentificarea serverului şi schimbul de chei, clientul generează un secret premaster de 48 de octeți, îl codifică cu cheia publică a

serverului şi îl trimite acestuia. La recepție, serverul face uz de cheia sa privată şi decodifică secretul, apoi ambele părți calculează secretul master, aşa cum s-a arătat mai sus.

Diffie-Hellman

Se efectuează un calcul Diffie-Hellman convențional. Cheia negociată Z se foloseşte ca secret premaster şi se calculează secretul master ca mai sus. De remarcat că parametrii Diffie-Hellman sunt fie conținuți în certificatul serverului fie generați pe loc.

FORTEZZA

Se generează un secret de 48 de octeți care se trimite codificat cu TEK serverului. Acesta decodifică secretul şi îl converteşte în secret master, ca mai sus. Cheile de codificare se generează de către client şi schimbate în cadrul mesajelor de schimb; secretul master se foloseşte numai pentru calcule MAC.

A.4.3. Criptografie simetrică

Tehnica utilizată pentru criptarea şi verificarea integrității înregistrărilor SSL este stabilită de specificația criptografică curentă (CipherSpec). Un exemplu tipic ar fi criptarea datelor cu DES şi generarea codurilor de autentificare cu MD5. Algoritmii de codificare şi algoritmii MAC sunt poziționați pe SSL_NULL_WITH_NULL_NULL la începutul protocolului SSL Handshake, indicând că nu se efectuează criptare sau verificare de date. Protocolul handshake se foloseşte pentru a negocia un set de protocoale mai sigure şi pentru a genera chei criptografice.

Secretul master

Înainte ca verificarea şi criptarea să se poată efectua asupra înregistrărilor, clientul şi serverul trebuie să genereze o informație secretă cunoscută numai de aceştia. Această valoare este o cantitate de 48 de octeți numită secretul master. Acesta este utilizat pentru generarea cheilor şi secretelor pentru criptare şi calcule MAC. Unii algoritmi, cum ar fi FORTEZZA, pot avea propriile proceduri de generare a cheilor, în acest caz secretul master folosindu-se doar pentru calcule MAC.

Convertirea secretului master în chei şi secrete MAC

Secretul master este dispersat într-o secvență de octeți securizați atribuiți secretelor şi cheilor MAC, în conformitate cu specificațiile în vigoare (CipherSpec). Specificația necesită:

1. Secret MAC de scriere al clientului
2. Secret MAC de scriere al serverului
3. Cheie de scriere a clientului
4. Cheie de scriere a serverului
5. Vector de inițializare (IV[1]) al clientului
6. Vector de inițializare (IV) al serverului

Toate acestea sunt generate din secretul master, în această ordine. Cheile nefolosite sunt goale, cum este cazul cheilor FORTEZZA comunicate în mesajul KeyExchange.

Următoarele intrări sunt disponibile procesului de definire a cheilor:

1. opaque MasterSecret[48]

[1] IV – Initialization Vector

2. ClientHello.random
3. ServerHello.random

Când se generează chei şi secrete MAC, secretul master este folosit ca sursă de entropie, iar valorile aleatoare furnizează material necodificat numit şi „sare”.

Pentru a genera material pentru chei, se calculează entitatea:

```
key_block =
      MD5(master_secret + SHA('A' + master_secret + ServerHello.random +
ClientHello.random)) +
      MD5(master_secret + SHA('BB' + master_secret + ServerHello.random
+ ClientHello.random)) +
      MD5(master_secret + SHA('CCC' + master_secret + ServerHello.random
+ ClientHello.random)) + [...];
```

până când datele calculate sunt suficiente. Entitatea key_block se partiționează după cum urmează:

```
client_write_MAC_secret[CipherSpec.hash_size]
server_write_MAC_secret[CipherSpec.hash_size]
client_write_key[CipherSpec.key_material]
server_write_key[CipherSpec.key_material]
client_write_IV[CipherSpec.IV_size] /* non-export ciphers */
server_write_IV[CipherSpec.IV_size] /* non-export ciphers */
```

Orice valori generate în plus se ignoră.

Algoritmii exportabili (pentru care *CipherSpec.is_exportable* este adevărată) necesită procesare suplimentară pentru derivarea cheilor finale:

```
final_client_write_key =
      MD5(client_write_key + ClientHello.random + ServerHello.random);
final_server_write_key =
      MD5(server_write_key + ServerHello.random + ClientHello.random);
```

Algoritmii de criptare exportabili îşi derivă vectorii de inițializare din mesajele aleatoare:

```
client_write_IV = MD5(ClientHello.random + ServerHello.random);
server_write_IV = MD5(ServerHello.random + ClientHello.random);
```

Ieşirile MD5 sunt ajustate la dimensiunile corespunzătoare prin eliminarea celor mai nesemnificativi octeți.

A.5. Atacuri rezolvate în SSL v3

În versiunea 2 a SSL, handshake-ul inițial nu este protejat, în aşa fel încât un atacator activ poate elimina din cererea inițială cifrurile cu criptare puternică, ceea ce cauzează alegerea unei criptări mai slabe între părțile ce comunică. În versiunea 3 atacul a fost ameliorat prin adăugarea unui mesaj de final la sfârşitul handshake-ului în care fiecare parte trimite un rezumat al mesajelor din handshake.

Versiunea 3 a SSL a adăugat un mesaj de terminare care să indice că nu mai există date de transferat. Versiunea 2 se bazează pe închiderea conexiunii de către protocolul TCP, dar acesta nu este protejat criptografic şi un atacator ar putea trimite un mesaj fals care să cauzeze închiderea conexiunii, fără ca aplicația să îşi dea seama că aceasta a fost închisă anormal.

A.6. SSL vs. TLS

Aşa cum am văzut la începutul acestui capitol, TLS a fost creat de către IETF ca răspuns la multitudinea de protocoale incompatibile care serveau aceluiaşi scop şi anume comunicarea securizată prin rețea. Modificările aduse de TLS nu sunt substanțiale şi multă lume îl priveşte mai degrabă ca o actualizare minoră a SSL. Diferențele între cele două protocoale sunt rezumate în cele ce urmează:

Schimbări diverse

- **Numărul de versiune:** pentru TLS, numărul de versiune este 3.1, față de 3.0 pentru SSL, ceea ce întăreşte convingerea unora că TLS este doar o actualizare minoră a SSL.
- **Cifruri:** TLS nu are suport pentru schimbul de chei Fortezza şi pentru criptarea Fortezza.
- **Padding:** TLS permite padding cu un număr variabil de octeți (maxim 255)
- **MAC:** TLS foloseşte ultima versiune de HMAC, iar acesta acoperă şi câmpul de versiune.
- **Mesajul CERTIFICATE_VERIFY:** Funcția de dispersie este calculată doar după mesajele de handshake
- **Coduri de avertizare:** TLS defineşte mai multe coduri de avertizare decât SSL.