CARNet
HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Rainbows tablice

CCERT-PUBDOC-2008-08-237

+CERT.hr u suradnji s LSS

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi.

Rezultat toga rada je i ovaj dokument, koji je nastao suradnjom CARNet CERT–a i LS&S–a, a za koji se nadamo se da će Vam koristiti u poboljšanju sigurnosti Vašeg sustava.

CARNet CERT, www.cert.hr

Nacionalno središte za **sigurnost računalnih mreža** i sustava.

LS&S, www.LSS.hr

Laboratorij za sustave i signale pri Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Sadržaj

1. Uvod

U današnje vrijeme većina poslužitelja su svakodnevno dostupni korisnicima što zahtjeva veliku razinu sigurnosti i zaštite. Jedan od najboljih načina zaštite je zahtjev za autentikacijom korisnika prilikom pristupa sustavu. Upravo zbog toga veliku pažnju treba posvetiti zaštiti podataka koji sadrže korisnička imena i zaporke. Iako većina sustava koristi neki od brojnih načina šifriranja podataka prilikom spremanja ili prijenosa (SHA-512, SHA-1, MD5) postoje brojne metode njihovog razbijanja.

Razbijanje zaporki predstavlja proces obnavljanja zaporke iz šifriranog zapisa koji napadač prethodno treba saznati (npr. provalom u sustav). Metode razbijanja mogu zasnovane na nasumičnom pogađanju zaporki kao što su *brute force* ili *dictionary* napadi. Najosnovniji oblik napada - *brute-force* koji isprobava sve kombinacije dok ne pronađe odgovarajuću, spor je i neefikasan ukoliko je zaporka preduga. Stoga je razvijena ideja da se jednom izračunate *hash* vrijednosti zaporki pohrane kako se ne bi morale svaki put nanovo izračunavati. U tu svrhu razvijeno je korištenje tzv. *rainbows* tablica koje se sastoje od unaprijed izračunatih lista *hash* vrijednosti „svih“ zaporki pa se uz poznavanje *hash* vrijednosti pretragom tablica može otkriti odgovarajuća zaporka. Pri tome proces otkrivanja zaporki može trajati samo nekoliko sekundi ili minuta dok proces generiranja *rainbows* tablica može trajati i godinama. *Rainbows* tablice temelje se na *Time-Memory Trade-Off* principu koji je pobliže opisan u nastavku dokumenta. Osnovno obilježje ovog principa je smanjenje vremena izvođenja algoritma razbijanja na „štetu“ memorije.

U ovom dokumentu navedene su osnove o zaporkama kao i osnovni oblici njihovog “razbijanja“. Detaljno su opisane *rainbows* tablica kao i način otkrivanja zaporki temeljen na njima, kao i osnove *Time-Memory Trade-Off* principa.

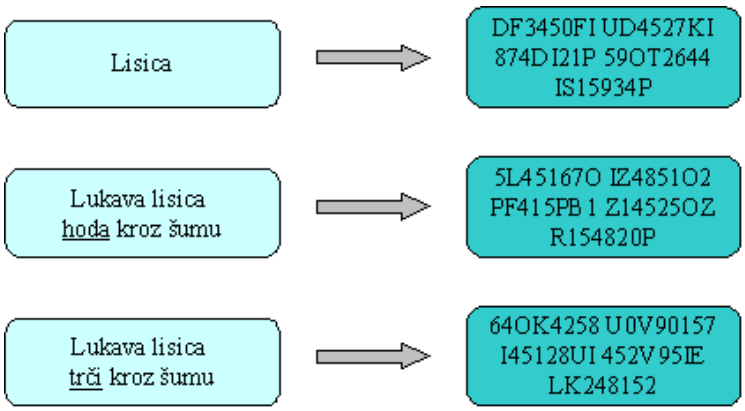
2. Razbijanje zaporki

Razbijanje zaporki je proces obnavljanja zaporke iz podataka koji su pohranjeni ili prenošeni računalnim sustavom. Osnovni pristup je višestruko pokušavanje pogađanja same zaporke. Korisnik ga može iskoristiti u slučaju zaboravljanja zaporke i neovlaštenog pristupa sustavu. Administratori sustava mogu ga koristiti kao preventivnu mjeru brojanjem neuspješnih pokušaja prijave na sustav te se takav sustav privremeno isključiti kako bi onemogućili izvođenje napada. Legitimnim korisnicima može se dozvoliti određen broj pokušaja prijave (3, 5, 8, itd.), dok prelazak tog broja označava zlonamjernog korisnika. Neki od ostalih načina razbijanja zaporki su: *brute force* ili *dictionary* napad, prebrojavanje, razbijanje *salting* mehanizma i sl.

Najčešće korištene funkcije sažimanja (eng. *hash*) mogu se provesti vrlo brzo i napadač može testirati zaporke više puta dok ne otkrije odgovarajuću. Zaporke za pristup računalnim sustavima pohranjene su, obično u obliku šifriranog teksta (eng. *cleartext*), u bazu podataka ili datoteku tako da sustav može obavljati provjeru zaporke kada se korisnici prijavljuju na sustav. U cilju očuvanja tajnosti zaporke, podaci za provjeru zaporke se obično generiraju primjenom jednosmjernih funkcija (eng. *one-way function*), eventualno u kombinaciji s drugim podacima. Iako se mislilo da su jednosmjerne funkcije za spremanje zaporki u *hash* obliku kriptografski sigurne, posjed takve šifrirane zaporke pruža razne mogućnosti neovlaštenog razotkrivanja.

2.1. Hash funkcija

Kriptografska funkcija (funkcija sažimanja, eng. *hash*) je transformacija koja ima niz znakova (tekst, zaporka) kao ulaz te kao izlaz daje niz fiksne duljine koji se naziva *hash* vrijednost. *Hash* vrijednost je prikaz poruke ili dokumenta iz kojeg je izračunata. Čak i male promjene u izvornom ulaznom nizu drastično mijenjaju izlaz funkcije, što je poznato kao Avalancheov efekt. Slika 1 prikazuje promjene u izračunatim *hash* vrijednostima kada se izmjeni samo jedna riječ niza.



Slika 1. *Hash* funkcija

Kriptografske *hash* funkcije se koriste za provjeru integriteta poruka i digitalnih potpisa u aplikacijama za informacijsku sigurnost (autentikacija). *Hash* vrijednost je vrsta "potpisa" za tok podataka koja predstavlja sadržaj.

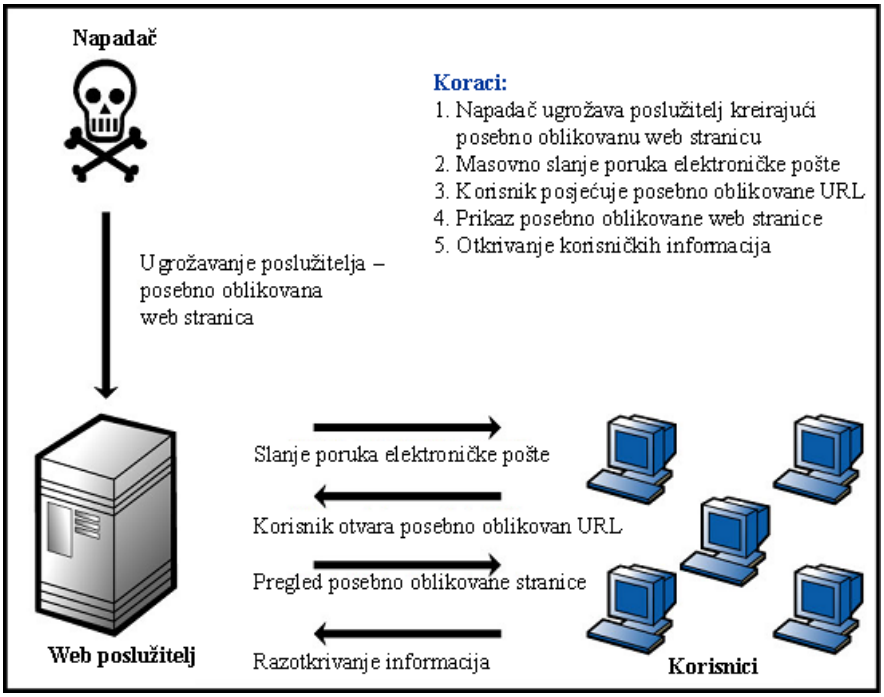
Važno svojstvo *hash* funkcije je to što različiti ulazni nizovi mogu generirati iste *hash* vrijednosti (sažetke). Zbog toga se prilikom razbijanja zaporke ne treba pronaći prava zaporka već bilo koji niz koji daje istu *hash* vrijednost kao i prava zaporka. To uvelike olakšava sam proces razbijanja.

2.2. Način otkrivanja zaporki

Pojam razbijanje zaporki je obično ograničen na otkrivanje jedne ili više zaporki pomoću kojih se dobiva poznata *hash* vrijednost. Postoje mnogi načini dobivanja zaporki, zbog čega napadač može i bez *hash* niza pokušati pristupiti na računalni sustav (npr. pogađanjem same zaporke). Jedan od sustava zaštite je dobar dizajn sustava koji ograničava broj pokušaja neuspješnog pristupa i obavještava administratora (kako bi takav sustav, ako je potrebno, privremeno isključio). Ipak, poduzimanje velikih mjera zaštite ponekad nisu dovoljne da bi se sustav obranio od napada.

Zaporke je moguće pokušati nabaviti različitim metodama kao što su:

- **tehnike socijalnog inženjeringa** (eng. *social engineering*) - problem korištenja raznih komunikacijskih vještina u uvjeravanju korisnika da postupe na određeni način,
- ***wiretapping*** - praćenje telefonske i Internet konverzacije od treće strane (a da sudionici komunikacije nisu toga svjesni),
- **prikriveno snimanje unosa na tipkovnici** (eng. *keystroke logging*) - hvatanje i snimanje pritisaka tipki prilikom prijave na sustav,
- ***login spoofing*** - korisnik upisuje korisničko ime i zaporku u posebno oblikovan program te tako napadaču otkriva osjetljive informacije,
- ***dumpster diving*** - pronalaženje korisnih informacija filtriranjem materijala koje je korisnik odbacio u smeće (najčešće *post-it* poruke s korisničkim imenima i zaporkama),
- ***phishing*** - usmjeravanje korisnika na unos detalja na web stranicama putem poruka elektroničke pošte ili izravnom razmjenom poruka (primjer je prikazan na slici 2),
- ***shoulder surfing*** - dobivanje informacija izravnim motrenjem („gledanje preko ramena"),
- ***timing attack*** - ugrožavanje sustava analizom vremena potrebnog za izvršavanje kriptografskog algoritma,
- **korištenje trojanskog konja ili virusa** - korištenje zlonamjernog programa da bi se zavarao korisnik (program korisniku ne izgleda zlonamjerno),
- **kompromitiranje sigurnosti poslužitelja**.



Slika 2. *Phishing* napad

2.3. Osnovne metode napada

Postoje brojne metode pomoću kojih je moguće razbiti zaporke, a neke od osnovnih metoda su:

- pogađanje zaporki,
- *brute force* napad,
- *dictionary* napad,
- prebrojavanje.

U nastavku je dan kratki opis svake od navedenih metoda.

2.3.1. Pogađanje zaporki

Mnoge zaporke mogu pogoditi, bilo ljudi, bilo sofisticirani programi razbijanja zaporki opremljeni s rječnicima i korisnikovim osobnim podacima. Općenito korištenje programa uvelike smanjuje vrijeme potrebno za pogađanje zaporke. Ipak u slučajevima kada napadač poznaje korisnika pogađanje zaporke može biti jednostavnije nego korištenje nekog od programa. Mnogi korisnici odabiru slabe zaporke, obično na neki način povezane sa sobom što olakšava napadaču pogađanje zaporke. Na primjer, ako korisnik odabere zaporku IVANPMF95 zato što se zove Ivan i diplomirao je na Prirodoslovno-Matematičkom fakultetu 1995.g., time omogućuje svim kolegama i prijateljima lakše pogađanje zaporke.

Jedan od većih problema za administratore predstavljaju predvidljivo postavljene zaporke koje programi lako razbijaju. Do toga dolazi jer velik broj korisnika za zaporke odabire neku riječ iz rječnika, ime poznate osobe, riječi poput „password“ ili „admin“, samo korisničko ime i sl. budući da im je sigurnije nizove teže pamtiti.

Primjeri nesigurnih izbora zaporki uključuju:

- *blank* (zaporka bez znakova),
- riječi „password“, „passcode“, „admin“ i sl.,
- samo korisničko ime (zaporka = korisničko ime),
- ime druge fizičke osobe (obično na neki način povezane s korisnikom),
- mjesto ili datum rođenja korisnika, prijatelja ili rodbine,
- ime ljubimaca,
- riječi iz rječnika bilo kojeg jezika,
- imena slavnih osoba,
- broj sa registarske pločice vozila,
- niz slova po standardnom rasporedu tipkovnice (npr. „123456“, „qwerty“, „asdf“,…),
- jednostavna modifikacija prethodno navedenog primjera, kao što je zamjena mjesta slova, itd.

Dodatne probleme u očuvanju privatnosti donosi nemarnost korisnika. Neki korisnici zanemare potrebu za promjenom zaporke koju inicijalno dobiju za računalni sustav. I neki administratori zanemare potrebu za promjenom zaporke koju pružaju operacijski sustavi ili sami dobavljači. Popisi osnovnih (eng. *default*) zaporki dostupni su na mnogim mjestima na Internetu (npr. http://www.phenoelit-us.org/dpl/dpl.html).

2.3.2. *Brute force* napad

Jedan od načina razbijanja je isprobavanje svake moguće zaporke, i taj je napad poznat kao *brute force* napad. U teoriji, *brute force* napad će uvijek biti uspješan jer pravila za prihvatljive zaporke

moraju biti javno poznata, ali kako se dužina zaporke povećava, raste i broj mogućih varijacija što eksponencijalno produžuje vrijeme provođenja napada. Zbog toga je ova metoda vrlo nepraktična osim ako je zaporka relativno mala (do 6-7 znakova).

Postoje dvije vrste ovog napada:

1. ***offline* napad** - napadač ima pristup *hash* zapisu zaporke. Općenito ga je puno lakše izvesti, jer se testiranje zaporki svodi na brzo obračunavanje matematičkog računanja, tj. računanje *hash* vrijednosti za sve moguće kombinacije znakova dok se ne pronađe odgovarajući niz znakova. Prilikom izvođenja ovog napada mogu se iskoristiti različiti crvi koji automatski preuzimaju *hash* vrijednosti zaporki iz zaraženih računala.
2. ***online* napad** - napadač nema pristup *hash* vrijednosti zaporke. Cilj napadača je pokušati dokazati svoju autentičnost sa svim mogućim zaporkama, pri čemu sustav može nametnuti proizvoljna pravila, a pokušaji autentikacije mogu biti prijavljeni. Temelji se na otvaranju autentikacijskih sjednica pa je najčešće vrlo sporo jer sustavi uglavnom postavljaju vremensko čekanje za potvrdu uspješne ili neuspješne prijave.

Općenito, tehnike pretrage *brute force* napadom mogu se koristiti kako bi se ubrzalo prebrojavanje. Stvarna prijetnja dolazi od *smart force* tehnika koje iskorištavaju znanje o tendenciji ljudi da odabiru zaporke.

Kako ovaj napad jednostavno pokušava koristiti svaku moguću kombinaciju znakova kao zaporku, da bi se otkrila zaporka od jednog malog slova potrebno je isprobati 26 kombinacija (od 'a' do 'z'). Zaporke koje se sastoje od dva mala slova zahtijevaju 26 * 26 = 676 kombinacija. Broj mogućih kombinacija (i vrijeme potrebno za razbijanje zaporki) raste eksponencijalno povećanjem duljine zaporke te uvođenjem drugih oblika znakova.

Vrijeme probijanja zaporke ovisi o vrsti znakova koje se pojavljuju unutar njih, što znači da zaporke koje sadrže i velika i mala slova (eng. *case sensitive*) zahtijevaju više vremena za razbijanje. Potrebno vrijeme za probijanje zaporki može se smanjiti korištenjem bržih računala pri čemu jedinu ulogu igra CPU brzina (količina RAM-a, tvrdi disk ne utječu na brzinu *brute force* napada). Također, korištenje više računala paralelno smanjuje vrijeme potrebno za probijanje zaporki.

Tablica 1 prikazuje vrijeme potrebno za *brute force* napad ovisno o dužini zaporke i korištenom skupu znakova. Izvođenje operacija pretpostavljeno je na računalu koje obavlja 500 000 operacija u sekundi. Dobiveno vrijeme je vrijeme za najgori mogući slučaj kada se pretpostavi da je posljednja kombinacija tražena zaporka. Vrijeme potrebno za ostale kombinacije duljina i složenosti zaporki te broja i jačine računala moguće je izračunati preko aplikacija na Internetu (http://lastbit.com/pswcalc.asp).

Dužina	Znakovi u zaporci				Broj računala
	Mala slova	Mala slova i znamenke	Mala i velika slova	Svi ASCII znakovi	
<= 4	Trenutno	trenutno	trenutno	3 min	1
7	4 h	43 h	23 dana	4 god	
9	4 mj	6 god	178 god	44530 god	
<= 4	Trenutno	trenutno	trenutno	2 min	2
7	134 min	22 h	12 dana	29 mj	
9	63 dana	4 god	90 god	22266 god	

Tablica 1. Vrijeme potrebno za razbijanje zaporki *brute force* napadom

Traženo vrijeme razbijanja zaporki može se dobiti preko formule: $((C^{L}) / S) / N$, gdje je C duljina skupa znakova, L je dužina zaporke, S je broj provjerenih zaporki po sekundi, a N je broj računala koji se koristi.

2.3.3. *Dictionary* napad

Dictionary napad je tehnika za razbijanje zaporki zasnovano na isprobavanju zaporka iz prethodno pripremljenih rječnika. Kao i kod prethodno opisane metode moguća su dva načina provođenja napada – *offline* i *online*. *Offline dictionary* metoda testira riječi pohranjene u rječniku, koje prethodno propusti kroz osnovne funkcije šifriranja, te dobivene *hash* vrijednosti uspoređuje s poznatom *hash* vrijednošiu nepoznate zaporke. *Online* napad koristi riječi iz rječnika za prijavu na željeni sustav pri čemu je potrebno poznavati korisničko ime.

Ovaj oblik napada često biva uspješan, jer korisnici uz kratke zaporke najčešće odabiru zaporke koje se sastoje od jedne riječi iz rječnika ili neke jednostavne varijacije tih riječi (dodavanje znamenki). Vjerojatnost razbijanja zaporki može se povećati korištenjem većeg ili više rječnika (na više jezika). Obično koristi neki alat koji ima ugrađen rječnik i implementiran algoritam pretraživanja.

Pretragom Interneta moguće je pronaći razni rječnici pa se tako npr. hrvatski uz neke druge rječnike može pronaći na http://www.word-list.com/.

Jedna od poboljšanih inačica ovog napada je ***hybrid*** napad kod kojeg su dodani određeni simboli i brojevi u rječnik.

2.3.4. Prebrojavanje

U osnovnom obliku, prebrojavanje (eng. *precomputation*) uključuje propuštanje svake riječi u rječniku kroz *hash* funkciju i spremanje u obliku *riječ - šifrana riječ* (*plaintext - ciphertext*) parova na način koji omogućava pregled na šifrirana polja. Na taj način, kada se ispituje šifrirana zaporka (*hash* vrijednost), nepoznata zaporka se otkrije vrlo brzo, tj. nakon pretrage svih prethodno izračunatih parova.

U pravilu prebrojavanje je mnogo brže od *brute-force* napada, a pruža jednaku točnost. Tu su postignute veće brzine „na račun" memorije - *time-memory trade-off* princip koji je opisan u nastavku dokumenta.

3. *Time - Memory Trade - Off* princip

Time-memory trade-off je princip po kojemu se korištena memorija može smanjiti na „štetu“ dužeg vremena izvršavanja programa, ili obratno. *Brute force* razbijanje zaporki koristi male količine memorije, ali zbog toga je prosječno vrijeme razotkrivanja zaporki izrazito dugo. Najčešća situacija ubrzanja razbijanja zaporki jest korištenjem algoritma koji uključuje tablicu za potrebe pretraživanja (eng. *lookup table*), a koja u pravilu zahtjeva velik memorijski prostor. Taj oblik ubrzanja koristi se i kod *rainbows* tablica. Implementacija može uključiti cijelu tablicu, što smanjuje vrijeme računanja, ali povećava količinu potrebne memorije. Tablica se također može unositi prema potrebi, što povećava vrijeme računanja, ali smanjenje memorijske zahtjeve.

Primjenom *time-memory trade-off* principa prostor pretraživanja veličine *N* može biti pretvoren u bazu podataka u kojoj pretraživanje kriptiranih zaporki ima složenost $O(N^{2/3})$.

3.1. Osnovni parametri

Prilikom implementacije principa potrebno je voditi računa o parametrima koji definiraju određena ograničenja. Parametri se dijele u tri osnovne skupine:

1. konstantne vrijednosti:
   - nemoguće ih je mijenjati,
   - tu se ubrajaju:
     - brzina računanja *hash* funkcije,
     - brzina tvrdog diska,
     - prostor zaporki (eng. *keyspace*) N,
2. ulazni parametri:
   - korisnik ih može prilagoditi da bi se postigli odgovarajući izlazni podaci,
   - to su sljedeći parametri:
     - duljina lanca tablice,
     - broj nizova tablice,
     - broj tablica.
3. izlazni parametri:
   - vremenska i prostorna kompleksnost ovih parametara ovisi o složenosti ulaznih parametara,
   - tu se ubrajaju:
     - korištenje tvrdog diska (potreban diskovni prostor za pohranu tablica),
     - vjerojatnost uspjeha (pronalaženje nešifriranog teksta iz šifriranog),
     - srednje vrijeme kripto-analize (vrijeme potrebno za računanje *hash* funkcija),
     - maksimalno vrijeme kripto-analize,
     - srednje vrijeme pristupa disku (vrijeme potrebno za učitavanje datoteka),
     - maksimalno vrijeme pristupa disku,
     - vrijeme prebrojavanja (vrijeme potrebno za izračunavanje seta tablica).

3.2. Optimizacija procesa razbijanja zaporki

U ranim 80-im godinama 20. stoljeća Martin Hellman i Ronald Rivest predstavili su metodu poznatu kao *Time-Memory Trade-Off* tehniku. Tehniku je razvio Martin Hellman 1980. g. i njeno osnovno svojstvo je ubrzavanje kriptoanalize korištenjem unaprijed izračunatih vrijednosti pohranjenih u memoriji. Ali da bi

se pohranile baš sve kombinacije zaporki i njihovih *hash* vrijednosti (sažetaka) potrebne su vrlo velike količine memorije pa je stoga TMTO metoda modificirana 1982. g. od strane Rivesta. Umjesto da se pohranjuju sve kombinacije zaporki i *hash* vrijednosti pohranjuje se samo dio iz kojeg je moguće dobiti ostatak.

Modificirana TMTO metoda iz jedne zaporke generira velik broj (npr. deset tisuća) različitih *hash* vrijednosti koje predstavljaju druge zaporke. Prva dobivena *hash* vrijednost predstavlja ulaznu zaporku za daljnje računanje *hash* vrijednosti. Budući da takva vrijednost sadrži određene znakove koji nisu prikladni za ulazne znakove, prije daljnjeg računanja koristi se tzv. funkcija smanjivanja (redukcije) koja *hash* vrijednost pretvara u ASCII niz pogodan za zaporku. Za novodobivenu zaporku računa se *hash* vrijednost, a proces se ponavlja definirani broj puta. U tablicu se potom pohranjuju početna zaporka i krajnja zaporka ili *hash* vrijednost.

Iako funkcionalan, osnovni *Time-Memory Trade-Off* princip ima i svoje nedostatke. Naime, ukoliko se prilikom generiranja tablica za dvije različite početne zaporke $Z_{0'}$ i $Z_{0''}$ dobiju identične *hash* vrijednosti H_x u nekim fazama (npr. $H_{56'}=H_{94''}$) cijeli lanac iza jedne od tih vrijednosti ($H_{94''}$) bit će beskoristan jer je već pokriven drugim lancem – slučaj kolizije. Stoga se vrijeme generiranja nepotrebno produljuje, a i pretraga nije toliko učinkovita jer je memorijska potrošnja veća. Dodatan problem je i kada se u nekom lancu dobivaju iste *hash* vrijednosti (npr. $H_{17}=H_{37}=H_{57}=\ldots$) – tzv. *loop* ili neograničena petlja. Njihovo postojanje dodatno usporava generiranje i pretraživanje tablica.

Kako bi izbjegao spomenute probleme Philippe Oechslin je predložio korištenje različitih funkcija redukcije na različitim pozicijama u lancu. Stoga čak i kada se dobije ista *hash* vrijednost u istom ili različitim lancima, do kolizija ili petlji neće doći jer će se nad njima primijeniti različite funkcije redukcije. Kolizija je moguća samo u slučaju kada se dvije *hash* vrijednosti dobiju na istim pozicijama u lancu čime bi se nad njima provele iste funkcije redukcije, ali za pojavu takvih događaja vjerojatnost je zanemariva. Upravo zbog tog principa po kojem svaka funkcija redukcije predstavlja jedan oblik zrake duge (eng. *rainbow*) generirane tablice nazivaju se *rainbows* tablicama.

Prednosti Oechslinove metode (*rainbows* lanaca) u odnosu na Hellman-ovu metodu:

- broj pregleda tablica je smanjen,
- spajanja *rainbows* lanaca rezultiraju identičnim krajnjim točkama pa ih je moguće detektirati kao s istaknutim točkama,
- *rainbows* tablice nemaju petlje, jer se svaka funkcija redukcije pojavljuje samo jednom,
- lanci tablice imaju konstantnu duljinu što smanjuje broj lažnih alarma.

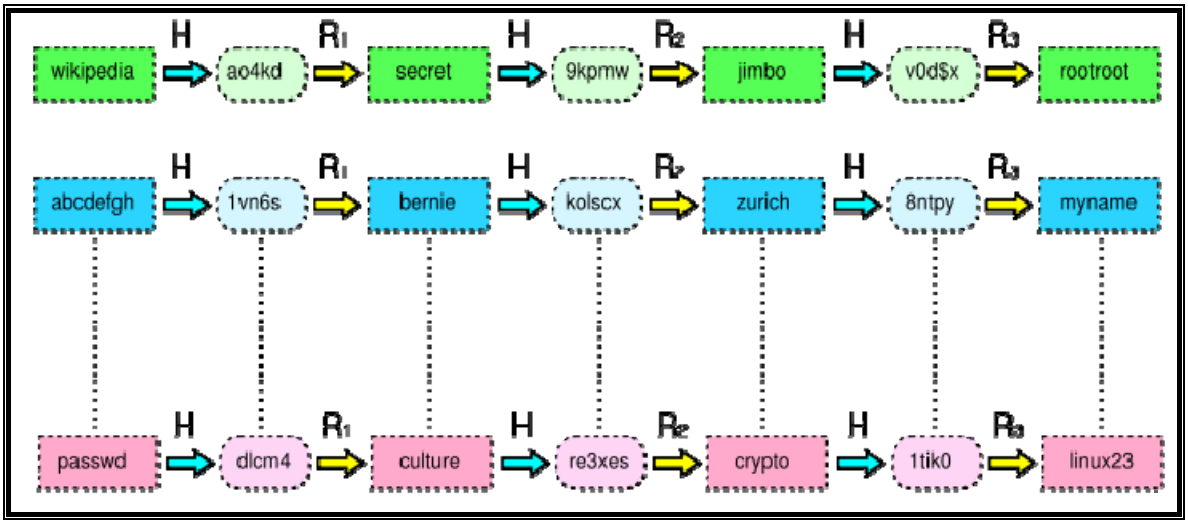
Razbijanje zaporki Oechslinovom metodom ubrzava sam proces do reda veličine 10 puta. To donosi veliku uštedu na vremenu potrebnom za razbijanje zaporki. Također, smanjuje se broj operacija potrebnih za računanje. Pri Oechslinovoj metodi broj potrebnih operacija je upola manji.

4. *Rainbows* tablice

4.1. Osnove

Rainbows tablice omogućuju ubrzavanje procesa razbijanja zaporki korištenjem *time-memory trade-of* principa koji se koristi prilikom otkrivanja nešifriranih (eng. *plaintext*) zaporki iz onih generiranih pomoću *hash* algoritma (kriptografskih *hash* funkcija).

Rainbow tablica je sažet prikaz povezanosti čistih sekvenci zaporki (lanac). Svaki lanac počinje s početnom zaporkom iz koje su izračunate sve ostale. Iz početne zaporke se pomoću *hash* funkcije izračunava *hash* vrijednost (sažetak). Dobivena *hash* vrijednost se zatim koristi u funkciji smanjivanja (redukcije) kako bi se *hash* vrijednost koja je u heksadekadskom nizu pretvorila u ASCII niz prikladan za predstavljanje zaporke. Proces se zatim ponavlja s fiksnim brojem iteracija i s različitim funkcijama redukcije u svakoj iteraciji. Broj iteracija je najčešće vrlo velik - npr. 10000, što znači da se u samo dva zapisa nalazi pohranjeno 10000 zaporki. Ta dva zapisa koji se pohranjuju u *rainbow* tablicu su početna i završna zaporka lanca. Opisani proces prikazan je na sljedećoj slici, ali u pojednostavljenom obliku sa samo 3 iteracije.

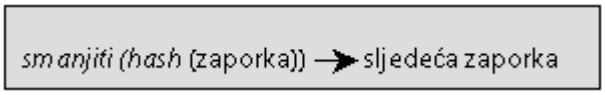


Slika 3. Primjeri nizova s funkcijama smanjivanja

Sadržaj tablice ne ovisi o ulaznim parametrima (*hash* vrijednosti nepoznatih zaporki) algoritma, jer se on jednom stvara i zatim višestruko koristi u nepromijenjenom obliku. Povećanje dužine lanca, tj. povećanje broja iteracija izračuna *hash* funkcije i redukcije smanjuje veličinu tablice. Ali to također povećava vrijeme potrebno za generiranje svakog lanca te za ispitivanje nepoznatih *hash* nizova. U jednostavnom slučaju kada postoji jedna stavka lanca pregled je vrlo brz, ali tablica je vrlo velika. Nakon produljenja lanaca usporava se pregled, ali smanjuje se veličina same tablice što donosi uštedu memorije.

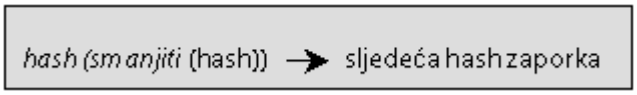
4.2. Način rada

Svaka tablica ovisi o *hash* funkciji i korištenim funkcijama smanjivanja, funkcijama koje zapisuju *hash* zaporke koristeći željeni niz znakova te o duljini zaporki. Lanac je niz zaporki koji počinje odabranom početnom zaporkom. Nakon odabira početne zaporke za dobivanje sljedeće zaporke treba provesti:



Nakon što je lanac s odgovarajućim brojem zaporki stvoren, konačna i početna zaporka pohranjuju se zajedno u *rainbow* tablicu.

Kako bi se našao inverz jedne *hash* zaporke potrebno je samo pretražiti tablicu. Ako u tablici ne postoji odgovarajući zapis za dobivanje sljedeće *hash* zaporke potrebno je provesti sljedeće:



Proces ponavljanja prestaje kada se zapis pronađe u tablici, a izvorna zaporka koja je započela lanac na čijem se kraju nalazi odgovarajući *hash*, koristi se za izradu svih ostalih zaporki. To znači da, kada je pronađena zaporka, napadač gotovo u trenu pronalazi i ostale koje posjeduje u *hash* obliku.

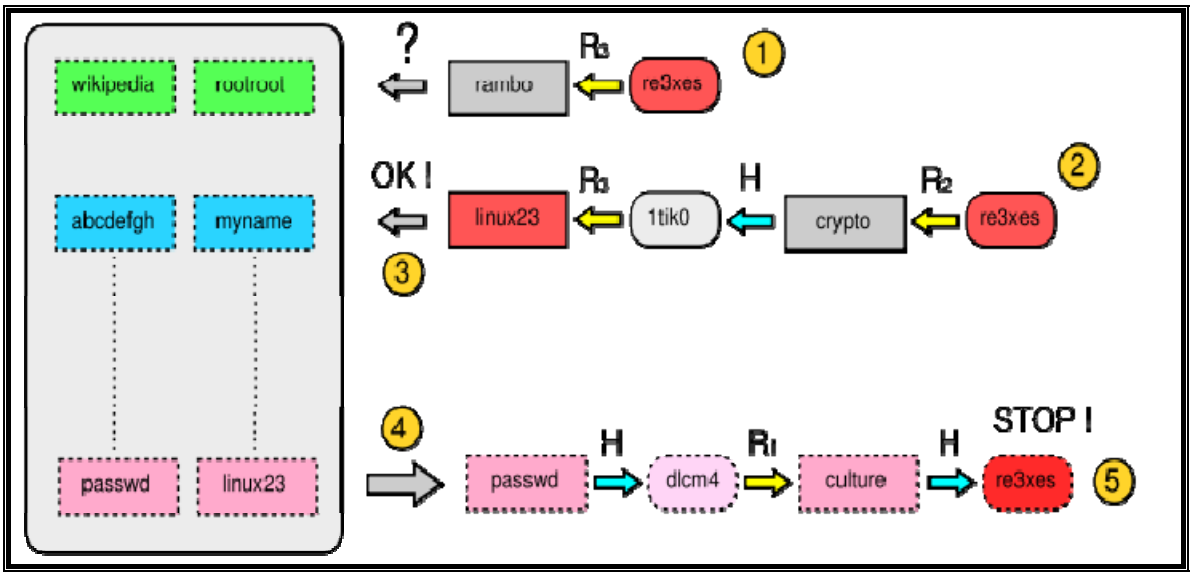
Rezultat procesa je tablica koja sadrži veliku vjerojatnost otkrivanja zaporke. Vjerojatnost uspjeha otkrivanja zaporke ovisi o parametrima koji se koriste za generiranje rezultata:

- korišteni skup znakova,
- duljina zaporke,
- duljina lanca,
- broj redaka tablice.

4.3. Primjer rada

Kad se iz poznate *hash* vrijednosti pokušava otkriti nepoznata zaporka prvo se *hash* vrijednost korištenjem zadnje funkcije redukcije (R_N) pretvori u ASCII niz te se taj niz potraži u *rainbow* tablici pri čemu se pretražuju završne zaporke. Ako se pronađe podudarnost tada je tražena zaporka ona koja je prethodila završnoj pa je stoga potrebno početi od početne zaporke koja je par pronađenoj završnoj zaporci. Potom je potrebno ponoviti prvih N-1 iteracija izračunavanja *hash* vrijednosti i primjene funkcija redukcije kako bi se došlo do tražene zaporke. A ako se pretraživani ASCII niz ne nalazi u *rainbow* tablici tada se nad tim nizom primjenjuje *hash* funkcija te predzadnja funkcija redukcije (R_{N-1}) te se potom dobiveni niz opet traži u *rainbow* tablici. Ukoliko se pronađe opet se kreće od početne zaporke te se primjenjuje N-2 iteracija, a ako se ne pronađe postupak se ponavlja – maksimalno N puta.

Prikaz rada algoritma dan je pomoću niza *re3xes*, dobivenog kao izlaz iz *hash* algoritma, za koji je potrebno pronaći početnu zaporku koja je dala takav niz. Za lakše razumijevanje postupka koraci razbijanja zaporke prikazani su na slici 4.



Slika 4. Koraci razbijanja zaporke pomoću *rainbow* tablice

Koraci razbijanja zaporke:

I. Polazeći od izlaza *hash* funkcije (niz "re3xes") računa se zadnja redukcija korištena u tablici i provjerava se bilo koja zaporka dobivena u zadnjem stupcu tablice (korak 1).
II. Ako test ne uspije (niz *rambo* ne pojavljuje se u tablici) računa se lanac s dvije posljednje redukcije,a te dvije redukcije predstavljene su korakom 2. (Napomena: ako ni ovo testiranje ne uspije računanje se nastavlja lancem s 3 redukcije, pa 4 redukcije itd., sve dok se ne pronađe zaporka. Ako nijedan lanac ne sadrži zaporku napad nije uspio.)
III. Ako je testiranje pozitivno (korak 3, niz *linux23* pojavljuje se na kraju lanca i u tablici), razbijena zaporka nalazi se na početku lanca koji na kraju sadrži niz *linux23*. Na početku odgovarajućeg lanca u tablici pronalazi se niz *passwd*.
IV. U ovom trenutku (korak 4) generira se lanac i *hash* u svakoj iteraciji se uspoređuje s ciljanim. Test je valjan i pronalazi se *hash re3xes* u lancu. Trenutna zaporka *(kultura)* je ona koju je proizveo cijeli lanac i napad je uspješan.

Osnovni nedostatak tablica je u što se nakon konstruiranja mogu koristiti za razbijanje samo onih *hash* funkcija za koje su prvotno namijenjene. Znači, ako je tablica konstruirana za razbijanje LM algoritma može se primijeniti samo na zaporke šifrirane tim algoritmom. U nastavku su dani prosječne vrijednosti generiranja i razbijanja LM zaporki uz performanse CPU 666MHz i RAM 256MB.

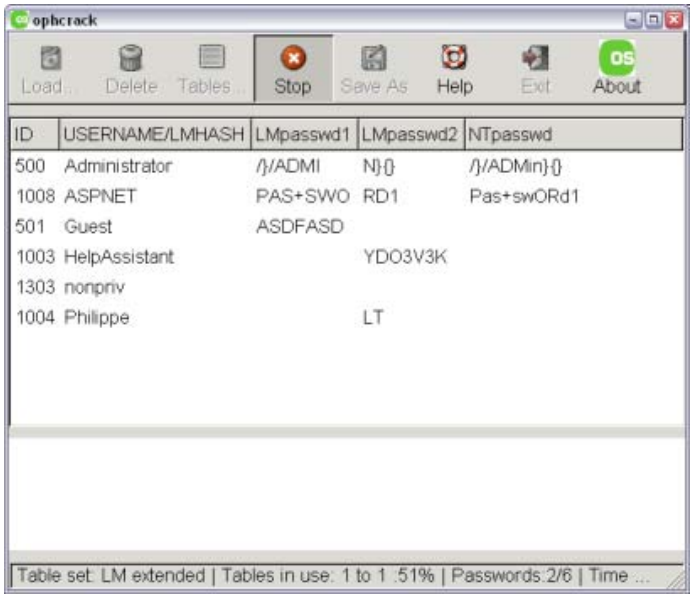
Skup znakova	Broj zaporki	Veličina tablica (GB)	Generiranje tablice (dani)	Prosjek razbijanja (s)	Vjerojatnost uspjeha (%)
A..Z	$8.35 * 10^9$	0.6	2.75	3.8	99,9
A..Z, 0..9	$8.06 * 10^{10}$	3	15.3	7.6	99,04
A..Z, 0..9, specijalni znakovi (14)	$7.97 * 10^{11}$	18.3	224	67	99,91
A..Z, 0..9, specijalni znakovi (32)	$6.82 * 10^{12}$	119	2354	197	99,9
Izvor: Project RainbowCrack, http://rainbowcrack.com/, siječanj 2007.					

Tablica 2. Prosječne vrijednosti generiranja i razbijanja LM 7-znakovnih zaporki

4.4. Dostupnost alata i tablica

Rainbows tablice moguće je samostalno kreirati pomoću nekog alata kao što je npr. program RainbowCrack. Kako bi korisnici dobili dvotjedno neograničeno korištenje 1.5TB tablica, prethodno moraju *upload*-ati neke od dogovorenih tablica koje se potom verificiraju. Podržani su LM, NTLM, MD2, MD4, MD5, SHA1, RIPEMD-160, MySQL v.3.23, MySQL SHA1 i Cisco PIX algoritmi. Dostupan je za operacijske sustave Linux, Microsoft Windows, Apple Mac OS X te OpenBSD, FreeBSD, Solaris, i / ili ostale UNIX inačice.

Besplatan program s besplatnim tablicama jest Ophcrack koji sadrže alfanumeričke zaporke za LM algoritam, dok su u komercijalnim oblicima dostupne tablice koje sadrže i specijalne znakove, a namijenjene su NTLM i LM algoritmima. Razvijen je za sljedeće operacijske sustave: Microsoft Windows, Linux, Unix i Mac OS X. Sučelje programa Ophcrack prikazano je na sljedećoj slici.



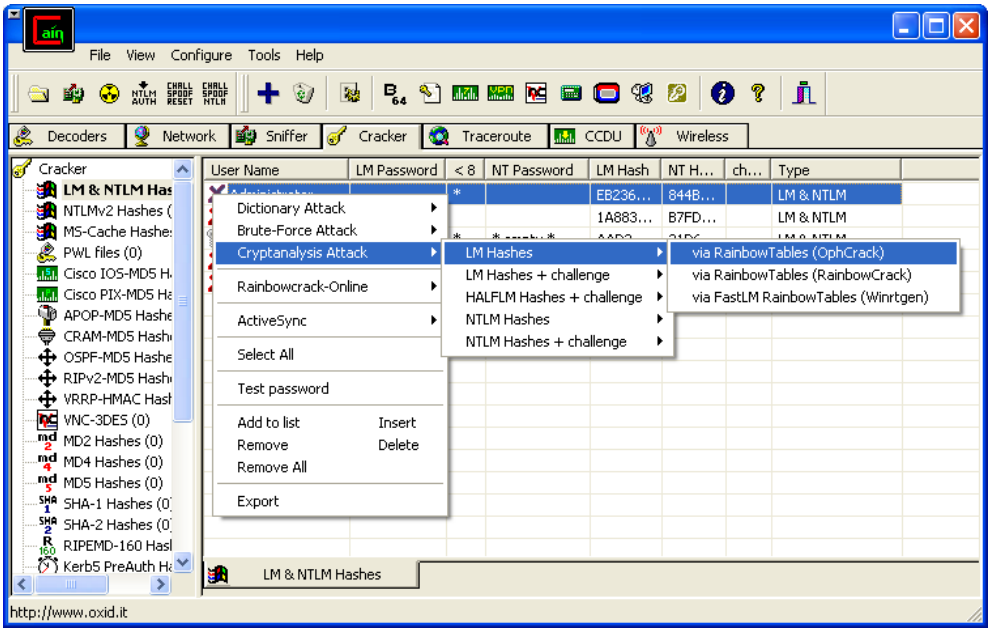
Slika 5. Sučelje programa Ophcrack

Drugi način dobivanja *rainbows* tablica je preuzimanje popularnih tablica koje se nude besplatno na stanicama Shmoo projekta (http://rainbowtables.shmoo.com/). Također, neke besplatne tablice moguće je pronaći i na sljedećoj stranici: http://wired.s6n.com/files/jathias/index.html.

Na kraju, *rainbows* tablice moguće je i kupiti na stranicama:

- http://www.rainbowtables.net/,
- http://www.rainbowcrack-online.com/.

Korištenje *rainbows* tablica moguće je i preko drugih programa kao što je npr. Cain and Abel - alat za oporavak zaporki Microsoft Windows operacijskog sustava. Omogućava oporavak različitih zaporka njuškanjem (eng. *sniffing*) po mreži, razbijanje kriptiranih zaporka pomoću rječnika, izvođenje *brute force* napada, snimanje VoIP razgovora, oporavak bežičnih mrežnih ključeva (eng. *wireless network keys*), analizu protokola usmjeravanja te mnoge druge pogodnosti.



Slika 6. *Rainbows* razbijanje korištenjem Cain & Abel alata

4.5. Zaštita od razbijanja zaporki

Mehanizam razbijanja temeljen na *rainbows* tablicama neuspješan je protiv jednosmjernih *hash* funkcija u kombinaciji sa *salting* mehanizmom (tzv. soljenje zaporki). Temelji se na dodavanju kratkog niza nazvanog „sol“ (eng. *salt*) zaporkama prije samog kriptiranja. *Salt* se razlikuje za svakog korisnika, pa napadač ne može izgraditi tablice s kriptiranim verzijama zaporka. Naime, takvim zaporkama se prilikom primjene *hash* funkcije dodaje određeni niz pa ako se razbijanjem pronađe neki niz koji generira traženu *hash* vrijednost, on se neće moći primijeniti jer će u *hash* funkciji biti modificiran – biti će mu dodan dodatak - *salt*. Stoga je jedino rješenje pronalaženje točne zaporke sa *salt* dodatkom koji se za potrebe prijave na sustav uklanja.

Na primjer, zaporka u *hash* obliku generira se pomoću sljedeće funkcije (gdje " . " predstavlja operator ulančavanja):

```
hash = MD5 (password . salt)
```

Salt učinkovito povećava duljinu i kompleksnost (dodavanje *salt* niza s posebnim znakovima povećava složenost alfanumeričkih zaporki) zaporke. Zbog toga ako *rainbows* tablice ne sadrže zaporke iste duljine i složenosti kao zaporka proširena *salt* nizom, zaporku nije moguće pronaći. Također, ako se pronađe odgovarajuća zaporka potrebno je, prije korištenja, ukloniti dodani *salt*.

Kako bi se spriječio napad pomoću *rainbows* tablica na određene web stranice, potrebno je pohraniti *salt* za svakog korisnika tj. dodijeliti svakom korisniku poseban *salt* niz. U protivnome napadač može kreirati *rainbow* tablicu za *salt* određene stranice i sve kombinacije mogućih zaporki.

Korištenje *salting* mehanizma na Unix operacijskim sustavima koristi se još od 1976. godine na način da se i korisničko ime doda uz zaporku prije izračunavanja *hash* vrijednosti. Iako je to inicijalno napravljeno kako bi se spriječilo da identične zaporke dvaju korisnika imaju istu *hash* vrijednost, prednost ovog principa vidljiva je i kod suzbijanja napada zasnovanih na korištenju *rainbows* tablica.

Stariji, ali i noviji Windows operacijski sustavi jedni su od rijetkih sustava; uz neke usmjerivače, vatrozide i baze podataka, koji koriste kriptografske algoritme bez tzv. *salt* primjesa, što *rainbows* razbijanje čini idealnom metodom. Autenikacijski protokoli LanManager (LM) i NTLM ne koriste nikakve primjese. Dodatno, LM protokol je posebno ranjiv jer sva slova promatra kao velika te dozvoljava maksimalno 14

znakova koje promatra kao dva niza od 7 znakova koji se zasebno sažimaju. Noviji Windows operacijski sustavi, zbog kompatibilnost sa starijim inačicama Windows sustava, prilikom autentikacije uz NTLM odgovor automatski šalju i LM odgovor čime se efektivno gube sigurnosna poboljšanja NTLM autentikacije. Ta opcija je na XP i 2003 predefinirana, a na Visti proizvoljna. Korisnici Windows sustava trebali bi stoga onemogućiti slanje LM zaporki u autentikacijskim porukama.

5. Zaključak

Sigurnost sustava na kojima su pohranjeni osjetljivi podaci oduvijek predstavlja veliki problem. Da bi se osigurao integritet nekog poslužitelja pristup podacima treba omogućiti samo autoriziranim korisnicima. Upravo zbog toga dolazi do potrebe za „skrivanjem" zaporki koje se koriste prilikom prijave na sustave.

Prilikom spremanja ili prijenosa podataka većina sustava prvo šifrira osjetljive podatke. Ipak napadač može prikupiti šifrirane osjetljive podatke te pokušati obnoviti zaporke. Razvijeno je mnogo načina razbijanja algoritama šifriranja koji omogućuju razotkrivanje zaporki. Jedan od najefikasnijih (najbržih) načina je primjena *rainbows* tablica koje smanjuju vrijeme potrebno za razbijanje zaporki zahvaljujući *time-memory trade-off* principu.

Iako korištenje *rainbows* tablica u većini slučajeva ne garantira otkrivanje zaporki, trenutno su raspoložive tablice s vrlo visokim vjerojatnostima otkrivanja (~99%). Unatoč dugom vremenu potrebnom za izračunavanje, njihova prednost svakako je i u tome da nakon što su jednom uspješno generirane moguće ih je neograničeno koristiti. Stoga se pokraj njih *brute-force* metode čine kao veliki gubitak resursa i vremena jer se brojne kombinacije zaporki kod svakog novog traženja ponovno isprobavaju. Dodatno, *rainbows* metode još uvijek su efikasne i kod novih Windows sustava koji radi kompatibilnosti sa starijim sustavima u autentikacijskim mrežnim porukama šalju i vrlo ranjive LM *hash* vrijednosti zaporki.

Kako postoje brojni načini na koje napadač može prikupiti podatke za razbijanje zaporki te obnoviti šifrirane zaporke, potrebno je primijeniti bar osnovne mjere zaštite. Za zaštitu korisnici trebaju koristiti i što duže zaporke s nestandardnim znakovima jer su *rainbows* tablice prilagođene za manje-više standardne znakove i za ograničene duljine zaporki. Svaki korisnik može pridonijeti sigurnosti sustava pravilnim izborom zaporke pridržavanjem pravila o odabiru nesigurnih zaporki.

Važnu ulogu igra i pravilna implementacija sustava, pa npr. administrator treba pratiti prijave na sustav da bi mogao uočiti pokušaje prijave pogađanjem zaporki. Jedan od načina zaštite zaporki je uvođenje *salting* mehanizma, koji izmjenjuje odabrane zaporke dodajući im *salt* dodatak.

6. Reference

[1] Razbijanje zaporki, http://en.wikipedia.org/wiki/Password_cracking, kolovoz 2008.,

[2] «Brute force» napad, http://en.wikipedia.org/wiki/Brute_force_attack, kolovoz 2008.,

[3] «Dictionary» napad, http://en.wikipedia.org/wiki/Dictionary_attack, kolovoz 2008.,

[4] «Salting», http://en.wikipedia.org/wiki/Salt_(cryptography), kolovoz 2008.,

[5] Alat L0phtSrask, http://en.wikipedia.org/wiki/L0phtCrack, kolovoz 2008.,

[6] Alat Ophcrack, http://ophcrack.sourceforge.net/, kolovoz 2008.,

[7] Alat Ophcrack, http://en.wikipedia.org/wiki/Ophcrack, kolovoz 2008.,

[8] Alat Ivan Ripper, http://www.openwall.com/john/, kolovoz 2008.,

[9] Alat Ivan Ripper, http://en.wikipedia.org/wiki/John_the_Ripper, kolovoz 2008.,

[10] Alat Rainbow crack, http://www.antsight.com/zsl/rainbowcrack/, kolovoz 2008.,

[11] Project RainbowCrack, http://rainbowcrack.com/, kolovoz 2008.,

[12] Alat Rainbow crack, http://en.wikipedia.org/wiki/RainbowCrack, kolovoz 2008.,

[13] Time - Memory Trade – Off princip, http://en.wikipedia.org/wiki/Time-space_tradeoff, kolovoz 2008.,

[14] Pillippe Oschslin: Making a Faster Gryptanalytic Time-Memory Trade-Off, LASEC, Switzerland, 2003., http://lasecwww.epfl.ch/~oechslin/publications/crypto03.pdf,

[15] J. Borst, B. Preneel I J. Vanderwalle: On the Time-Memory Tradeoff Between Exhaustive Key Serach and Table Precomputition, Belgija,

[16] Martin E. Hellman: A Cryptoanalytic Time-Memory Trade-Off, IEEE, 1980., http://www-ee.stanford.edu/~hellman/publications/36.pdf,

[17] Zhu Shuanglei: Parameter optimization of time-memory trade-off cryptanalysis in RainbowCrack, 2004., http://www.antsight.com/zsl/rainbowcrack/optimization/optimization.htm,

[18] T. Katić, Razbijanje zaporki u bojama duge, časopis „Mreža“, Zagreb, ožujak 2008.

[19] Rainbows tablice, http://en.wikipedia.org/wiki/Rainbow_table, kolovoz 2008.

[20] How Rainbow Tables work: http://kestas.kuliukas.com/RainbowTables/, kolovoz 2008.

[21] Rainbows tablice, http://www.rainbowtables.net/faq.php, kolovoz 2008.