HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Autentikacija na web poslužiteljima

CCERT-PUBDOC-2006-05-158

CARNet CERT u suradnji s LS&S

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

Rezultat toga rada ovaj je 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

Tema ovog dokumenta su metode autentikacije koje se koriste za identifikaciju i autentikaciju korisnika i web klijenata koji pristupaju web poslužiteljima. Autentikacija je proces kojim se potvrđuje identitet korisnika prilikom pristupanja nekom servisu, u ovom slučaju web poslužitelju. Ukoliko je autentikacijom potvrđen identitet onda obično slijedi proces autorizacije kojim se korisniku na osnovu njegovog identiteta, daju određene ovlasti nad resursima web poslužitelja. Iz ovog je očita važnost točnog utvrđivanja i potvrđivanja identiteta, tj. važnost sigurnosnog aspekta procesa autentikacije.

Sigurnosna praksa definira 3 osnovne vrste autentikacije kojima korisnik potvrđuje svoj identitet: pomoću nečeg što korisnik zna (zaporka), nečeg što korisnik posjeduje (kreditna kartica, ID token) ili nečeg što korisnik jest (biometrijski podaci korisnika). Zbog praktičnosti i jednostavnosti, za autentikaciju prilikom pristupa web poslužiteljima najčešće se koristi autentikacija pomoću zaporke. Implementacija takve autentikacije tema je ovog dokumenta kao i rizici kojima je takva autentikacija izložena. Za izbjegavanje takvih rizika moguće je implementirati i napredne metode autentikacije koje će također biti opisane u nastavku dokumenta.

2. Osnove autentikacije

Kao što je navedeno u uvodu, 3 osnovna sredstva autentikacije su:

- nešto što korisnik zna (zaporka),
- nešto što korisnik posjeduje (kreditna kartica, ID token) i
- nešto što korisnik jest (biometrijski podaci korisnika).

Različita sredstva autentikacije daju različite nivoe sigurnosti i praktičnosti u realizaciji što je opisano u sljedećim poglavljima.

2.1. Zaporka

Najčešće korišteno sredstvo autentikacije u informacijskim sustavima je nešto što korisnik zna, tj. zaporka (eng. *password*) iz jednostavnog razloga što je taj način autentikacije jednostavan za korisnika i za administratora sustava koji ga autenticira, a istovremeno garantira razuman nivo sigurnosti.

Ali ova metoda ima i svojih nedostataka, od toga što korisnici često zaboravljaju svoje zaporke ili biraju prejednostavne odnosno očite zaporke pa do toga da koriste iste zaporke za više različitih sustava ili čak za sve sustave kojima pristupaju. Osim toga, zaporka kao takva mora biti unesena u cijelosti od strane korisnika te nakon toga i prenesena od korisnika do sustava koji ga autenticira čime je izložena riziku da bude uočena i otkrivena, što zlonamjernom korisniku omogućava preuzimanje potpune kontrole nad sustavom.

Problem prejednostavnih zaporki može se donekle riješiti prisiljavanjem korisnika u vezi korištenja kompleksnih zaporki koje se sastoje od brojeva, slova i interpunkcijskih znakova, ali problem korištenja identičnih zaporki na različitim sustavima ostaje. Većina pružatelja web usluga je savršeno svjesna kako bi uvidom u svoje baze podataka u kojima pohranjuju korisničke adrese elektroničke pošte i zaporke, vrlo lako mogla s istim korisničkim podacima zlorabiti servise koje taj korisnik koristi kod nekih drugih poslužitelja. Jedino rješenje ovog problema je izdavanje slučajnih zaporki korisnicima pri čemu im se ne daje mogućnost mijenjanja. Međutim, takve zaporke su u većini slučajeva teško pamtljive za korisnike pa se ovakvo rješenje preporuča samo za sigurnosno izrazito osjetljive usluge.

Problem mogućnosti otkrivanja zaporke prilikom njenog transporta moguće je do određene mjere riješiti enkripcijom zaporke (npr. pomoću SSL protokola), ali to još uvijek ostavlja mogućnost njenog otkrivanja na mjestu njenog unosa ili na mjestu njenog korištenja za autentikaciju, prije čega ona mora biti dekriptirana. Otkrivanje na mjestu unosa je daleko vjerojatnije, npr. prilikom unosa zaporke u Internet *cafe*-u – zlonamjerni vlasnik *cafe*-a uvijek može instalirati neki alat za pamćenje unesenih zaporki. Smanjenje ovog rizika može se postići upotrebom tzv. „*password challenge*“ metode autentikacije gdje se od korisnika ne traži unos cijele zaporke već samo nekih njenih znakova biranih slučajno kod svakog novog unosa. Na taj način čak ako napadač i uspije presresti podatke koje korisnik unese, neće ih imati dovoljno za iskorištavanje. Ovo rješenje se često koristi za pristup bankarskim uslugama jer je jednostavno za korisnika, a opet garantira značajno veći nivo sigurnosti.

2.2. *Phishing*

Phishing je termin koji se koristi za opis različitih metoda kojima se korisnika navodi na otkrivanje svoje zaporke, bez da je korisnik toga svjestan. Ova pojava postaje sve učestalija posebno u području financijskih usluga gdje je registriran popriličan broj ovakvih napada na sigurnost i podaci govore o 5% korisnika koji su na ovaj način bili navedeni na otkrivanje sadržaja svoje zaporke.

Osnovni način zaštite od ovakvih napada je edukacija korisnika – korisnici moraju biti upozoreni da ne odaju svoje osobne podatke te da ih ne šalju elektroničkom poštom kao odgovor na neprovjereni zahtjev. Pružatelji web usluga bi također morali voditi brigu o tome te sami ne tražiti od korisnika da im osobne podatke šalju putem elektroničke pošte.

Moguće je, bar do određene mjere, i tehničko rješenje ovog problema. Jedna opcija bi bila da administratori mreže blokiraju sve poznate *phishing* web poslužitelje. Druga mogućnost pruža se upotrebom raznih traka (eng. *bar*) upozorenja koje su dostupne u nekim web klijentima, npr. Mozilla TrustBar koja korisniku prikazuje stvarnu domenu web stranice na način koji je gotovo nemoguće lažirati. Na taj način provjerene domene mogu biti konfigurirane tako da se prikazuju na specifičan način te bi korisnik lako uočio promjenu prilikom prelaska na lažnu.

2.3. Metoda jedinstvene autentikacije

Ideja za rješenje mnogih problema vezanih uz autentikaciju zaporkom je implementacija mehanizma jedinstvene autentikacije (eng. *Single Sign-On*). Ideja je zasnovana na tome da se korisnik ne autenticira na svaki web poslužitelj posebno, već se on autenticira samo jednom i to na centralnom autorizacijskom sustavu čijoj autentikaciji onda vjeruju svi ostali web poslužitelji. Mnogo je potencijalnih prednosti ovakve autentikacije uključujući smanjenu potrebu za administracijom web poslužitelja kao i manji broj zaporki koje korisnik mora zapamtiti. Metoda jedinstvene autentikacije ne mora nužno biti zasnovana na zaporki, ali obično jest.

Metoda jedinstvene autentikacije može biti implementirana na različite načine – Windows Domain Controller je jedan od njih gdje se autentikacija web poslužitelja može integrirati s autentikacijom domene korištenjem Kerberos protokola. Ali ovakvo rješenje je uglavnom korisno u intranet okruženjima jer nisu svi korisnici Interneta i korisnici iste domene. Postoje i Internet varijante metode jedinstvene autentikacije od kojih je najraširenija Microsoft Passport.

Međutim nijedno od postojećih rješenja nije ni blizu doseglo razinu od upotrijebljenosti od 100% pa web poslužitelji i dalje moraju omogućavati pristup i korisnicima koji nemaju *Single Sign-On* identitet. Naravno, postoji opcija gdje se korisnike prisiljava na kreiranje *Single Sign-On* identiteta, ali to se kao dodatni zahtjev korisnicima ne mora nužno svidjeti i može ih potpuno odvratiti od korištenja usluge pa kao najbolja opcija ostaje kombiniranje metode jedinstvene autentikacije i klasične autentikacije putem zaporke.

2.4. SSL klijentski certifikati

Malo korištena funkcionalnost SSL protokola nalazi se u mogućnosti korištenja klijentskih certifikata kao alternative autentikacije pomoću nečeg što korisnik zna. U takvoj situaciji klijent odgovara na zahtjev za autentikaciju prezentiranjem certifikata i odgovorom na kriptiranu poruku čime se rješavaju neki od mnogih problema koje donosi autentikacija zaporkom.

Klijent u ovom slučaju ne otkriva sadržaj svoje zaporke već koristi svoj privatni ključ za dekriptiranje poruke i slanje odgovora, dok sam ključ nikad ne šalje web poslužitelju. Na taj način korisnik može koristiti isti certifikat za autentikaciju prema raznim poslužiteljima i tako potpuno rješava problem korištenja iste zaporke za više poslužitelja. Također, time je riješen i problem *phishing*-a jer se privatni ključ nikad ne šalje izvan korisnikovog računala.

S druge strane postavlja se problem pamćenja privatnog ključa – obično je to slučajan niz binarnih podataka koji korisnik ne može zapamtiti pa ga mora pohraniti na svom računalu. Time se ograničava kretanje korisnika jer se on može autenticirati samo sa svog računala i to predstavlja glavno ograničenje ove metode autentikacije dok god se korisniku ne omogući nošenje privatnog ključa. Takvo rješenje danas postaje sasvim moguće upotrebom pametnih kartica koje su dovoljno male i praktične dok istovremeno osiguravaju dovoljan nivo sigurnosti.

2.5. Autentikacija pomoću nečeg što korisnik posjeduje

Ovakva autentikacija može biti korištena i u *on-line* svijetu i neka rješenja već postoje iako ne daju uvijek zadovoljavajući nivo sigurnosti. Najjednostavniji primjer je pohranjivanje slučajno generirane zaporke na USB memorijski priključak koji se onda koristi kao fizički token za autentikaciju. Ipak, nitko ne sprečava ovlaštenog korisnika ili čak zlonamjernog, kod neovlaštenog kopiranja podataka s priključka.

Sigurniji način implementacije ove metode autentikacije je korištenje RSA Secure ID tokena, male naprave s LCD ekranom s kojeg se očitava slučajan niz brojeva koji se mijenja svake minute i predstavlja jednokratnu zaporku čime se efektivno dobiva autentikacija pomoću nečeg što korisnik ima i nečeg što korisnik zna. Ipak, osim što je nepraktično, ovo rješenje nije ni jeftino pa se upotrebljava samo u situacijama gdje je sigurnost od vitalnog značaja.

Jednostavnija implementacija ovakve autentikacije koristi se često za *on-line* bankarske usluge – npr. izdavanje lista s mrežom slučajno generiranih znakova gdje je za autentikaciju potrebno unijeti znakove s određenih pozicija u mreži ili pak izdavanje lista s 50 jednokratnih zaporki gdje korisnik mora izgrepsti zaštitni sloj sa svake zaporke kako bi je pročitao i koristio.

2.6. Autentikacija pomoću biometrije

Autentikacija pomoću biometrije nije nova metoda – primjer takve autentikacije je korištenje fotografije ili otiska prsta. Nažalost postoji velik problem koji sprečava korištenje ove metode za autentikaciju na web poslužiteljima – kako bi se korisnik autenticirao ovom metodom on mora skenirati svoj otisak i poslati sliku web poslužitelju. Ali web poslužitelj koji prima samo sliku ne može znati ukoliko je slika stigla sa skenera ili s nekog drugog uređaja pa je tako vrlo lako lažirati ovaj oblik autentikacije pri čemu mora biti ispunjen uvjet prema kojem napadač posjeduje sliku otiska. To dovodi do drugog problema biometrijske autentikacije – ako napadač dođe do slike korisnikovog otiska prsta, korisnik ga ne može promijeniti te se više ne može autenticirati pomoću te metode.

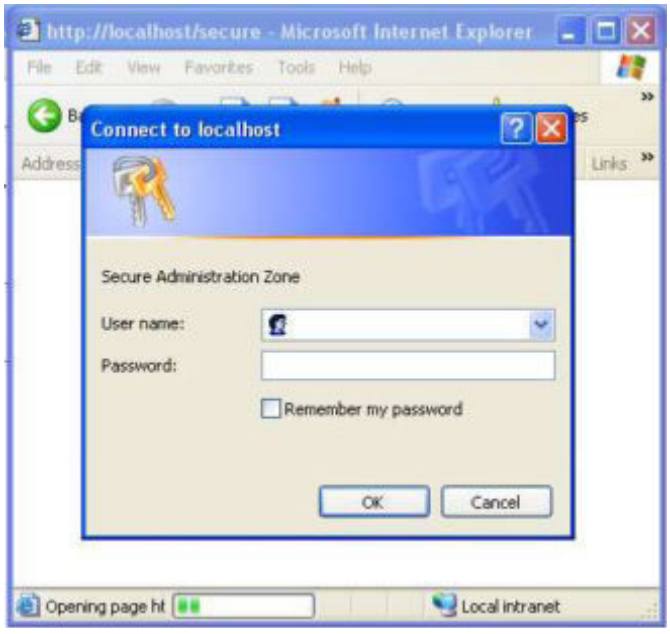
Moguće rješenje za taj problem je kombiniranje dvije metode, npr. korisnik tek nakon autentikacije pomoću otiska prsta dobiva uvid u jednokratnu zaporku. Iako ovo izgleda kao dovoljno sigurna metoda, ipak se opet postavlja pitanje uređaja koji skenira – kako skener koji je elektronička naprava zna da je skenirao otisak prsta, a da nije umjesto toga primio elektroničku simulaciju istog. Zbog ovakvih problema biometrijska autentikacija je pogodna za korištenje samo u kontroliranim uvjetima gdje ljudi osiguravaju ispravno korištenje uređaja za biometrijsku autentikaciju.

3. Osnovne metode autentikacije na web poslužiteljima

Sljedeća poglavlja daju kratak prikaz implementacije različitih metoda autentikacija koje se uobičajeno koriste za autentikaciju na web poslužiteljima.

3.1. HTTP autentikacija

HTTP autentikacija ili osnovna autentikacija je često korištena implementacija autentikacije putem zaporke koja je sastavni dio HTTP protokola te je podržana od većine postojećih web poslužitelja i web klijenata. Sa strane klijenta to se manifestira kao pojavljivanje posebnog prozora koji iskače (eng. *pop-up*), a kroz njega se od korisnika traži unos korisničkog imena i zaporke.



Slika 1: Izgled *pop-up* prozora za dijalog kod HTTP autentikacije

Nakon što korisnik, tj. web klijent pošalje tražene podatke korištenjem *pop-up* prozora za dijalog, oni su putem HTTP zahtjeva poslani web poslužitelju koji ih verificira te potom odobrava ili ne odobrava pristup traženim resursima ili uslugama. Pošto HTTP protokol ne podržava različita stanja komunikacije, web klijent mora poslati podatke za autentikaciju prilikom svakog sljedećeg zahtjeva, ali to za korisnika odrađuje klijentska aplikacija koja za vrijeme jedne sesije privremeno pohranjuje korisnikove podatke te ih potom po potrebi automatski šalje web poslužitelju.

Pošto je HTTP autentikacija u većini slučajeva već implementirana kao dio aplikacije web poslužitelja, dizajner web aplikacija nema puno mogućnosti za eventualne izmjene njene implementacije. To znači da web dizajner ne može mijenjati izgled *pop-up* prozora, lokaciju njegovog pojavljivanja ili njegov sadržaj pa tako na primjer nije moguće u sadržaj *pop-up* prozora dodati vezu kojom bi se kreirao novi korisnički račun.

3.1.1. HTTP autentikacija na Apache web poslužitelju

HTTP autentikaciju, kao najjednostavniji oblik autentikacije na web poslužiteljima, je vrlo jednostavno implementirati jer to uglavnom zahtijeva unošenje određenih konfiguracijskih parametara prilikom konfiguracije web poslužitelja. Primjer koji slijedi prikazuje konfiguraciju HTTP autentikacije na Apache web poslužitelju.

Prvi korak je specificiranje resursa na web poslužitelju kojima se želi ograničiti pristup, tj. za koje će se tražiti autentikacija korisnika. To se postiže smještanjem svih sadržaja kojima se želi ograničiti pristup u zaseban direktorij unutar poslužiteljevog osnovnog direktorija. Zaštita nad tim direktorijem se zatim konfigurira dodavanjem direktiva u `.htaccess` datoteku koja se nalazi u tom direktoriju ili dodavanjem istih direktiva u odgovarajući `<Directory>` dio unutar globalne konfiguracijske datoteke `httpd.conf` web poslužitelja. Direktive su identične u oba slučaja i sadrže minimalno sljedeće podatke:

- autentikacijska domena (*Authentication Realm Name*),
- naziv datoteke ili baze podataka s korisničkim podacima (eng. *User Database Name*) – ime i lokacija datoteke ili baze podataka u kojoj su pohranjena važeća korisnička imena i pripadne zaporke (napomena: lokacija ove datoteke ili baze podataka ne smije biti osnovni direktorij web poslužitelja jer je on dostupan svima, a sama datoteka ili baza podataka dakako ne smije biti dostupna nikom osim administratoru sustava) i
- ograničene operacije – popis HTTP transakcija za koje je potrebna autentikacija.

Nakon ovakve konfiguracije, ukoliko klijent pokuša pristupiti web stranici ili CGI skripti pohranjenoj u zaštićenom direktoriju, događa se sljedeći niz događaja:

1. Web klijent šalje HTTP zahtjev poslužitelju za pristup stranici.
2. Pokreće se HTTP proces na poslužitelju i uočava se traženje pristupa stranici u zaštićenom direktoriju. Potom HTTP proces provjerava konfiguraciju tog direktorija i uočava da pristup tom direktoriju zahtijeva autentikaciju. Pošto u zahtjevu nema autentikacijskih podataka, HTTP proces odbija pristup traženoj stranici i šalje odgovor u kojem se traže autentikacijski podaci za pristup autentikacijskoj domeni prema podatku upisanom kao *Authentication Realm Name* unutar `.htaccess` datoteke ili `<Directory>` dijela konfiguracijske datoteke.
3. Korisnikov web klijent prima poruku o odbijenom zahtjevu i pretražuje ukoliko u lokalnoj memoriji ima pohranjene autentikacijske podatke za traženu autentikacijsku domenu i web poslužitelj. Ukoliko ne nađe potrebne podatke prikazuje *pop-up* prozor za dijalog u koji korisnik upisuje autentikacijske podatke te ih potom pohranjuje u lokalnu memoriju.
4. Nakon dobivanja autentikacijskih podataka korisnikov web klijent šalje ponovljeni zahtjev (opisan u koraku 1), ali ovaj put s autentikacijskim podacima.
5. HTTP proces na web poslužitelju obrađuje zaprimljeni zahtjev kao u koraku 2. Ponovno provjerava `.htaccess` datoteku ili `<Directory>` dio konfiguracijske datoteke i otkriva kako je za pristup potrebna autentikacija, ali ovaj put u zahtjevu nalazi autentikacijske podatke.
6. HTTP proces provjerava da li autentikacijski podaci odgovaraju onima pohranjenim u bazi podataka. Ukoliko dani korisnički račun ne postoji u bazi ili je za dani račun upisana pogrešna zaporka, HTTP proces odgovara s odbijanjem zahtjeva, što obično rezultira ponovnim pojavljivanjem autentikacijskog *pop-up* prozora na strani korisnika.
7. Ako dani podaci odgovaraju onima u bazi podataka, HTTP proces provjerava ako postoje ikakva dodatna ograničenja (da li je pristup nekim dokumentima dozvoljen samo pripadnicima određene grupe). Ako postoje dodatna ograničenja, HTTP proces još uvijek može odbiti zahtjev za pristupom.
8. Ako su sve provjere zadovoljene, a traženi sadržaj je web stranica, web poslužitelj šalje stranicu, a ako je traženi sadržaj CGI skripta, web poslužitelj pokreće skriptu uzimajući

korisničko ime dobiveno u zahtjevu kao vrijednost `REMOTE_USER` konfiguracijske varijable (zaporka se ne koristi) – time obavještava CGI skriptu tko ju koristi.

Teoretski cijeli ovaj proces bi se trebao ponoviti za svaki sljedeći zahtjev, ali zbog inteligencije i lokalne memorije web klijenta koji automatski pronalazi autentikacijske podatke procedura je skraćena. Moguća je pojava pogrešne autentikacije ukoliko za jedan web poslužitelj postoji više definiranih autentikacijskih domena što bi možda moglo zbuniti neke web klijente, ali vjerojatnost za to je minimalna.

Direktive kojima se aktivira HTTP autentikacija na Apache web poslužitelju su:

- `AuthType` – služi za podešavanje tipa autentikacije,
- `AuthName` – služi za podešavanje imena autentikacijske domene,
- `AuthUserFile` – služi za podešavanje lokacije *password* datoteke koja sadrži autentikacijske podatke,
- `AuthGroupFile` – služi za podešavanje lokacije *group* datoteke (opcionalno) i
- `require` – služi za podešavanje zahtjeva koje treba zadovoljiti.

Primjer takve direktive je:

```
AuthType Basic
AuthName "Samo lokalni korisnici"
AuthUserFile /direktoriji/password
require user korisnik1 korisnik2 //dozvoljen je pristup samo za 2
korisnika
-ili-
require valid-user // svim autenticiranim korisnicima se dozvoljava
                   // pristup
```

U prikazanom primjeru klijentima će biti prikazan tekst „Samo lokalni korisnici“ unutar *pop-up* prozora kao ime autentikacijske domene. Prikazane direktive mogu biti ili dio `.htaccess` datoteke ili Apache-ove konfiguracijske datoteke `httpd.conf`. Ako se direktive ubace u `.htaccess` datoteku onda nije potrebno ponovo pokretati poslužitelj nakon modificiranja datoteke. Također ako se tijekom rada direktoriji ograničenog pristupa premještaju, `.htaccess` datoteka premješta se s njima. S druge strane ako se direktive ubacuju u Apache konfiguracijsku datoteku `httpd.conf` onda se sve kontrole pristupa nalaze na jednom mjestu i web poslužitelj mora biti ponovno pokrenut kako bi izmjene postale aktivne.

Sami autentikacijski podaci na web poslužitelju nalaze se u *password* datoteci. Prilikom konfiguracije autentikacije na web poslužitelju, uz već opisano, potrebno je i kreirati ili modificirati *password* datoteku, a kada je to potrebno, isto učiniti i za *group* datoteku kojom se definiraju grupe klijenata.

Kreiranje *password* datoteke obavlja se korištenjem pomoćnog programa `htpasswd` koji dolazi kao dio Apache web poslužitelja i nalazi se u Apache-ovom `bin` direktoriju. *Password* datoteka sadrži listu važećih korisničkih imena i zaporki gdje su zaporke pohranjene u kriptiranom obliku. Ova datoteka nikako ne smije biti pohranjena u direktoriju gdje se nalaze i podaci nad kojima ona ograničava pristup.

Ukoliko ne postoji `password` datoteka potrebno ju je kreirati korištenjem `htpasswd` naredbe i dodati u nju jednog korisnika. Primjer kreiranja `password` datoteke unutar `security` direktorija u 4 koraka izgleda ovako:

```
cd /direktorij
mkdir security
htpasswd -c /direktorij/security/password korisnik1
//slijedi dvostruki unos odabrane zaporke za korisnika korisnik1
```

Nakon ova 4 koraka kreirana je `password` datoteka unutar `security` direktorija koja sadrži jednog korisnika. Za dodavanje korisnika u već postojeću `password` datoteku potrebno je napraviti sljedeće:

```
cd /direktorij/security
htpasswd /direktorij/security/password korisnik2
```

```
//slijedi dvostruki unos odabrane zaporke za korisnika korisnik2
```

Rezultat ovih naredbi je datoteka `password` koja na dnu ima dodanog još jednog korisnika „korisnik2“.
Iako su same zaporke unutar datoteke kriptirane, sama datoteka bi trebala biti maksimalno zaštićena promjenom ovlasti nad datotekom. Vlasnik datoteke (`root`) treba imati ovlasti za pisanje, dok proces web poslužitelja (HTTP proces) mora imati samo ovlasti za čitanje. Za promjenu ovlasti nad `password` datotekom gdje je `root` vlasnik, a `apache` grupa, potrebno je koristiti sljedeći niz naredbi.

```
chown root /direktorij/security/password
chgrp apache /direktorij/security/password
chmod 640 /direktorij/security/password
```

Ukoliko se želi dozvoliti istovrstan pristup za više korisnika, to se može učiniti njihovim grupiranjem pomoću *group* datoteke. Ta datoteka mora biti smještena na istom mjestu gdje se nalazi i `password` datoteka. Dodavanje i brisanje korisnika iz grupe se radi modificiranjem *group* datoteke, a kako bi promjene postale efektivne nije potrebno ponovo pokretati Apache web poslužitelj. Format *group* datoteke je jednostavan – prvo se navodi ime grupe, a zatim imena njenih članova odvojena znakom razmaka. Sljedećim primjerom kreira se datoteka imena `grupa1` unutar `grupa` direktorija koji se pak nalazi u `security` direktoriju:

```
mkdir /direktorij/security/grupa
cd /direktorij/security/grupa
vi grupa1
// unos sljedećeg teksta:
// grupa1: korisnik5 korisnik6 korisnik7
```

Da bi se grupa aktivirala potrebno je dodati sljedeće direktive unutar `.htaccess` datoteke ili globalne konfiguracijske datoteke `httpd.conf`.

```
AuthType Basic
AuthName "Samo lokalni korisnici"
AuthUserFile /direktoriji/password
AuthGroupFile /direktoriji/grupa
require group grupa1
AuthAuthoritative on
```

U ovakvoj situaciji, kad web poslužitelj primi zahtjev za resursom, prvo provjerava ukoliko se dano korisničko ime nalazi u definiranoj grupi prema *group* datoteci, a tek onda provjerava da li zaporka odgovara onoj pohranjenoj u *password* datoteci.
Važno je primijetiti u gornjem primjeru kako je `AuthAuthoritative` opcija postavljena na *`On`* čime su autorizacijske direktive proglašene „autoritativnim“ što znači da u slučaju u kojem iz nekog razloga *password* datoteka ili *group* datoteka postanu nedostupne, web klijentu će biti onemogućen pristup resursima. Ako je ova opcija postavljena na *`Off`*, onda klijent može koristiti neku drugu direktivu ukoliko takva postoji unutar Apache konfiguracijske datoteke `httpd.conf` i na osnovu nje ostvariti pristup.
HTTP autentikacija nije dovoljno sigurna jer iako su korisnička imena i zaporke pohranjene u zaštićenom direktoriju na samom poslužitelju, oni se svejedno prenose preko mreže u obliku nekriptiranog teksta prilikom slanja svakog HTTP zahtjeva. To znači da svako tko može pratiti promet mreže (npr. nekim alatom za praćenje mrežnog prometa), može vrlo lako doznati korisničko ime i zaporku koje bi mu omogućile pristup web poslužitelju.

3.2. *Digest* autentikacija

Digest autentikacija je u osnovi gotovo identična HTTP autentikaciji i kao takva je također dio HTTP standarda. Glavni razlog uvođenja *Digest* autentikacije je glavni nedostatak HTTP autentikacije –

prijenos korisničkog imena i zaporke preko mreže u obliku nekriptiranog teksta. Kod *Digest* autentikacije, autentikacijski podaci su prilikom transfera od klijenta do poslužitelja kriptirani.
Procedura *Digest* autentikacije ukratko se može opisati ovim koracima:

1. Klijent zatraži pristup poslužitelju i biva odbijen, a povratna poruka o odbijanju zahtjeva sadrži obavijest kako je potrebna *Digest* autentikacija i sadrži niz znakova koji se naziva „*nonce*“ i obično se zasniva na trenutnom vremenu i IP adresi podnositelja zahtjeva te je različit za svaki novi zahtjev.
2. Kao i kod HTTP autentikacije korisnik unosi korisničko ime i zaporku (ili ih klijent dohvaća iz lokalne memorije), klijent ih konkatenira zajedno s nazivom autentikacijske domene i izračunava MD5 sumu za taj konkatenirani niz znakova.
3. Klijent konkatenira traženi URL i metodu zahtjeva te računa MD5 sumu tog konkateniranog niza znakova.
4. Konkatenira dvije dobivene MD5 sume i konkatenira ih s „*nonce*“ nizom znakova te računa MD5 sumu tog konkateniranog niza znakova
5. Klijent šalje HTTP zahtjev za novom stranicom, a unutar zahtjeva nalazi se MD5 suma izračunata u koraku 4, kao i čisti tekst korisničkog imena i „*nonce*“ niza znakova

MD5 je algoritam koji uzima niz znakova te neovisno o njegovoj dužini generira 16 bitnu MD5 sumu. Algoritam je dizajniran tako da je iz same MD5 sume gotovo nemoguće dešifrirati originalni niz znakova iz kojeg je nastao pa u biti predstavlja jednosmjerni algoritam za enkripciju.
Kad web poslužitelj zaprimi novi zahtjev s autentikacijskim podacima, uzima korisničko ime i pronalazi mu pridruženu zaporku te računa iste MD5 sume koje je računao klijent. Ako se krajnji rezultat poklapa, znači da je klijent koristio istu zaporku i pristup mu je odobren. Druga mogućnost je da je netko lažirao zaporku, ali na taj način je vrlo teško dobiti istu krajnju MD5 sumu za što su šanse 1 nasuprot $340*10^{36}$. U osnovi je bolje kada poslužitelj pohranjuje prvu MD5 sumu (korisničko ime + autentikacijska domena + zaporka) umjesto čistog teksta zaporke i tako osigura dodatnu zaštitu i ujedno skrati vrijeme računanja, ali to ovisi o implementaciji.
Osim zaštite autentikacijskih podataka prilikom prijenosa, *Digest* autentikacija nudi i još jednu mogućnost zaštite, a ta je izračunavanje MD5 sume cjelokupnog zahtjeva ili odgovora koja može biti uključena u zahtjev/odgovor. Time se omogućava objema stranama (klijent i poslužitelj) provjeravanje ukoliko je primljeni zahtjev/odgovor bio izmijenjen prilikom prijenosa.
Iako je *Digest* autentikacija nešto sigurnija od HTTP autentikacije, ona ima i svojih nedostataka. Naime, iako potencijalni napadač ne može praćenjem prometa dobiti uvid u korisnikovu zaporku, on svejedno može vidjeti cijeli zahtjev te ga takvog u cijelosti ponovo poslati i dobiti pristup traženom resursu. Ovaj problem bi se mogao riješiti pomoću „*nonce*“ niza znakova provjerom broja njegovog korištenja (jednostruko korištenje) ili njegovim rokom trajanja („*nonce*“ sadrži oznaku vremena, a poslužitelj provjerava da li je istekao), ali obje ove metode su komplekse i uglavnom se ne koriste. Također problem je sigurnost `password` datoteke koja kod *Digest* autentikacije sadrži MD5 sumu korisničkog imena, zaporke i autentikacijske domene. Iako napadač nema direktan uvid u samu zaporku on može iskoristiti pohranjenu MD5 sumu direktno za izračunavanje krajnje MD5 sume i tako si omogućiti neovlašten pristup resursima.
Zbog svega ovog, iako je znatno sigurnija od HTTP autentikacije, *Digest* autentikacija nikad nije dobila na popularnosti, već se češće koriste druge metode. Kao posljedica toga, a možda i kao uzrok, *Digest* autentikacija nije podržana od strane svih web klijenata – za sada je podržavaju sljedeće inačice klijenata: Opera 4.0 ili novija, Microsoft Internet Explorer 5.0 ili noviji, Mozilla 1.0.1 ili novija, Netscape 7 ili noviji.

3.2.1. *Digest* autentikacija na Apache web poslužitelju

Apache web poslužitelj također podržava *Digest* autentikaciju i ona se aktivira u 3 osnovna koraka:

1. kreiranje *password* datoteke,
2. konfiguriranje poslužitelja da koristi *password* datoteku i
3. kreiranje *group* datoteke ako je potrebno.

Kreiranje *password* datoteke je identično kao kod HTTP autentikacije s razlikom što se umjesto `htpasswd` koristi alat `htdigest`. U primjeru to izgleda ovako:

```
htdigest -c /direktorij/security/digest_password realm korisnik1
```

Direktive za aktiviranje *Digest* autentikacije su također identične s razlikom korištenja jednog modula:

```
AuthType Digest
AuthName "Samo lokalni korisnici"
AuthDigestFile /direktorij/security/digest_password
require user korisnik1
AuthAuthoritative on
```

Razlika u odnosu na HTTP autentikaciju je u tome što `htdigest` ne kreira višestruke zapise ako je definiran isti korisnik za više različitih autentikacijskih domena. Rješenje tog problema je u pohranjivanju korisničkih imena i pripadnih zaporki u odvojene datoteke za različite autentikacijske domene.

3.3. Autentikacija pomoću baze podataka

Obje prethodno opisane metode autentikacije koriste datoteke za pohranjivanje korisničkih imena i zaporki. To ne samo da nije siguran način rada, nego može i značajno usporiti rad web poslužitelja posebno u slučaju velikog broja korisnika. Štoviše, korisniku koji posjeduje važeće korisničko ime i zaporku, moguće je odbijanje dozvole pristupa zbog ograničenog vremenskog roka za odgovor, konfiguriranog na web poslužitelju.
Kako bi se riješilo ovo pitanje, većina današnjih web poslužitelja, uključujući i Apache, ima mogućnost pohrane autentikacijskih podataka u bazu podataka. Kako se takva autentikacija konfigurira na Apache web poslužitelju opisano je u sljedećem poglavlju.

3.3.1. Autentikacija pomoću baze podataka na Apache web poslužitelju

Apache web poslužitelj sadrži dva modula za omogućavanje ovog tipa autentikacije, `auth_db` i `mod-auth_dbm` za podršku za dva tipa baza podataka (DB i DBM baza podataka). Za oba tipa baza kreiraju se iste *Database* datoteke dok o platformi ovisi koja će baza biti dostupna. Naredbe za konfiguraciju su identične.
Nijedan od potrebnih modula `auth_db` i `auth_dbm` nisu konfigurirani po standardnim podešenjima pa ih je prije korištenja potrebno konfigurirati upotrebom sljedećih naredbi unutar Apache-ovog osnovnog direktorija:

```
./configure --enable-module=auth_dbm
./configure --enable-module=auth_db
```

Nakon toga potrebno je kreirati datoteku s autentikacijskim podacima i konfigurirati poslužitelj da je koristi. *Database* datoteka se kreira korištenjem `dbmmanage` alata, npr za kreiranje datoteke imena `dbmpasswds` i dodavanje korisnika `korisnik3`:

```
dbmmanage dbmpasswds adduser korisnik3
```

Nakon unosa naredbe od korisnika se traži dvostruko unošenje svoje zaporke koja je potom pohranjena u `dbmpasswds` bazu podataka. Za sve operacije administriranja korisnicima također se koristi `dbmmanage` alat koji ima vrlo jednostavnu sintaksu:

```
dbmmanage bazaPodataka naredba korisnik zaporka
```

`dbmmanage` alat podržava sljedećih 7 naredbi od kojih samo `add` naredba koristi parametar `zaporka`:

- *add* – dodaje novog korisnika i njegovu kriptiranu zaporku u bazu,
- *adduser* – dodaje novog korisnika u bazu te pita za njegovu zaporku,
- *check* – traži zaporku i provjerava da li korisnik postoji u bazi i da li je unesena zaporka ispravna,
- *delete* – uklanja korisnika iz baze,
- *import* – unosi kombinacije korisničkog imena i zaporke u bazu,
- *update* – isto kao i *adduser*, ali pri tome provjerava da korisnik već postoji u bazi,te

- *view* – prikazuje sve zapise iz baze ili samo zapis za specifičnog korisnika.

Ukoliko se na primjer želi prijeći s HTTP autentikacije na autentikaciju pomoću baze podataka moguće je već postojeću `pasword` datoteku jednostavno ukrcati u bazu podataka za autentikaciju:

```
dbmmanage dbmpasswds import < password
```

Direktive za konfiguriranje web poslužitelja u vezi korištenja ove vrste autentikacije su sljedeće:

```
AuthName "Samo korisnici baze podataka"
AuthType Basic
AuthDBMUserFile /direktorij/security/dbmpasswds
require user korisnik3
```

Kao što se može vidjeti administracija autentikacije pomoću baze podataka je nešto kompleksnija od HTTP autentikacije i to je njena glavna zamjerka, ali za obučenog administratora to ne predstavlja problem.

3.4. Anonimna autentikacija

Apache web poslužitelj podržava i anonimnu autentikaciju koja se zasniva samo na korisničkom imenu koje šalje korisnikov web klijent. Za anonimnu autentikaciju kod Apache web poslužitelja koristi se `auth_anon` modul dok se njegovu konfiguraciju mogu koristiti sljedeće direktive:

```
AuthName
AuthType
require
Anonymous_Authoritative
Anonymous <list of users> (No Default)
Anonymous_MustGiveEmail on / off (On is the Default)
Anonymous_VerifyEmail on/off (Off is the Default)
Anonymous_LogEmail on/off (Off is the Default)
Anonymous_NoUserID on/off (Off is the Default)
```

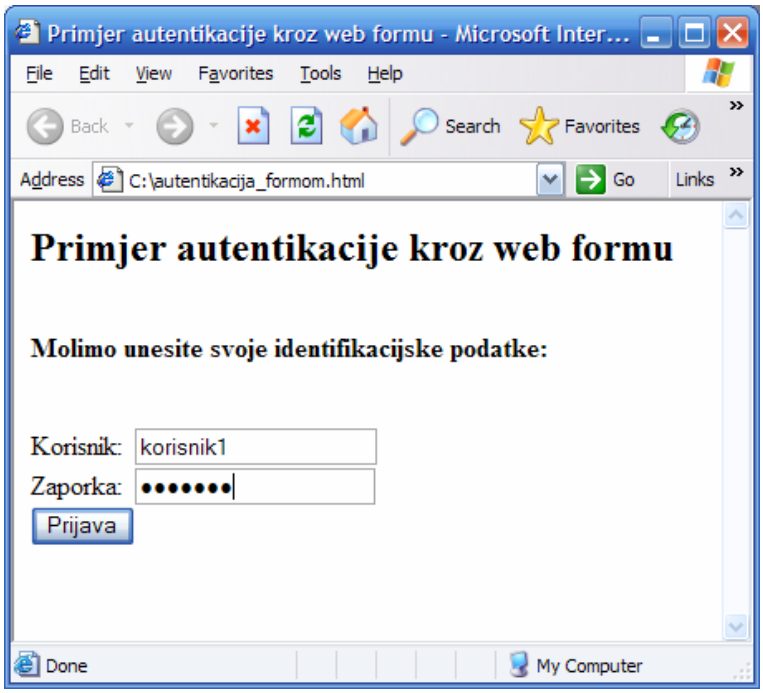
`AuthName`, `AuthType` i `require` se koriste na isti način kao i kod HTTP autentikacije, dok `Anonymous_Authoritative` označava korištenje ili nekorištenje metode anonimne autentikacije. *User*, *password* i *group* datoteke se ne koriste kod anonimne autentikacije. `Anonymous` direktiva sadrži listu korisničkih imena kojima je dozvoljen pristup resursima poslužitelja i ona se može isključiti podešavanjem opcije `Anonymous_NoUserID` na `off` (isključeno).
`Anonymous_MustGiveEmail` direktiva specificira ukoliko zaporka mora biti u obliku adrese elektroničke pošte dok `Anonymous_VerifyEmail` provodi provjeru da li unešena zaporka formatom odgovara adresi elektroničke pošte, ali ne provjerava ukoliko je adresa nevažeća. Informacija o korištenim korisničkim imenima pohranjuje se u poslužiteljevoj log datoteci, uobičajeno u `access_log` datoteci, pri čemu mora biti direktiva `Anonymous_VerifyEmail` podešena na `on` (uključeno)..
Primjer konfiguracije anonimne autentikacije koji dozvoljava pristup web klijentima `korisnik1` i `korisnik2` uz korištenje adrese elektroničke pošte kao zaporke prikazan je u nastavku:

```
AuthName "Za anonimne goste"
AuthType Basic
Anonymous korisnik1 korisnik2
require valid-user
Anonymous_Authoritative on
Anonymous_MustGiveEmail on
Anonymous_VerifyEmail on
Anonymous_LogEmail on
Anonymous_NoUserID off
```

Glavna zamjerka ove autentikacije je nedostatak provjeravanja ukoliko je unesena adresa elektroničke pošte važeća ili ne, te pamćenje samo uspješnih pristupa. Također zbog vjerojatno velikog broja korisnika koji bi morali biti navedeni u direktivi kao *guest* korisnici, u većini slučajeva zahtjev za unošenjem bilo kakvih podataka neće biti uključen što samo po sebi predstavlja sigurnosni problem.

4. Autentikacija pomoću web forme

Pošto osnovne metode autentikacije nekim korisnicima nisu zadovoljavajuće, dizajnerima web poslužiteljskih aplikacija ostavljena je sloboda vlastite implementacije. Jedna od takvih implementacija je autentikacija putem web formi opisana u sljedećim poglavljima.
Web forme su generički mehanizam za unos podataka preko web stranica i podržane su od gotovo svih web klijenata. Uobičajeno web klijenti ne obrađuju unesene podatke već ih samo prosljeđuju web aplikaciji. Web forme podržavaju *password* tip unosa koji maskira zaporku prilikom unosa pa ista nije vidljiva na korisnikovom zaslonu kao na primjeru na slici.



Slika 2: Autentikacija pomoću web forme na strani klijenta

Prednost ove metode je fleksibilnost – web dizajnerima je ostavljena potpuna sloboda prilikom izbora izgleda forme i dodatnih funkcionalnosti koje ona pruža. Nedostatak je u tome što web dizajner mora sam voditi brigu i o provedbi autentikacije jer web poslužitelj u ovom slučaju ne pruža nikakvu podršku. Obično se autentikacija provodi unutar CGI skripte čija izrada je podložna i unosu pogreški koje pak mogu predstavljati prijetnje po sigurnost sustava.

4.1. Implementacija autentikacije pomoću web formi

HTML kod za implementaciju autentikacije pomoću web formi može biti prilično jednostavan kao što je vidljivo iz sljedećeg primjera:

```
<form>
<input type="text" name="korisnicko_ime">
<input type="password" name="zaporka">
<input type="submit" value="Prijava">
</form>
```

Uobičajeno se za unos korisničkog imena specificira „*text*“ kao tip polja za unos, dok se za unos zaporke koristi „*password*“ tip polja za unos. Sigurnost se može povećati ako se za naziv polja ne

upotrijebe riječi tipa „*username*“ i „*password*“ jer će potencijalni napadači najčešće tražiti baš te riječi prilikom praćenja Internet prometa.
Osim toga potrebno je voditi brigu i o još nekoliko parametara web forme:

- `ACTION` parametar forme za autentikaciju mora sadržavati ime CGI skripte koja će provesti provjeru autentikacijskih parametara,
- `METHOD` parametar forme mora obavezno biti postavljen kao `POST`. Ako nije bit će korištena standardna metoda `GET` što ima za posljedicu sljedeće:
    - Mnogi web poslužitelji će zapisivati argumente `GET` formi, ali neće zapisivati argumente `POST` formi. Bilo bi vrlo loše kad bi se u log zapisima web poslužitelja našle korisničke zaporke (zapisane u tekstualnom obliku) svih korisnika autenticiranih putem web forme.
    - Argumenti `GET` forme prenose se u CGI skriptu za autentikaciju kao `QUERY_STRING` varijabla. Na većini Unix sustava bilo koji korisnik može vidjeti sadržaj `QUERY_STRING` varijable za bilo koji aktivni proces jednostavnom upotrebom naredbe kao što je „`ps -auxwwwe`“. Čak i ako su svi korisnici Unix sustava povjerljive osobe nije nikako poželjna situacija u kojoj su korisničke zaporke tako lako vidljive. Upotrebom `POST` forme argumenti se prenose u CGI skriptu kroz cjevovod (eng. *pipe*) koji je znatno zaštićeniji.
    - Argumenti `GET` forme će biti vidljivi unutar „*Location*“ dijela zaslona prilikom prikaza sljedeće web stranice na korisnikovom web klijentu što definitivno nije poželjno.

Također, važno je osigurati da korisnikov web klijent ne pohrani web stranicu s autentikacijskim podacima u svoju privremenu memoriju jer se time omogućava nekom sljedećem korisniku koji koristi isti web klijent, iskorištavanje pohranjene stranice za pristup web poslužitelju korištenjem opcije povratka (eng. *back*) web klijenta. Kako bi se spriječilo pohranjivanje stranice u privremenu memoriju web klijenta, potrebno je upotrijebiti sljedeća HTML zaglavlja prilikom odgovora na web formu:

```
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Pragma: no-cache
Cache-Control: no-cache
```

Nijedna od ove 3 opcije korištena samostalno ne sprječava pohranjivanje podataka u privremenu memoriju, ali sve 3 zajedno uglavnom to osiguravaju kod većine web klijenata. Isto se može postići i upotrebom HTML META oznaka (npr. `<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">`), ali različiti web klijenti će ih različito interpretirati, a neki čak i potpuno ignorirati.
Poželjno je da stranica koja sadrži formu bude što jednostavnija kako bi se skratilo vrijeme njenog ponovnog učitavanja, a ako se već želi staviti više informacija na istu stranicu za autentikaciju s web formom, onda je potrebno formu odvojiti u poseban okvir i podesiti za samo taj okvir, ali ne i za cijelu stranicu, nepohranjivanje u privremenu memoriju.
Noviji web klijenti također omogućavaju *autocomplete* funkcionalnost kako korisnik ne bi morao svaki put upisivati autentikacijske podatke kod ponovljenog posjeta stranici. Ovo predstavlja pogodnost za korisnika, ali također i sigurnosni rizik pa se zbog toga preporuča specificiranje `autocomplete=off` opcije za web formu:

```
<form action="login.cgi" method="post" autocomplete="off">
```

Ovo će onemogućiti *autocomplete* funkcionalnost kod većine web klijenata, a povrh toga to se može spriječiti variranjem URL-a stranice koja sadrži web formu (npr. dodavanjem dodatne informacije o poziciji stranice - *path*) ili variranjem imena polja za autentikaciju (korisničko ime može biti slučajno generirana riječ koja počinje slovom „k“, a zaporka slučajno generirana riječ koja počinje slovom „z“).

4.1.1. Identifikator sesije

Nakon unosa autentikacijskih podataka u web formu, oni se prenose na validaciju na web poslužitelj i to u obliku nekriptiranog teksta baš kao kod HTTP autentikacije. Na strani poslužitelja CGI skripta verificira ukoliko su podaci ispravni i ako jesu korisniku se šalje odgovor kojim ga se preusmjerava na glavnu/sljedeću stranicu. Tu se pojavljuje problem – ako korisnik sad pokuša ponovo učitati stranicu na kojoj se nalazi, neće uspjeti jer njen URL više ne sadrži autentikacijske podatke i operacija odabira tog URL-a nije identična operaciji prijave. U tom trenutku korisnikov web klijent više ne zna autentikacijske podatke i ne šalje ih web poslužitelju. To je posljedica HTTP protokola koji je protokol bez stanja (eng. *stateless*) i koji ne omogućava klijentu pamćenje činjenice o tome kako je već autenticiran s određenim autentikacijskim podacima. Web klijent nema mogućnosti povezivanja jedne web stranice s drugom – to mora odraditi web poslužiteljska aplikacija.
Jedan način za povezivanje web stranica je uključivanje korisničkog imena u URL stranice, što znači da se URL-ovi moraju generirati dinamički kao u sljedećem primjeru:

```
<a href="transakcije.html?korisnik=pero">Transakcije</a>
<a href="detalji.html?korisnik=pero">Osobni podaci</a>
```

Na taj način URL svake stranice uključuje i korisničko ime i time daje do znanja web poslužitelju da je korisnik autenticiran. Ipak, ovakva implementacija ima veliki nedostatak – svakom korisniku je vidljiv URL stranice i korisničko ime unutar URL-a. To znači da bilo koji korisnik može ručno promijeniti URL i umjesto svog korisničkog imena ubaciti neko drugo i tako ostvariti neovlašteni pristup.
Da bi se izbjegli ovakvi problemi umjesto korisničkog imena kao veza između web stranica može se koristiti neki drugi identifikator korisnika i to u kriptiranom obliku. Preporuka je da se kao identifikator koristi niz znakova dobiven konkatenacijom sljedećih parametara:

- naziv usluge i poslužitelja kojem se pristupa,
- korisničko ime,
- IP adresa web klijenta i
- trenutno vrijeme i datum.

Dobiveni niz znakove se još potom enkriptira pomoću tajnog ključa pohranjenog na web poslužitelju i to onda postaje korisnikov certifikat ili identifikator sesije (eng. *session ID*) koji mu se šalje kao dio URL-a sljedeće stranice.

```
transactions.html?sid=9c4d81a96351ab84e5c637f349a324ca
```

Kad web poslužitelj primi zahtjev s novim URL-om on može dekriptirati identifikator sesije i utvrditi da li on sadrži ispravne podatke ili ne, te na taj način očuvati sigurnost sustava. Ako se još uz to ugradi i vremenska provjera nad vrijednošću identifikatora sesije, sprečava se i neovlaštena upotreba isteklih identifikatora sesija, npr. korisnik koji izgubi pravo pristupa ne može koristiti stari identifikator sesije.
Problemi ovakve implementacije veza između stranica mogu biti sljedeći:

- sigurnost ovisi o tajnom ključu koji je pohranjen na web poslužitelju i ako se netko domogne ključa sigurnost je ugrožena i
- upotreba jednosmjerne enkripcije koja nije svugdje zakonom dozvoljena za upotrebu.

Rješenje je u upotrebi dvosmjerne enkripcije gdje se konkateniraju sljedeći parametri:

- naziv usluge i poslužitelja kojem se pristupa,
- korisničko ime,
- IP adresa web klijenta,
- trenutno vrijeme i datum i
- tajni ključ poslužitelja.

Od dobivenog niza znakova računa se MD5 suma i ona se zajedno s tekstualnim zapisom korisničkog imena i trenutnog vremena uključuje u URL web stranice kao identifikator sesije.
Kad web poslužitelj primi sljedeći zahtjev on iz dobivenih podataka može ponovo izračunati MD5 sumu i usporediti je s onom koju je generirao – ako se poklapaju radi se o autenticiranom korisniku i pristup je omogućen. Kako bi cijela ova operacija bila brža, generirane MD5 sume autenticiranih korisnika se pohranjuju u bazu podataka na web poslužitelju radi bržeg pristupa.

4.1.2. Web kolačići

Web kolačići (eng. *cookies*) su alternativa za pohranjivanje identifikatora sesije na strani klijenta. Naime, nije dobra praksa pohranjivati identifikator sesije u nešto što je svima vidljivo kao URL stranice pa se on pohranjuje na strani klijenta unutar web kolačića. Prvo web poslužitelj podesi web kolačić za trenutnu sesiju i putem *Set-Coookie* zaglavlja prenese ga klijentu koji ga pohranjuje. Web kolačić se sastoji od imena i slučajno generiranog niza znakova, a opcionalno može sadržavati još i naziv autentikacijske domene, relevantnu putanju i rok trajanja. Nakon toga kad god pristupa stranici čiji URL je dio dane autentikacijske domene i resursu identificiranom podacima o putanji, web klijent automatski u zahtjev uključuje i odgovarajući web kolačić. Time se izbjegava ubacivanje identifikatora sesije u svaki URL za koji je potrebna autentikacija te se unatoč tome postiže zadovoljavajući nivo sigurnosti.

Nedostatak korištenja web kolačića je njihova izloženost pošto se podaci iz web kolačića šalju uz svaki HTTP zahtjev i napadači mogu praćenjem prometa dobiti uvid u njihov sadržaj. Očito rješenje u ovakvim slučajevima bi bilo korištenje enkripcijskog protokola, kao na primjer SSL protokola za komunikaciju, ali čak ni to ne pruža potpunu zaštitu. Naime, ako web poslužitelj uz zaštićene stranice za koje koristi web kolačiće ima i stranice kojima se pristupa putem običnog HTTP protokola, web klijent će prilikom pristupa nezaštićenim stranicama u zahtjev uvrstiti nekriptirani web kolačić i time ga izložiti napadačima. U pravilu to dizajneri web poslužiteljskih aplikacija ne rade, ali zato zlonamjerni korisnici znaju simulirati web stranice kojima navede korisnike na slanje vlastitih web kolačića, npr. simuliranje originalnih web stranica s ubačenom slikom s lažnim URL-om za krađu web kolačića, slanje poruke elektroničke pošte s URL-om originalne web stranice, ali s `http` prefiksom umjesto `https`, itd…

Osim navedenih problema, web kolačići imaju lošu reputaciju među korisnicima pa mnogi od njih u svojim web klijentima onemogućavaju slanje web kolačića radi vlastite zaštite. Međutim, to je prilično bezrazložno jer korištenje web kolačića ili pohranjivanje identifikatora sesije u URL su u biti različite implementacije identične zaštite. Nijedna od tih metoda ne daje savršenu zaštitu, ali nijedna od njih nije bitno lošija od druge.

5. Zaključak

Autentikacija na web poslužiteljima se može implementirati na jednostavniji ili složeniji način, pri čemu je jedino sigurno kako nijedna implementacija ne osigurava potpunu zaštitu. Za web poslužitelje koji nemaju visoke sigurnosne zahtjeve dovoljno je u većini slučajeva implementirati HTTP autentikaciju i osigurati kod korisnika korištenje tzv. „jake“ zaporke te ih obrazovati u vezi *phishing* napada krađe zaporki.

Za web poslužitelje s visokim sigurnosnim zahtjevima preporuča se implementacija autentikacije pomoću web formi i identifikatora sesije ili web kolačića preko kriptiranog SSL kanala. Pri tome se za dodatno povećanje sigurnosti preporuča i sljedeće:

- generirati 128 bitni kriptografski otporan identifikator sesije radi sprečavanja tzv. „*brute force*“ napada razbijanja vrijednosti identifikatora sesije,
- koristiti „*secure*“ i „*HTTPOnly*“ opcije za web kolačiće za sprečavanje krađa web kolačića kroz XSS (eng. *Cross Site Scripting*) napade te gubitak podataka kroz „*non-SSL*“ zahtjeve koji ne koriste SSL protokol,
- onesposobiti „*TRACE/TRACK HTTP*“ metode za sprečavanje zaobilaženja „*HTTPOnly*“ opcije,
- mijenjati identifikator sesije u vrijeme autentikacije ili svaki put generirati novi za sprečavanje „*session fixation*“ napada krivotvorenja identifikatora sesije,
- osigurati kod svih zahtjeva koji uzrokuju izmjenu podataka na poslužitelju, korištenje slučajnog autorizacijskog tokena,
- implementirati funkciju odjave s web stranica (eng. *log out*) kojom se poništava identifikator sesije na web poslužitelju,
- staviti vremensko ograničenje na identifikatore sesije,
- razdvojiti velike web poslužiteljske aplikacije na više autentikacijskih domena,
- osigurati upotrebu tzv. „jakih“ zaporki,
- implementirati zapisivanje neuspješnih pokušaja autentikacije, itd…

Na kraju treba napomenuti da unatoč svim metodama autentikacije najveći dio odgovornosti ipak obično leži na samim korisnicima koji su najčešći uzrok sigurnosnih propusta. Zbog toga puno pažnje treba uložiti na obrazovanje korisnika i uvođenje procedura kojima će se osigurati da se sami korisnici pridržavaju pravila koja njima samima garantiraju veću sigurnost.

6. Reference

[1] Authentication and Session Management on the Web, Paul Johnston, GIAC Security Essentials Certification Practical Assignment Version 1.4b, studeni 2004.

[2] A Guide to Web Authentication Alternatives, Jan Wolter, http://unixpapa.com/auth/index.html , svibanj 2006.

[3] Apache web poslužitelj, http://www.apache.org, svibanj 2006.

[4] Web Authentication Security, Donna Selman, lipanj 2003.