Sigurnost web aplikacija

Gosarić, Goran

Master's thesis / Diplomski rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:867532

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-11-26**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Goran Gosarić

SIGURNOST WEB APLIKACIJA

Diplomski rad

Voditelj rada:
prof. dr. sc. Robert Manger

Zagreb, rujan, 2016.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvaljujem se svom mentoru prof. dr. sc. Robertu Mangeru na ukazanom strpljenju, pomoći i podršci prilikom izrade ovog diplomskog rada.

Sadržaj

Uvod

Za razliku od svojih početaka, *World Wide Web* postao je gotovo neprepoznatljiv. Većina web stranica zapravo su aplikacije koje obiluju razno raznim funkcionalnostima i omogućuju bolju interakciju s korisnicima. Sadržaj samih aplikacija približen je individualnosti korisnika te obično nastaje procesuiranjem brojnih osjetljivih informacija. Stoga, ni ne čudi činjenica da je sigurnost web aplikacija neizostavan dio razvoja i održavanja svake web aplikacije. Napadi na web aplikacije događaju se svakodnevno. Gotovo u tolikoj mjeri da smo svjedoci da samo najveće povrede sigurnosti privlače dovoljno pažnje, dok se i dalje većina manjih napada prikriva te rješava u situacijama kada su tek otkriveni. Ako dobro promotrimo moderno društvo u kojem živimo te količinu informacija i financijskih sredstava koji su svakodnevno ugroženi od potencijalnih napadača, shvatit ćemo važnost razvoja i korištenja sigurnih web aplikacija.

Glavni cilj ovog diplomskog rada "Sigurnost web aplikacija" jest opis sigurnosnih principa koji se koriste za siguran rad web aplikacija, klasifikacija i opis napada u vezi s pojedinim principima te opis koraka zaštite koje je potrebno poduzeti u slučaju određene vrste napada. Dodatno, bit će opisane dvije metode razvoja sigurnih web aplikacija, koje omogućuju da se pojam sigurnosti i oprez od mogućih propusta ugrade u same temelje postupka razvoja web aplikacija.

Glavni izvori podataka korišteni kao podloga za pisanje ovog diplomskog rada su knjige: *Web Application Security, A Beginner's Guide* i *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*. Ostatak literature korišten je u svrhu dopune pojedinih dijelova radnje.

Ovaj diplomski rad strukturiran je u pet poglavlja. U prvom poglavlju rada opisani su sigurnosni principi ili mehanizmi (kao što se spominje u nekim knjigama) koji su sastavni dio sigurnosti web aplikacije. Najprije se opisuju sigurnosni principi autentifikacija i autorizacija. Nakon toga slijede još i opis upravljanja sesijom te princip sigurnog rada baze podataka. Na kraju poglavlja dan je opis dva pravila koja su temelj principa sigurnog rada web preglednika.

Drugo poglavlje rada daje opis i primjere napada u vezi s pojedinim sigurnosnim principima. Najprije su opisani napadi vezani uz autentifikaciju. Zatim slijede napadi vezani uz autorizaciju, napadi vezani uz upravljanje sesijom te napadi na baze podataka. Poglavlje

završava opisom i primjerima najčešćih napada na web preglednike.

Treće poglavlje rada slično je organizirano kao i prethodno poglavlje, međutim umjesto opisa napada, opisuje se način obrane i koraci zaštite od napada na pojedine sigurnosne principe.

Četvrto poglavlje opisuje dvije važne metode razvoja sigurnih web aplikacija. Najprije opisujemo metodu *SDL* tvrtke Microsoft a potom metodu organizacije OWASP.

Zadnje, odnosno peto poglavlje, sadrži studijski primjer.

Poglavlje 1

Sigurnosni principi

U ovom poglavlju opisani su sigurnosni principi koji su sastavni dio obrane sigurnosti web aplikacije. Najprije se opisuje princip autentifikacije gdje je opisan sam proces autentifikacije, zatim su opisani glavni primjeri autentifikacijskih mehanizama te je opisano mjesto i vrijeme upotrebe ovog mehanizma. Nakon autentifikacije opisuje se autorizacija, odnosno sam proces autorizacije, slojevi autorizacije te razvoj mehanizma autorizacije. Nakon autorizacije slijede još i opis mehanizama upravljanja sesijom te opis principa sigurnog rada baze podataka. Na kraju poglavlja, opisana su dva važna pravila postignuta dogovorom proizvođača web preglednika koja su zaslužna za siguran rad web preglednika.

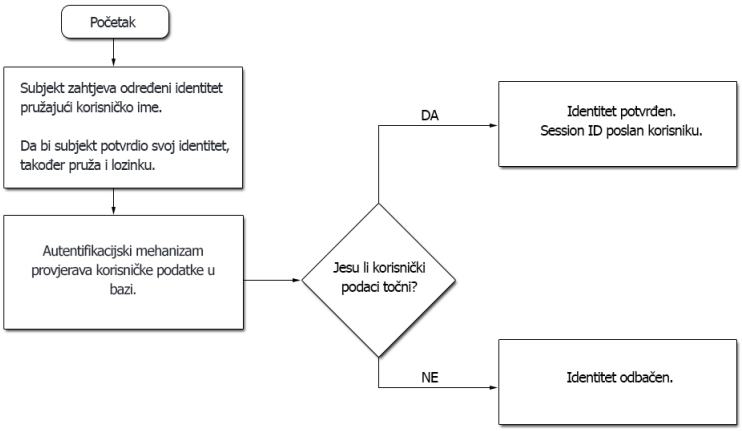
1.1 Autentifikacija

Općenito o autentifikaciji

Ubrzanim razvojem web aplikacija mnoge od njih zahtijevaju određenu kontrolu pristupa zaštićenim podacima i uslugama. Zahtijevanje takve kontrole podrazumijeva činjenicu da web aplikacija mora imati dobar sustav za kontrolu pristupa. Takav sustav možemo definirati kao mehanizam koji regulira pristup podacima ili uslugama web aplikacije na način da utvrđuje ima li subjekt dozvolu za to. Da bi mogao to utvrditi, sustav se oslanja na dva procesa, autentifikaciju i autorizaciju.

Autentifikacija (engl. *Authentification*) ili kraće AuthN je proces u kojem subjekt dokazuje da je ono što zapravo tvrdi. Taj proces se sastoji od dva koraka: identifikacije i potvrde. *Identifikacija* omogućuje subjektu da tvrdi da je određeni subjekt, dok *potvrda* omogućuje da tu tvrdnju potvrdi. [15]

Na primjer, ako pristupamo web aplikaciji za studentski mail, pretpostavimo da autentifikacija zahtijeva korisničko ime i lozinku. Unos korisničkog imena predstavlja korak



Slika 1.1: Skica jednostavnog procesa kontrole pristupa

identifikacije, dok korak potvrde predstavlja unos lozike. Dakle, uz pretpostavku da samo određeni subjekt zna korisničko ime i lozinku, tada je taj subjekt identificiran.

Postoje tri klase faktora koje se upotrebljavaju za dokazivanje identiteta:

- *„nečeg što znaš“*,
- *„nečeg što imaš“*,
- *„nečeg što jesi“*.

Prvu klasu autentifikacijskih faktora, odnosno „nečeg što znaš“ najčešće čine lozinka ili PIN. Drugu klasu čine digitalni potpis, pametna kartica ili sigurnosni žeton (engl. *token*). Najčešći faktor koji se koristi iz druge klase je sigurnosni žeton jer ne zahtjeva neki specijalni hardver koji korisnik mora imati. Treću klasu čine otisak prsta, geometrija ruke ili topografija lica. Takve faktore često poistovjećujemo s biometrijom, jer ovise o fizičkim i karakternim osobinama osobe. No, napadači postaju sve uporniji u pokušajima provale ili obilaska autentifikacije, stoga mnoge aplikacije koriste faktore iz dvije ili tri kategorije. Kao primjer toga, možemo navesti aplikaciju za internet bankarstvo koja koristi sigurnosni žeton te lozinku, odnosno faktore iz klasa „nečeg što imaš“ i „nečeg što znaš“. Korisničko ime te lozinka predstavljaju standard za autentifikaciju, međutim u određenim okolnostima, gdje je potrebna viša razina sigurnosti upotrebljavaju se dodatno hardver ili sigurnosni žetoni.

Posljednji korak procesa autentifikacije predstavlja validacija korisničkih podataka. Postoji četiri načina validacije s obzirom na lokaciju provjere podataka te u kakvom obliku je lozinka spremljena:

- provjera unutar aplikacije pomoću nešifrirane lozinke,
- provjera unutar baze podataka pomoću nešifrirane lozinke,
- provjera unutar aplikacije pomoću šifrirane lozinke,
- provjera unutar baze podataka pomoću šifrirane lozinke. [15]

Vrlo je važno shvatiti načine na koje se provodi validacija korisničkih podataka jer na temelju toga možemo utvrditi kako napadi na aplikaciju funkcioniraju.

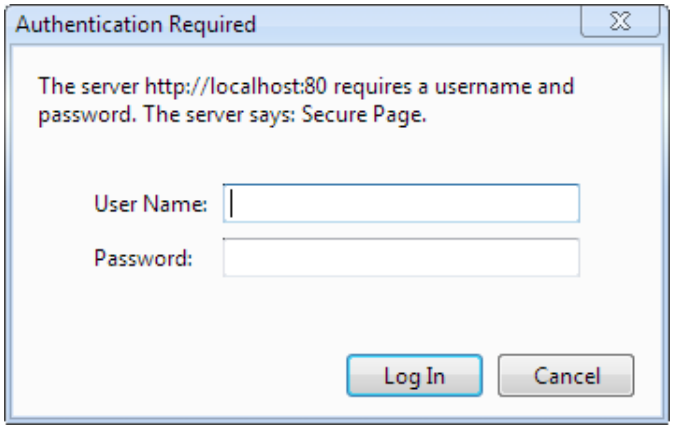
Primjeri autentifikacijskih mehanizama

U ovom dijelu poglavlja govorit ćemo o primjerima autentifikacijskih mehanizama koji se baziraju na upotrebi korisničkog imena i lozinke. U cilju potpunosti, najprije ćemo spomenuti najjednostavniji ali i vrlo nesiguran mehanizam koji nam pruža HTTP protokol. Nadalje, govorit ćemo o autentifikaciji na jednom mjestu koja postaje sve popularnija te na kraju o autentifikacijskim mehanizmima zasnovanim na HTML formama, a koji su trenutno najraširenij u upotrebi.

HTTP autentifikacijski mehanizmi

Najjednostavnije autentifikacijske mehanizme nudi nam HTTP. Osigurava nam *Basic access* i *Digest access* autentifikacijske mehanizme. Međutim zbog brojnih mana, njihova upotreba nije preporučljiva. *Basic access* autentifikacijski mehanizam je mehanizam autentifikacije koji zahtijeva od korisnika da unese korisničko ime te lozinku prije nego li pristupi zaštićenom resursu na web poslužitelju. U nastavku opisujemo proces autentifikacije korisnika pomoću tog mehanizma:

- Proces počinje kada korisnik želi pristupiti zaštićenom resursu. Web poslužitelj šalje korisniku *401 Authorization Required* kod.
- Kada web preglednik dobije taj odgovor, izbacuje dijalog koji zahtjeva od korisnika korisničko ime te lozinku (slika 1.2).
- Nakon što korisnik unese svoje podatke i klikne gumb *OK*, web preglednik konkatenira korisničko ime, znak ":" te lozinku. Takvu vrijednost šifrira u *base64* notaciju i šalje putem GET metode web poslužitelju unutar HTTP zaglavlja.
- Ako poslužitelj prihvati podatke, tada korisnik ima pristup zaštićenom resursu, dok u protivnim korisnik dobija status pogreške.



Slika 1.2: HTTP *basic access* autentifikacija

Ova metoda autentifikacije ima brojne mane: nesiguran prijenos podataka, ponovljena izloženost riziku te nesigurno spremanje podataka od strane preglednika. *Digest access* autentifikacijski mehanizam je vrlo sličan prethodnom mehanizmu, no koristi MD5 algoritam za haširanje lozinke prije nego li se šalje. Ovaj mehanizam podložan je riziku „čovjeka u sredini“, te je lako moguće smanjiti sigurnost na *Basic access* autentifikacijski mehanizam ili neku stariju veziju *Digest access* autentifikacije. [15]

Autentifikacija na jednom mjestu

Autentifikacija na jednom mjestu (engl. *Single Sign-on*) je proces autentifikacije koji omogućava korisniku da predoči svoje akreditacijske podatke samo jednom kako bi mogao pristupiti različitim resursima ili aplikacijama. Na temelju prethodnog, možemo reći da autentifikacija na jednom mjestu predstavlja dijeljenje autentifikacijskih podataka. [13]

Takav autentifikacijski mehanizam upotrebljava malu količinu podataka koje poslužitelj sprema na klijentsko računalo putem preglednika. Obično takvu vrstu informacija nazivamo *kolačić* (engl. *cookie*). Dakle, prilikom prve prijave na klijentsko računalo se sprema kolačić koji sadrži korisničko ime. Za svaki zahtjev, poslužitelj može iz pohranjenog kolačića dohvatiti ime te odlučiti što dalje.[13] Najpoznatiji primjeri mehanizama za autentifikaciju na jednom mjestu su *Google Accounts* (vidi sliku 1.3), *Facebook Login*, *Windows Live ID*.

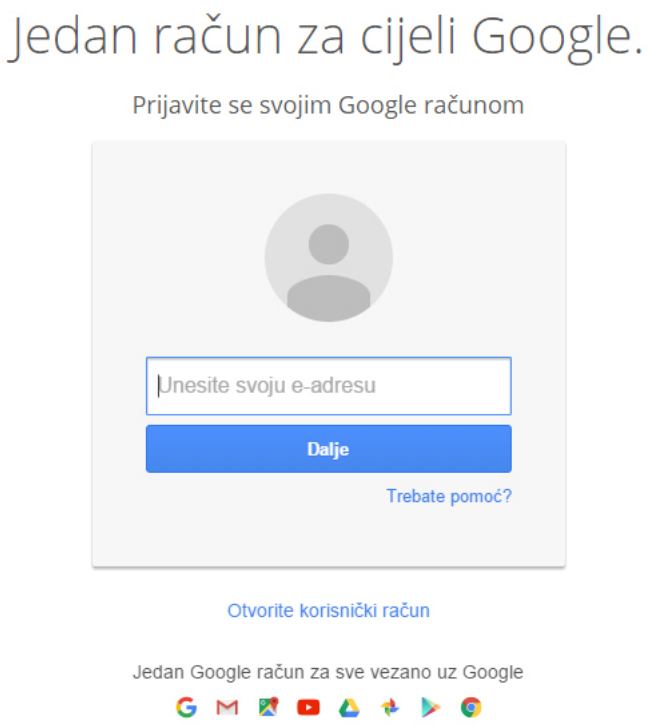
Prednosti koje donosi korištenje autentifikacije na jednom mjestu su:

- korisnik unosi svoje podatke na jednom mjestu,
- jedinstvena baza sigurnosnih podataka,

- korisnik ne mora pamtiti gomilu korisničkih podataka potrebnih za prijavu u različite web aplikacije,
- smanjenje troškova,
- jedinstveno autentifikacijsko sučelje.

Međutim, moramo spomenuti i neke od mana koje donosi korištenje autentifikacije na jednom mjestu:

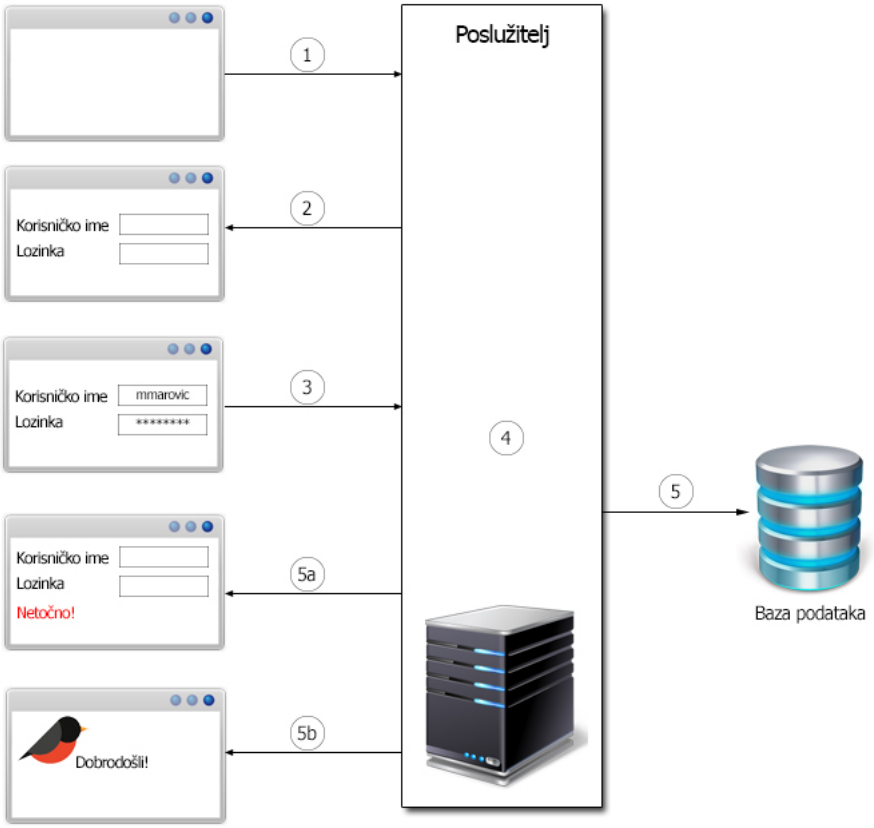
- promjena postojećih aplikacija može biti komplicirana, dugotrajna i skupa,
- ako se korisnik uspješno prijavi te napusti računalo, svi autorizirani resursi mogu biti iskorišteni,
- dovoljno je napasti jednu točku.



Slika 1.3: *Google accounts SSO*

Autentifikacijski mehanizam zasnovan na HTML formama

Kako su HTTP autentifikacijski mehanizmi vrlo nesigurni, a autentifikacija na jednom mjestu još ne predstavlja standard, aplikacije vrlo često koriste mehanizme za autentifikaciju korisnika koje programeri kreiraju koristeći HTML forme. U nastavku opisujemo



Slika 1.4: Prikaz jednostavnog procesa autentifikacije

korake (vidi sliku 1.4) kako funkcionira mehanizam za autentifikaciju koji se koristi u većini aplikacija:

1. Korisnik putem web preglednika šalje zahtjev web aplikaciji za stranicom za prijavu.
2. Aplikacija vraća pregledniku stranicu za prijavu korisnika.
3. Korisnik unosi svoje korisničke podatke u HTML forme, te ih šalje putem POST metode web aplikaciji.

4. Web aplikacija analizira informacije koje je prihvatila.
5. Aplikacija radi upit prema bazi podataka u svrhu provjere podataka.
   (a) Ako na temelju upita nema poklapanja, aplikacija šalje korisniku status pogreške sa stranicom za prijavu.
   (b) Ako na temelju upita imamo poklapanje, aplikacija uspostavlja novu sesiju (engl. *session*) za korisnika. U većini slučajeva to podrazumijeva generiranje i slanje jedinstvenog identifikatora sesije (engl. *session ID*) korisniku.
6. Nakon što preglednik dobije odgovor, unutar HTTP zaglavlja iz direktive *Set-Cookie* spremit će identifikacijsku vrijednost. Kako je identifikacijska vrijednost postavljena u kolačiću, preglednik će ju automatski koristiti za sve zahtjeve prema web aplikaciji. Na taj način aplikacija autorizira korisnika za svaki zahtjev. [15]

Mjesto i vrijeme autentifikacije

Važno pravilo koje se nameće u procesu autentifikacije glasi da se ona provodi na svaki zahtjev za pristupom zaštićenom dijelu aplikacije. Općenito, autentifikaciju moramo primjeniti u sljedeća tri slučaja:

- u slučaju promjene korisničkih prava pristupa,
- sa svakim zahtjevom za zaštićenim djelom aplikacije,
- za svaki pristup aplikaciji izvan nje.

1.2 Autorizacija

U prethodnom poglavlju spomenuli smo da aplikacija mora imati dobru kontrolu pristupa kako bi pristup zaštićenim podacima i uslugama bio zaštićen. U ovom poglavlju govorit ćemo o terminologiji i metodologiji vezanoj uz autorizaciju, odnosno drugom važnom procesu koji nastupa nakon autentifikacije.

Općenito o autorizaciji

Autorizacija (engl. *Authorization*) ili kraće „AuthZ“ je proces utvrđivanja ima li subjekt dozvolu za izvođenje određenih operacija na zaštićenom resursu. Koristimo pojam *subjekta* jer osim osoba, subjekti mogu biti i web servisi, baze ili druga računala. *Resurse*

nam predstavljaju zaštićeni podaci ili funkcionalnosti koje subjekt želi korisiti. Autorizaciju promatramo kao složen pojam koji ovisi o individualnosti same web aplikacije, pa se dizajn i implementacija razlikuju od aplikacije do aplikacije. [15]

Autorizacija ima tri glavna cilja:

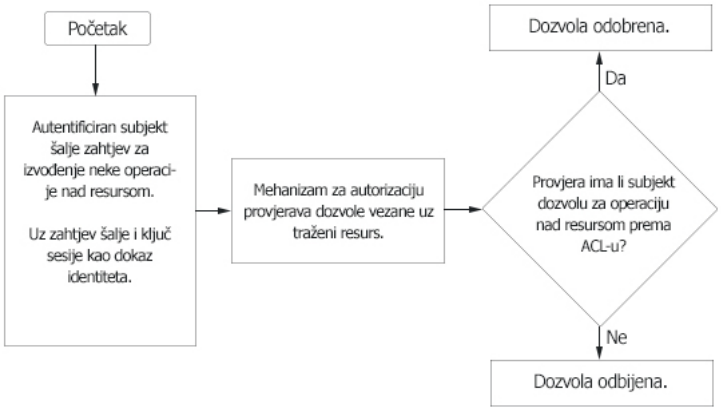
1. Osigurati da subjekt izvodi samo dozvoljene akcije.
2. Kontrolirati pristup zaštićenim resursima ovisno o ulozi subjekta te njegovim privilegijima.
3. Umanjiti rizike od napada vezanih uz privilegije.

Kada aplikacija utvrdi tko je subjekt i kojem resursu želi pristupiti, tada mora odrediti je li tražena operacija dozvoljena. Stoga aplikacija mora imati određena pravila po kojima može to odrediti. Postoje tri vrste modela kontrole pristupa koji određuju dodjelu pravila:

- *Role-Based Access Control (RBAC)* - model koji se bazira na uporabi *uloga* (engl. *role*) kojoj pripada grupa subjekata. Svakom subjektu je određena jedna ili više uloga, dok svaka uloga dopušta određene operacije. Tada se na temelju uloge kojoj subjekt pripada, dodjeljuju dozvole za pristup zaštićenom resursu.
- *Discretionary Access Control (DAC)* - model koji dozvoljava korisnicima ili administratoru da definira *listu kontrole pristupa* (engl. *Access control list, ACL*) na nekom zaštićenom resursu. Lista kontrole pristupa sadrži stavke koje određuju koji korisnik smije pristupiti resursu te koje operacije smije raditi nad tim resursom.
- *Mandatory Access Control (MAC)* - model koji koristi strogu klasifikaciju. Kontrolu pristupa određuju sistem ili administrator. Vrlo slabo se upotrebljava, uglavnom za vojne sustave. [15]

Odluka o odabiru modela donosi se prilikom dizajna aplikacije. Opisujemo bitne razlike između modela RBAC i DAC:

- U DAC modelu pravila su pridružena određenom resursu, dok u RBAC modelu pravila pridružujemo određenoj grupi korisnika.
- Pravila unutar RBAC modela su obično statična, dok se unutra modela DAC često mijenjaju.
- Model DAC se bazira na osobnim pravilima, dok model RBAC koristi grupna pravila.
- Tipično pitanje unutar model RBAC je: ”Što može raditi ovaj korisnik?”, dok DAC postavlja pitanje: ”Tko ima pristup mojem resursu?”. [8]



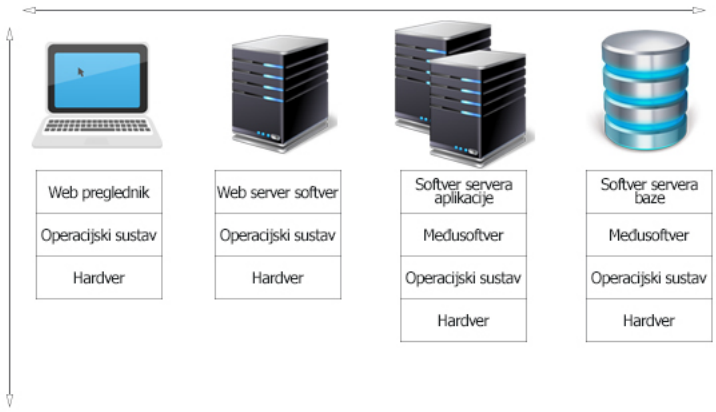
Slika 1.5: Model autorizacije

Na kraju, nakon što smo opisali modele kontrole pristupa te na koji način definiraju pravila na temelju kojih se provodi autorizacija, preostaje nam podjela dozvola. Postoje tri vrste dozvola koje se primjenjuju ovisno o resursu:

- *Dozvola čitanja* (engl. *Read access*) - dozvola čitanja odnosi se na podatke. Pitanje na koje autorizacija mora dati odgovor jest: ”Smije li ovaj korisnik vidjeti ove podatke?”.
- *Dozvola pisanja* (engl. *Write access*) - dozvola pisanja odnosi se također na podatke, u svrhu promjene. Pitanje na koje autorizacija mora dati odgovor jest: ”Smije li ovaj korisnik promjeniti ove podatke?”.
- *Dozvola izvršavanja* (engl. *Execute access*) - dozvola izvršavanja odnosti se na funkcionalnost unutar aplikacije. Pitanje na koje autorizacija mora dati odgovor jest: ”Smije li ovaj korisnik pokrenuti ovu operaciju?”. [15]

Slojevi autorizacije

Proces autorizacije provodi se na brojnim mjestima unutar aplikacije, odnosno kada god subjekt mora pristupiti nekoj funkcionalnosti ili nekom resursu. Na slici 1.6 vidimo da se autorizacija odvija i u horizontalnom smjeru i vertikalnom smjeru.



Slika 1.6: Jednostavan prikaz slojeva na kojima se provodi autorizacija

Najprije opisujemo horizontalne slojeve:

- *Klijentov web preglednik* - posao mu je slanje zahtjeva web poslužitelju i prikazivanje dobivenih rezultata. Klijentov web preglednik postavlja sljedeće pitanje: "Ima li korisnik dozvolu za pristup web pregledniku?". Klijent mora imati minimalnu ulogu u procesu autorizacije, jer je nemoguće kontrolirati klijentsku stranu.
- *Front-end web server* - riječ je o poslužitelju koji prati promet na nekoj IP adresi. Pitanje koje postavlja prilikom autorizacije glasi: "Trebam li uopće komunicirati s ovim računalom?". Ovaj sloj omogućuje filtriranje IP adresa traženih zahjeva i predstavlja najbolju vrstu obrane.
- *Back-end aplikacijski server* - riječ je o poslužitelju ili više njih zaduženih za obradu traženih zahtjeva. Obično uspoređuju informacije procesuirane od strane *Front-end web servera* i resursa koji se traži na temelju određenih pravila koja posjeduju.
- *Baza podataka* - riječ je o komponenti koja je zaslužna za čuvanje podataka potrebnih web aplikaciji. Također mogu sudjelovati u procesu autorizacije a pitanje koje postavljaju glasi: "Ima li klijent koji radi neki zahtjev uopće dozvolu za pristup bazi podataka?".

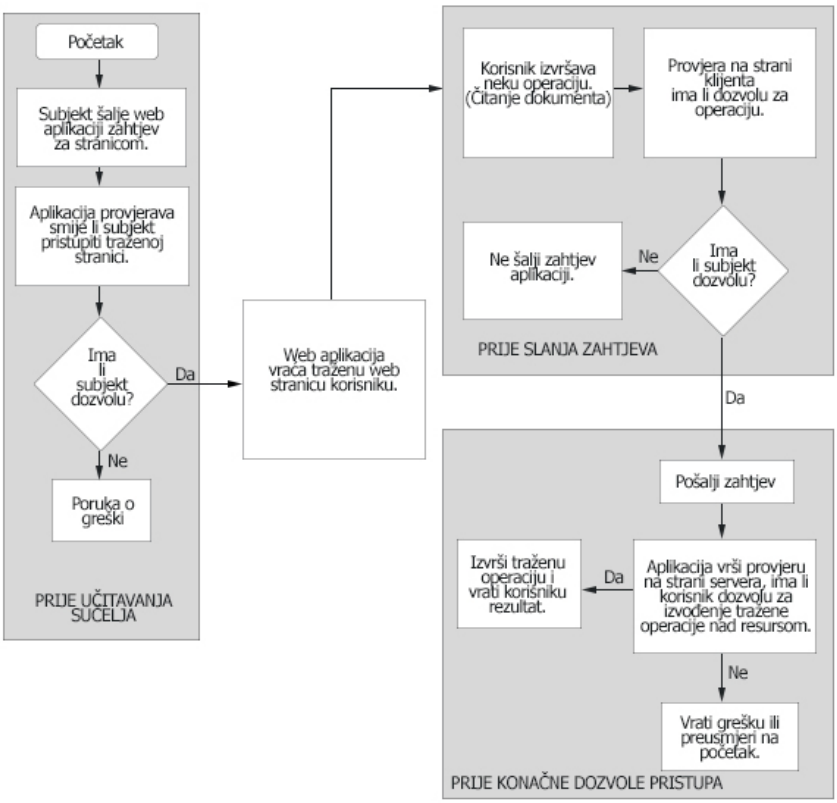
Ovisno o vrsti i složenosti aplikacije, neki slojevi mogu se nalaziti i na istom poslužiteljskom računalu, primjerice *Front-end web server* i *Back-end aplikacijski server*. [15]

Vertikalni smjer ovisi o specifičnom računalu, no slojevi koje se javljaju su obično: korisnički sloj, aplikacijski sloj, međuaplikacijski sloj, sloj operacijskog sustava i sloj hardvera. Što se tiče ovog smjera, važno je imati na umu da postoji i da treba također posvetiti pažnju na sigurnost tog smjera, međutim nije previše bitan za ovu temu.

Razvoj mehanizma autorizacije

Upotreba gotovih kostura (engl. *frameworks*), odnosno autorizacijskih modula omogućuje programerima uštedu vremena potrebnog za testiranje, ali i troškova potrebnih za razvoj autorizacijskog mehanizama. Međutim u nekim situacijama, programeri razvijaju vlastite mehanizme za autorizaciju. Svaki od mehanizama autorizacije bazira se na upotrebi faktora iz sljedećih skupina: što i kada. Prva skupina faktora, odnosno *što*, određuje sudionike autorizacije. Postoje tri vrste sudionika:

- *Subjekti* - entiteti koji traže zahtjev za određenim resursom.
- *Operacije* - funkcionalnosti ili specifične akcije koje subjekt može izvesti.
- *Objekti* - podaci o kojima brine aplikacija.



Slika 1.7: Primjer upotrebe jednostavnog modela autorizacije

Druga skupina faktora, odnosno *kada*, određuje vrijeme kada je potrebno provesti kontrolu dozvola i davanja odobrenja. U ovu skupinu spadaju tri faktora:

- *Prije učitavanja sučelja* (engl. *Before loading the interface*) - interakcija između korisnika i web aplikacije obično se odvija kroz web stranice koje sadrže razne HTML elemente. Ovaj faktor označava da će se autorizacija provesti prije slanja određenog dijela sučelja, odnosno nekih elemenata stranice, korisničkom pregledniku. Primjerice, možemo korisniku vratiti stranicu bez da sadrži link na određenu funkcionalnost. Međutim ovaj korak sam nije dovoljan jer korisnik može pogađati link putem URL-a. Dakle, drugi korak bi bio da aplikacija na zahtjev za tim URL-om najprije provede postupak autorizacije, a zatim ako je korisnik prošao autorizaciju dozvoli traženu akciju.

- *Prije nego li su zahtjevi poslani* (engl. *Before requests are submitted*) - autorizacija se provodi na strani korisnika. Primjerice pomoću *Javascript* koda. Bazira se na upotrebi efekata koji ne zahtjevaju interakciju s poslužiteljem. Dakle, ovaj je faktor važan u smislu performansi no napadač ga može zaobići.

- *Prije davanja konačnog pristupa* (engl. *Before granting final access*) - riječ je o faktoru koji označava najvažnije vrijeme za provedbu autorizacije. To je autorizacija na strani poslužitelja neposredno prije davanja dozvole za pristup resursu. Važno je da vremenski interval između procesa autorizacije, dozvole za korištenje resursa te same akcije bude minimalan. [15]

1.3 Upravljanje sesijom

HTTP protokol je protokol bez stanja, odnosno protokol koji ne može pamtiti stanje između dvije transakcije. Bazira se na modelu zahtjev-odgovor, gdje svaki par predstavlja nezavisnu transakciju. Protokol nema ugrađenih mehanizama za praćenje različitih zahtjeva, odnosno bez sesija i upravljanja sesijama web poslužitelj ne može pripisati određeni niz zahtjeva specifičnom korisniku. Spomenimo još da su autorizacija i sesija usko vezane jer sesija objedinjuje proces autentifikacije i autorizacije za vrijeme interakcije korisnika i web aplikacije.

Sesija predstavlja niz HTTP zahtjeva i odgovora koji su pridruženi određenom korisniku. Obično je ostvarena kao vrsta jedinstvenog trajnog žetona ili identifikatora sesije kojeg web klijent koristi prilikom slanja svakog zahtjeva web aplikaciji. U većini slučajeva, aplikacija razmjenjuje identifikatore sesija s korisnikom pomoću HTTP kolačića. [15]

Na primjer, nakon što se klijent uspješno autentificira, poslužitelj mu odgovara slanjem HTTP zaglavlja koje sadrži:

Set-Cookie: SessionId = 6fktilab3hldc5277r94qh2204

Tada svaki klijentov zahtjev poslužitelju sadrži zaglavlje:

Cookie: SessionId = 6fktilab3hldc5277r94qh2204

Ako pogledamo prethodni slučaj, važno je primjetiti da taj identifikator bude zaista teško predvidljiv i šifriran kako bi otežali posao napadača.

U nekim aplikacijama, obično se sesija uspostavlja i prije same autentifikacije. Na primjer, aplikacija za kupovinu koja prati sadržaj košarice gosta. Tada je vrlo važno da se prilikom autentifikacije korisnika dodjeli novi identifikator sesije. Također, ako gledamo taj primjer, možemo zaključiti da poslužitelj mora bilježiti koje je korisnik proizvode izabrao te o kojoj se količini tih proizvoda radi. Za praćenje takvih informacija služi nam *stanje sesije* (engl. *Session state*). U nastavku navodimo načine na koje možemo pamtiti stanje sesije:

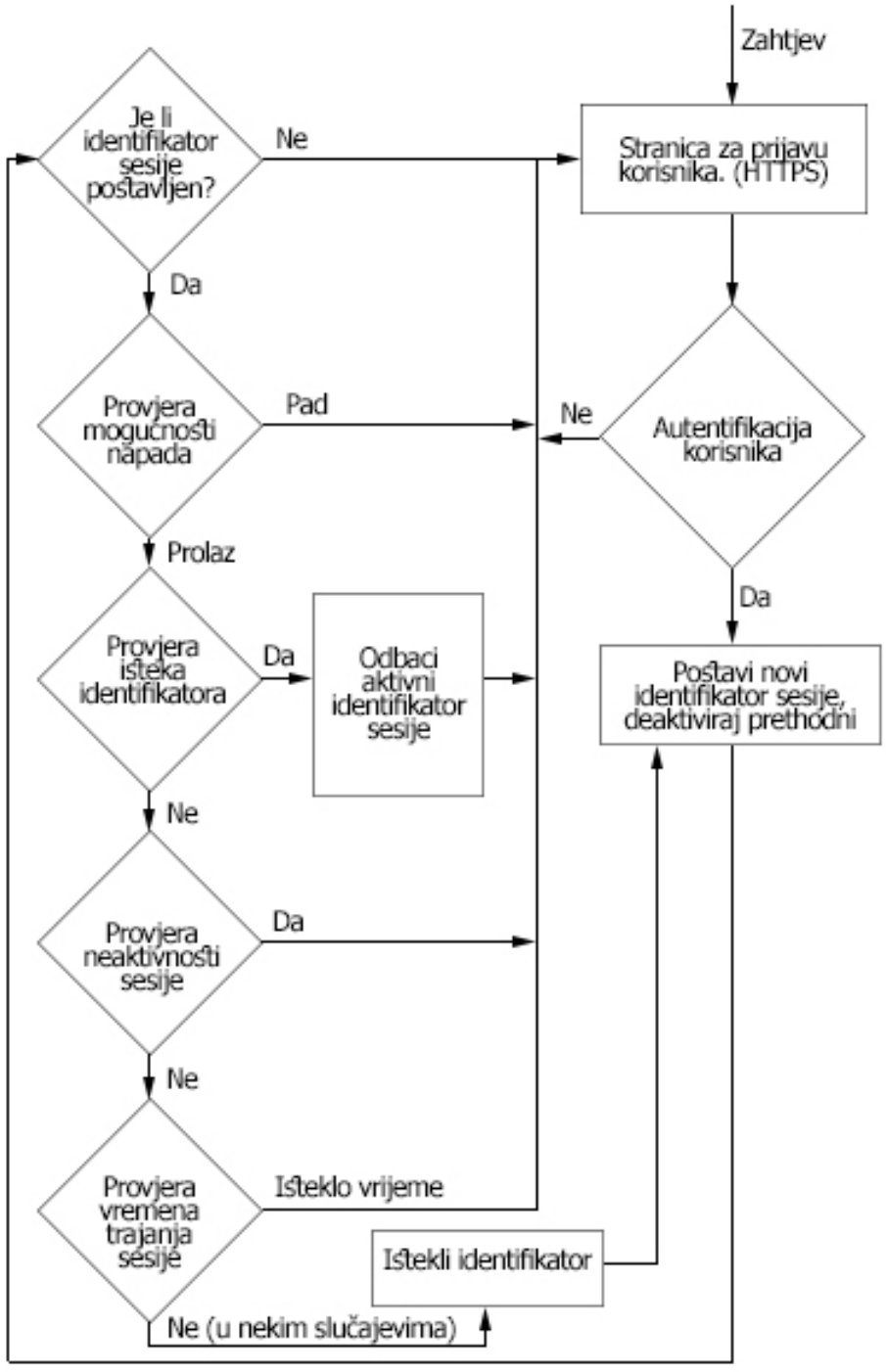
- Pomoću kolačića - informacije se nalaze unutar jednog ili više kolačića na strani klijenta. Prednost korištenja ovog načina jest jednostavnost implementacije. Mana je mali kapacitet spremanja, potreba za šifriranjem i nepotrebno slanje informacija za svaki zahtjev.

- Pomoću forma i URL parametara - informacije se nalaze unutar skrivenih formi ili unutar URL parametara. Uporaba ni jednog od ovih načina nije preporučljiva zbog toga što korisnik vrlo lako može promjeniti vrijednosti informacija.

- Pomoću *HTML5 Local Storage* - informacije se spremaju na strani klijenta, u obliku parova koji sadrže ključ i vrijednost. Ne šalju se za svaki zahtjev, međutim i dalje je potrebno šifriranje informacija.

- Spremanjem na strani poslužitelja - najsigurnija opcija spremanja informacija o sesiji. Obično je također riječ o parovima koji sadrže ime i vrijednost pridruženu nekom identifikatoru sesije.

Upravljanje sesijom (engl. *Session Management*) važan je dio sigurnosti web aplikacije te se obično ugrađuje putem dobro osmišljenog procesa dizajna. Na slici 1.8 prikazan je primjer dijagrama mogućeg procesa upravljanja sesijom.



Slika 1.8: Primjer procesa upravljanja sesijom

1.4 Siguran rad baze podataka

Baze podataka predstavljaju nezaobilazan i važan dio svake moderne web aplikacije. U ovom poglavlju najprije ćemo govoriti o pojmovima kao što su baza podataka i sustav za upravljanje bazom podataka. Zatim ćemo navesti određena pravila koja doprinose sigurnosti baza podataka.

Općenito o bazi podataka

Baze podataka predstavljaju skup međusobno povezanih podataka koji su pohranjeni u vanjskoj memoriji. O njihovom održavanju, distribuiranju i nadziranju brinu *sustavi za upravljanje bazama podataka* (engl. *Data Base Management System - DBMS*). Neki od važnijih sustava za upravljanje bazama podataka su:

- MySQL,
- Microsoft SQL Server,
- Oracle.

Ciljevi upotrebe baze podataka su:

- fizička nezavisnost podataka,
- logička nezavisnost podataka,
- fleksibilan pristup,
- istovremeni pristup,
- čuvanje integriteta,
- mogućnost oporavka,
- zaštita od neovlaštene upotrebe,
- brzina pristupa,
- mogućnost održavanja i kontrole. [11]

Prethodno navedeni DBMS-i zasnovani su na relacijskom modelu podataka, odnosno zasnivaju se na pojmu relacije.

Sigurnosni elementi baze podataka

Dodjela minimalnih prava pristupa

Korisnicima je potrebno ograničiti prava pristupa na samo ona neophodna za normalan rad. Time se smanjuju potencijalne točke napada na web aplikaciju. [15]

Na primjer, ako aplikacija koristi jednog korisnika za pristup bazi podataka (tzv. aplikacijskog korisnika), tada je potrebno reducirati prava na razinu potrebnih za normalan rad aplikacije. Dakle, ako aplikacija nema potrebu za dodavanje ili brisanje određenih redaka unutar neke tablice tada je potrebno ukinuti prava aplikacijskog korisnika za te operacije. Ako aplikacija zahtjeva pristup bazi od strane više različitih korisnika koji se razlikuju po ulogama, primjerice obični korisnik i administrator, potrebno je kreirati više računa za pristup bazi te na temelju uloga ograničiti prava pristupa. Posebnu pažnju potrebno je posvetiti ažuriranju prava nakon uklanjanja nekih funkcionalnosti iz web aplikacije jer u protivnom redukcija prava nema smisla.

Upotreba spremljenih procedura

Upotrebom spremljenih procedura koristimo maksimalan stupanj zaštite od ugnježđenih SQL naredbi. Potrebno je korisnicima baze ograničiti pristup samo na izvršavanje spremljenih procedura bez mogućnosti izvršavanja vlastitih SQL naredbi. [7]

Koristiti enkripciju

Preporuča se upotreba enkripcije prilikom prijenosa podataka, što većina DBMS podržava upotrebom SSL protokola. Osim prijenosa, kriptiranje je moguće i na podacima koji se nalaze u bazi, no to se podrazumijeva na najosjetljivije podatke. [15]

Pravilna upotreba korisničkih podataka

Korisnički podaci, odnosno podaci potrebni za pristup bazi podataka, moraju biti postavljeni i zaštićeni u skladu s pravilima. Više o tome biti će rečeno u dijelu zaštita autentifikacije.

Izrada sigurnosnih kopija

Izrada sigurnosnih kopija baze podataka omogućuje vraćanje podataka uslijed pogreške unutar aplikacije ili same baze podataka. [7]

Nadzor i evidencija

Nadzor i evidencija važan su dio sigurnosti rada baze podataka. Ako je poznata adresa pristupa korisnika bazi podataka, moguće je pratiti određene promjene vezane uz adresu pristupa korisnika ili pak određenog vremena, za koje smo sigurni da nije pristupano od validnog korisnika. [14]

1.5 Siguran rad web preglednika

Web preglednik je program koji omogućuje lociranje, primanje i prikaz informacija koje se nalaze na internetu. Prema podacima sa slike 1.9, korisnici su u ovoj godini najčešće upotrebljavali preglednike Google Chrome i Mozilla Firefox.

2016	Chrome	IE	Firefox	Safari	Opera
July	71.9 %	5.2 %	17.1 %	3.2 %	1.1 %
June	71.7 %	5.6 %	17.0 %	3.3 %	1.1 %
May	71.4 %	5.7 %	16.9 %	3.6 %	1.2 %
April	70.4 %	5.8 %	17.5 %	3.7 %	1.3 %
March	69.9 %	6.1 %	17.8 %	3.6 %	1.3 %
February	69.0 %	6.2 %	18.6 %	3.7 %	1.3 %
January	68.4 %	6.2 %	18.8 %	3.7 %	1.4 %

Slika 1.9: Upotreba pojedinih web preglednika u 2016. godini

Kako su web preglednici nezaobilazan svakodnevni alat koji korisnici koriste, vrlo često postaju metom napada. U ovom poglavlju govoriti ćemo o pravilima kojih se web preglednici moraju pridržavati kako bi što bolje zaštitili svoje korisnike.

Pravilo istog podrijetla

Pravilo istog podrijetla (engl. *same-origin policy*) predstavlja pravilo koje je usuglašeno od strane vodećih proizvođača web preglednika, a osigurava ograničenje funkcionalnosti skripti pokrenutih na web pregledniku korisnika. Točnije, pravilo tvrdi da u trenutku kada

korisnik pregledava neku web stranicu, pokrenuta skripta može čitati i pisati u sadržaj druge web stranice ako oboje imaju isto podrijetlo. [15]

Podrijetlo je kombinacija tri važne stavke:

- protokola aplikacije - HTTP ili HTTPS,
- TCP porta,
- imena domene - ”http://www.stranica.com”.

Uzmimo kao primjer stranicu ”http://www.primjer.com” na kojoj se nalazi određena skripta. U tablici 1.1 su navedene web stranice čije informacije skripta ne može čitati.

URL	Razlog
https://www.primjer.com	Razlika u protokolima, ova stranica koristi HTTPS
http://www.primjer1.com	Razlika u domenama
http://www.primjer.com:8080	Razlika u portovima
http://moj.primjer.com	Razlika u domenama

Tablica 1.1: Primjer odnosa podrijetla stranica

Također, bitno je spomenuti da pravilo istog podrijetla ne vrijedi za poslužiteljsku stranu, odnosno ne ograničava kod na strani poslužitelja. Na primjer, poslužitelj ”www.primjer.com” može slati zahtjeve prema ”www.primjer1.com”. Glavni cilj ovog pravila jest spriječiti web aplikaciju da čita osjetljive i privatne podatke korisnika drugih stranica.

Dogovorom između proizvođača web preglednika omogućeno je i zaobilaženje ovog pravila. Navodimo neke primjere zaobilaženja ovog pravila:

- pomoću HTML <script> elementa,
- upotrebom JSON (*JavaScript Object Notation*),
- upotrebom elemenata <iframe>, <frame>.

Iako je zaobilaženje ovog pravila moguće, valja biti oprezan jer u protivnom možemo aplikaciju izložiti napadima u kojima korisnik može izgubiti privatne podatke.

Pravilo sigurnosti sadržaja

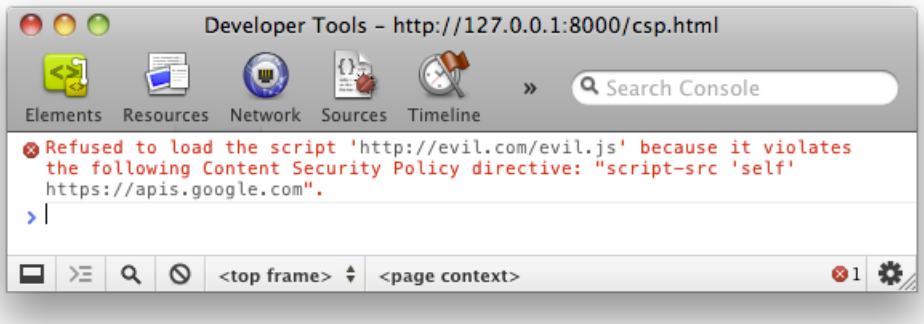
Pravilo sigurnosti sadržaja (engl. *Content-secure-policy*) predstavlja dodatni sloj sigurnosti web preglednika koji definira CSP (*Content-secure-policy*) HTTP zaglavlja koja dozvoljavaju kreiranje bijelih lista izvora (onih za koje znamo da su validni) provjerenog sadržaja

i nalažu web pregledniku da isključivo izvršava i koristi samo te resurse. Ako napadač i pronađe način da umetne skriptu, ona se neće izvršiti jer neće odgovarati *bijeloj listi* (engl. *whitelist*).

Većina vodećih preglednika podržava ovo pravilo te se vrlo lako primjenjuje. Dovoljno je konfigurirati web poslužitelj da vraća CSP HTTP zaglavlje. Pogledajmo sljedeći primjer zaglavlja:

```
Content-Security-Policy: script-src 'self' https://apis.google.com
```

Ovim pravilo definirali smo da su validni izvori skripte: naša aplikacija i "https://apis.google.com". Web preglednik će u slučaju bilo kojeg drugog izvora za skripte izbaciti pogrešku (vidi sliku 1.10).



Slika 1.10: CSP povreda pravila

Valja spomenuti da uz ovo pravilo postoji još mnogo direktiva vezanih uz resurse. Dodatno moguće je kontrolirati i izvršavanje tzv. *inline* skripti.

Ovo pravilo koristi se i u svrhu obrane od *Clickjacking* napada. Dovoljna je upotreba direktive *frame-ancestors* koja web pregledniku označava smije li učitavati stranice unutar *<frame>* i *<iframe>* elemenata.

Na kraju spomenimo još da postoje i direktiva *report-uri* koja šalje povratne informacije vezane uz povredu pravila natrag web poslužitelju. To je vrlo korisna mogućnost pomoću koje lako možemo pratiti vrstu i izvor povrede. Moguće je koristi i zaglavlje *Content-Security-Policy-Report-Only* koje omogućuje da web poslužitelj prima informacije o narušavanju pravila, no web preglednik ne blokira izvršavanje takvih skripti.

Poglavlje 2

Napadi na web aplikaciju

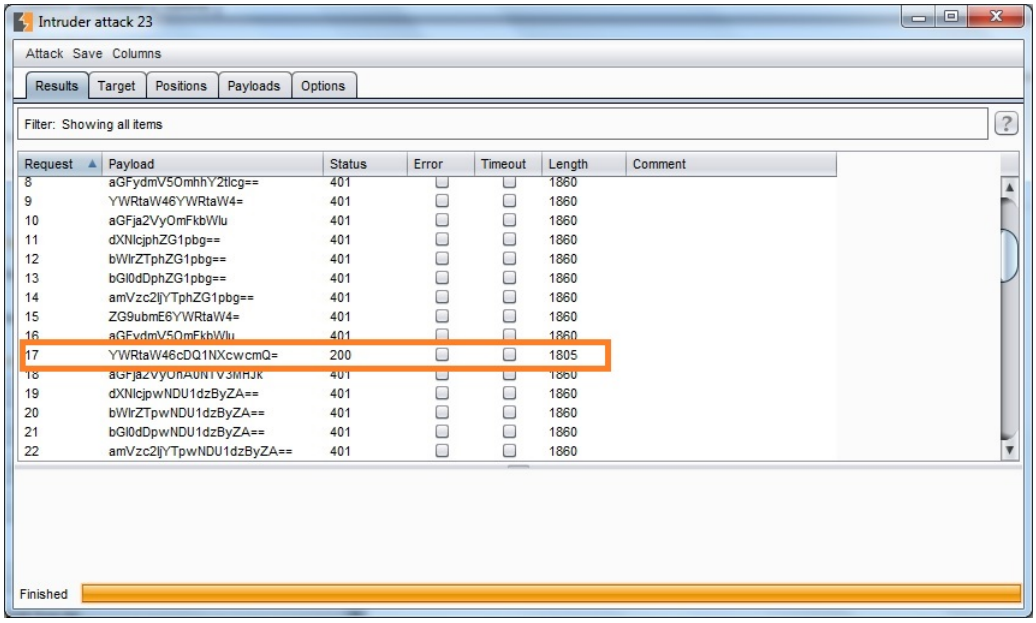
U ovom poglavlju govorit ćemo o raznim vrstama napada na aplikaciju. Napadi su klasificirani s obzirom na sigurnosni princip koji napadaju. Najprije su opisani napadi vezani uz autentifikaciju, zatim slijede napadi na autorizaciju, upravljanje sesijom i baze podataka. Na kraju poglavlja navedeni su i opisani napadi na web preglednike.

2.1 Napadi vezani uz autentifikaciju

Napad grubom silom

Napad grubom silom (engl. *Brute-Force attack*) predstavlja proces pogađanja nepoznate vrijednosti koristeći pritom sve moguće kombinacije te analiziranje dobivenih odgovora. Ova vrsta napada koristi se najčešće u svrhu otkrivanja korisničkog imena, lozinke ili sigurnosnog broja kreditne kartice. Ako web aplikacija dozvoljava neograničeni broj pokušaja prijave korisnika, sve dok nisu unijeti točni podaci, tada je ta aplikacija podložna napadu grubom silom. Web aplikacije koje nemaju dovoljnu kontrolu nad kvalitetom korisničkih lozinki omogućavaju korisnicima da koriste kratke lozinke, lozinke koje označavaju neke nazive ili lozinke koje su iste korisničkom imenu. Posljedica toga jest da će napadi grubom silom biti uspješni. Vrijeme potrebno za uspješno izvršavanja ovakve vrste napada ovisi o duljini traženog podatka te o znakovima od kojih se taj podatak sastoji.

Spomenimo još i da postoji *reverzivan napad grubom silom*. Ovakva vrsta napada koristi činjenicu da više korisnika može imati istu lozinku te se pogađanje odnosi na korisničko ime. Primjena reverznog napada grubom silom ima smisla u slučaju kada aplikacija ima velik broj korisnika. Najčešći alati koji se koriste za napad grubom silom su: Burp Intruder, THC Hydra, Brutus. Ovi alati posjeduju i *rječnike* (engl. *dictionary*) u koje je moguće dodati određene pojmove, riječi i najčešće lozinke. [15]



Slika 2.1: Uspješan napad grubom silom pomoću alata Burp Intruder.

U nastavku navodimo neke korake kako izvesti napad grubom silom, ali i kako možemo provjeriti je li naša aplikacija podložna tom napadu:

- Za napad grubom silom koristimo za to predviđene alate, kao što smo prije naveli.
- Upotrebljavamo više različitih rječnika, na više jezika.
- Pratimo moguće pogreške koje se javljaju u procesu autentifikacije.
- Ako aplikacija dozvoljava više pokušaja prijave s iste IP adrese ili pak je dozvoljeno više desetaka pokušaja prijave u sekundi, tada ta aplikacija vrlo vjerojatno ne može detektirati ovakav napad. [5]

Na slici 2.1 prikazan je uspješan napad grubom silom na HTTP *Basic access* autentifikacijski mehanizam primjenom alata *Burp Intruder*. Kako znamo da taj autentifikacijski mehanizam koristi *base64* šifriranje, tada iz podatka ”YWRtaW46cDQ1NXcwcmQ=”, dešifriranjem pomoću online alata vrlo lako dobivamo korisničke podatke:

”admin:p455w0rd”

Napadi na propuste u dizajnu autentifikacije

Napadi vezani uz propuste u dizajnu autentifikacije posljedica su lošeg dizajna mehanizma za autentifikaciju. Od velike je važnosti da softverski inžinjeri na vrijeme utvrde sve zahtjeve koji su važni za dizajn mehanizma autentifikacije, pošto on predstavlja prvu liniju obrane od neželjenog pristupa zaštićenim dijelovima aplikacije. U nastavku opisujemo propuste u dizajnu autentifikacijskog mehanizma koji su česti uzrok napada na autentifikacijski mehanizam.

Preopširne poruke o neuspjehu

Aplikacije prilikom autentifikacije korisnika, koji žele pristupiti zaštićenim djelovima, zahtijevaju korisničke podatke. Ako neki od unesenih podataka nije točan dolazi do neuspjeha autentifikacije. Stoga, kako bi olakšale posao korisniku, neke aplikacije vraćaju poruke o tome koji od podataka nije točan. Za jednostavne aplikacije koje zahtijevaju unos korisničkog imena i lozinke, ovakav propust nije prevelik problem jer napadač može saznati samo koja od korisničkih imena su valjana, a koja nisu.

Slika 2.2: Opis slike

U složenijim aplikacijama, koje koriste autentifikaciju kroz više slojeva, ovakva vrsta propusta omogućuje napadaču da pogađa određene informacije i da se kreće kroz slojeve. To u konačnici može rezultirati time da napadač pristupi zaštićenim djelovima.

Nesiguran prijenos korisničkih podataka

Riječ je o propustu u dizajnu mehanizma za autentifikaciju gdje dolazi do otkrivanja korisničkih podataka.[14] Ako se korisnički podaci šalju putem nešifrirane HTTP veze, tada je moguće pomoću alata za prisluškivanje presresti korisničke podatke u nešifriranom obliku. Ako pak se prijava korisnika odvija putem HTTPS veze, korisnički podaci se mogu otkriti na sljedeće načine:

- Korisnički podaci su poslati kao niz parametara unutra URL-a, umjesto u tijelu POST metode. Napadač tada može otkriti korisničke podatke ako se domogne povijesti korisničkog preglednika ili zapisa poslužitelja.
- Ako su korisnički podaci spremljeni u kolačiću na korisničkom računalu.

Promjena lozinke

Promjena lozinke vrlo je važna funkcionalnost za svaki dobro dizajniran autentifikacijski mehanizam. Jedan od vrlo važnih ciljeva promjene lozinke jest da korisnik prisili napadača da ponovno pokrene napad grubom silom. Time smanjuje vjerojatnost uspješnog napada. Neke aplikacije zbog lošeg dizajna autentifikacijskog mehanizma omogućuju pristup ovoj funkcionalnosti te time dovode napadača u dobar položaj. Unutar same funkcionalnosti mogući su sljedeći propusti:

- preopširne poruke o neuspjehu omogućuju provjeru točnosti korisničkog imena,
- omogućeno je neograničeno pogađanje postojeće lozinke. [14]

Način na koji možemo provjeriti je li aplikacija ranjiva opisujemo sljedećim koracima:

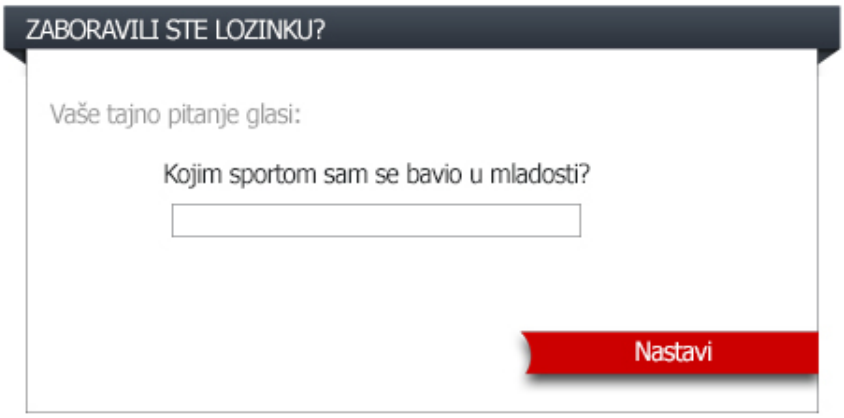
- promijenimo lozinku,
- ponovno pokušamo promijeniti lozinku u vrlo kratkom vremenskom intervalu. Ako uspijemo, tada je ova aplikacija ranjiva a u najgorem slučaju moguća je i primjena napada grubom silom.
- pokušavamo razne kombinacije unosa podataka unutar formi za promjenu lozinke te promatramo ponašanje aplikacije. [5]

Zaboravljena lozinka

Korisnici dnevno koriste na desetke različitih web aplikacija. Posljedica toga je da za većinu aplikacija moraju pamtiti različite lozinke. Mehanizam za oporavak od zaboravljene lozinke omogućuje korisniku obnovu lozinke u slučaju da je istu zaboravio. Međutim ako je ovaj mehanizam lošeg dizajna, tada postaje najslabija karika te potencijalna meta napadača. Neki primjeri propusta:

- Mehanizam za oporavak od zaboravljene lozinke dozvoljava neograničen broj pokušaja pogađanja odgovora na tajno pitanje. Time napadaču omogućuje primjenu *brute-force* alata.

- Mehanizam umjesto tajnog pitanja omogućuje korisniku da za svoju lozinku sam izabere *savjet* (engl. *hint*) kako da se sjeti lozinke. Obično korisnici nesvjesni opasnosti postavljaju savjete koji omogućuju lako pogađanje lozinke.
- Mehanizam dozvoljava korisniku da sam izabere tajno pitanje prilikom registracije. Kao i u slučaju izbora savjeta, korisnici često izaberu tajna pitanja s malim skupom potencijalnih odgovora.
- Mehanizam koristi jedinstvenu i vremenski ograničenu URL poveznicu za oporavak od zaboravljene lozinke, no dopušta korisniku da izabere mail adresu na koju se ta poveznica šalje.
- Mehanizam nakon odgovora na tajno pitanje, korisniku daje uvid u trenutnu lozinku ili ga autentificira. Na taj način napadač može trajno koristiti korisnikov račun a da korisnik nije ni svjestan te činjenice. [14]



Slika 2.3: Zaboravljena lozinka

Zapamti me

U nekim aplikacijama često susrećemo opciju „zapamti me“. Ona omogućuje korisnicima da ne moraju unositi svoje korisničke podatke prilikom autentifikacije s određenog računala. Propusti u dizajnu ove funkcionalnosti često izlažu korisnika napadaču. Neki od propusta u dizajnu funkcionalnosti „zapamti me“ su:

- vrijednosti unutar kolačića koji je aplikacija poslala se mogu predvidjeti,

- lozinka unutar kolačića nije šifrirana ili je šifrirana pomoću lošeg algoritma,
- vijek trajanja kolačića je predug.

Napadi na propuste u implementaciji autentifikacije

Propusti unutar implementacije mehanizma za autentifikaciju mogu narušiti sigurnost dobro dizajniranih i oblikovanih autentifikacijskih mehanizama. U pravilu se teže otkrivaju nego li propusti unutar dizajna autentifikacijskog mehanizma te dovode do curenja informacija, mogućnosti zaobilaženja prijave korisnika pa i čitave sigurnosti aplikacije. Postoje različite vrste propusta koji se mogu dogoditi prilikom implementacije autentifikacijskog mehanizma. U nastavku navodimo neke primjere propusta:

- Autentifikacijski mehanizam nema dobru implementaciju rukovanja iznimkama, stoga ponašanje aplikacije može biti nepredvidivo.
- Neke aplikacije zbog povećanja sigurnosti uvode višeslojnu autentifikaciju. No, zbog propusta je moguće smanjenje sigurnosti. Primjerice, ako mehanizam nema kontrolu redoslijeda izvršavanja koraka, tada napadač lako može preskočiti neke od potrebnih koraka autentifikacije. Nadalje, problem se javlja i ako aplikacija vjeruje podacima koji su provjereni u prethodnim koracima. Napadač tada može promjeniti podatke između koraka.
- Implementacija mehanizma za autentifikaciju je ostvarena na način da korisnik dodatno odgovara na neko tajno pitanje. Ako aplikacija za svakog od korisnika ne prati postavljena pitanja, već generira nova, tada napadač vrlo lagano može iterirati ponovnim pokušajima kroz sva pitanja te odgovoriti na ono pitanje na koje zna odgovor.

2.2 Napadi vezani uz autorizaciju

Prisilno pregledavanje

Prisilno pregledavanje (engl. *Forced browsing*) je napad na proces autorizacije kada napadač manualno unosi URL za resurs kojem ne može pristupiti putem aplikacijskog sučelja. [15]

Na primjer, napadač upisuje URL ”www.webstranica.com/admin/”. Ako aplikacija ne provodi autorizaciju za pristup svakom zaštićenom resursu, tada je moguća primjena ovakve vrste napada.

Falsifikacija parametara

Falsifikacija parametara (engl. *Parameter Tempering*) je vrsta napada kada napadač falsificira parametre zahtjeva prije nego li zahtjevi stignu web aplikaciji.[15] Cilj ovog napada su obično podaci unutar URL parametara, kolačića ili formi aplikacije.

Na primjer, napadač otkrije skrivenu formu koja označava neko radno mjesto unutar tvrtke. Uplata bonusa moguća je samo za određena radna mjesta. Tada pomoću *Firebug* dodatka za preglednik *Firefox* promjeni parametre u željene. Ako je autorizacijski mehanizam loš, ovaj napad je vrlo vjerojatan.

Manipulacija HTTP zaglavlja

HTTP zahtjev sadrži nekoliko linija zaglavlja koja sadrže metapodatke o zahtjevu. Na slici 2.4 vidimo primjer jednog HTTP zaglavlja.

Slika 2.4: Primjer HTTP zaglavlja

Riječ je o napadu kada napadač mijenja podatke unutar HTTP zaglavlja. Na primjer, pomoću podatka *Accept-Language* može izvesti napad na bazu podataka, ako aplikacija koristi direktni unos kao upit na bazu. Aplikacija je podložna ovoj vrsti napada ako vjeruje podacima unutar zaglavlja HTTP zahtjeva, koje je kreirano na strani korisnika.[15]

2.3 Napadi na upravljanje sesijom

Cilj napada na upravljanje sesijom obično predstavlja impersonalizacija korisnika ili manipulacija informacijama vezanim uz sesiju kako bi se pristupilo određenim zaštićenim resursima. U ovom poglavlju govorimo o raznim vrstama napada na sesiju i nekim propustima u dizajnu mehanizma za upravljanje sesijom koji to omogućuju.

Trovanje kolačića

Trovanje kolačića (engl. *Cookie poisoning*) predstavlja tehniku falsificiranja identifikatora sesije ili drugih informacija na strani klijenta. Napad se izvodi na stanje sesije spremljeno na strani klijenta kao što je primjerice kolačić ili neki od već opisanih mehanizama na strani klijenta koje smo opisali u prvom poglavlja. Svrha napada jest pokušaj zaobilaženja procesa autorizacije.

Krađa identifikatora sesije

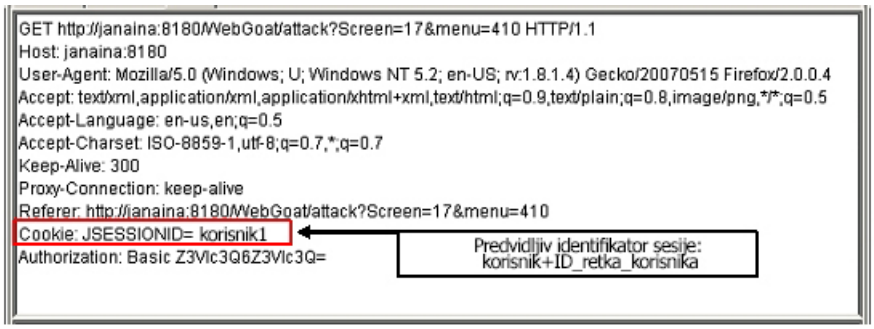
Krađa identifikatora sesije (engl. *Stealing session ID*) najraširenija je vrsta napada na upravljanje sesijom.[15] Napadač pokušava ukrasti identifikator sesije drugog korisnika u svrhu daljnje impersonalizacije. [15]

Na primjer, jedan način krađe identifikatora sesije moguće je izvesti putem *cross-site scripting (XSS)* napada. Odnosno, napada koji prevari korisnikov preglednik pokrenuvši neku štetnu skriptu na strani korisnika. Taj napad bit će opisan u napadima na web preglednike. Drugi način krađe može biti putem prisluškivanja mreže koja nema zaštićen prijenos podataka između klijenta i poslužitelja.

Predvidljivost identifikatora sesije

Predvidljivost identifikatora sesije jest propust u generiranju identifikatora koji omogućuje napadaču predviđanje vrijednosti te zaobilaženje procesa autentifikacije i autorizacije.

Na primjer, vrlo loše generiranje identifikatora sesije bilo bi jednostavno korištenje identifikatora koji se javlja unutar redova tablice korisnika iz baze podataka (slika 2.5).



Slika 2.5: Predvidljivo generiranje identifikatora sesije

Predvidljivost identifikatora sesije možemo podijeliti u sljedeće podkategorije propusta:

- Značajan identifikator sesije (engl. *Meaningful session tokens*) - riječ je o identifikatorima koji su nastali transformacijom nekih korisničkih podataka kao što su: korisničko ime, ime i prezime korisnika, e-mail adresa i uloga korisnika. Ti podaci su obično razdvojeni određenim delimiterima te u većini slučajeva šifrirani nekim od sljedećih supstitucija: *XOR*, *Base64* ili heksadecimalna supstitucija. Napadač najprije analizira a zatim raspakirava vrijednosti.

- Identifikator vezan uz vrijeme nastanka - identifikatori sesije generaju se na temelju vremena generiranja. Aplikacije koje imaju veliku količinu posjeta, napadaju se skriptiranim napadima i traženjem uzorka. Na primjer, identifikatori sesije:

  3124553-1172764800468
  3124554-1172764800609
  3124555-1172764801109
  3124556-1172764801406
  3124557-1172764801703

  predstavljaju uzorak sljedećeg tipa: inkrementirajući indeks + ”-” + trenutno vrijeme u milisekundama. Ako promotirimo niz ”1172764801703” tada se lako vidi da je vrijeme prema nultoj vremenskoj zoni:

  01.03.2007 16:00:01

- Loš generator brojeva - riječ je o propustu zbog kojeg se generiraju predvidljivi brojevi a samim time i identifikator sesije. [14]

Fiksacija sesije

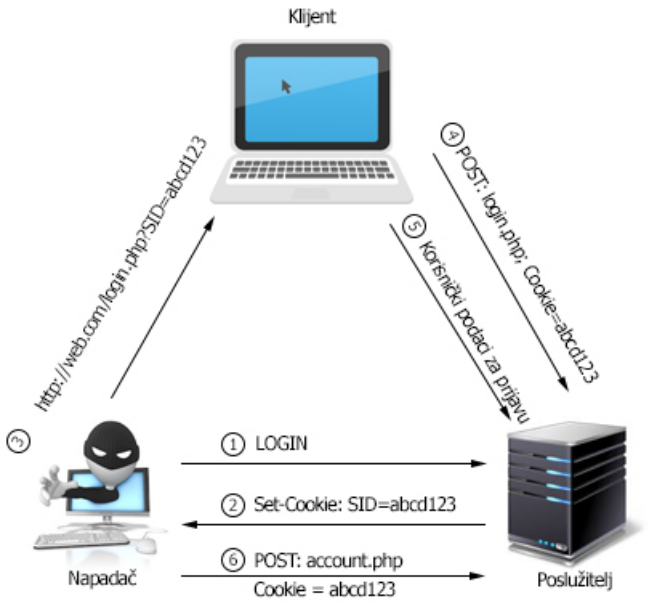
Fiksacija sesije (engl. *Session Fixation*) je napad koji se javlja kada napadač može dodati identifikator sesije unutar korisnikovog web preglednika.[15] Dakle, napad se izvršava prije prijave korisnika. Ovakva vrsta napada omogućena je jer se prilikom autentifikacije korisnika, korisniku ne pridružuje novi identifikator sesije. Na napadaču je da upotrebi validan identifikator i podmetne ga korisniku. Tehnike napada ovise o načinu podvale validnog identifikatora sesije korisniku:

- putem URL parametara,

- putem skrivene HTML forme,

- putem kolačića.

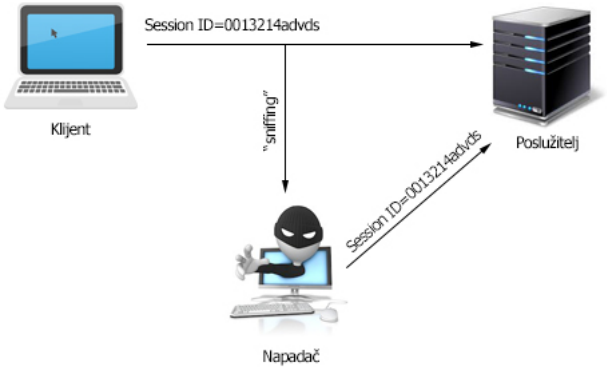
Na slici 2.6 prikazan je jednostavan primjer napada fiksacijom sesije koristeći URL parametre.



Slika 2.6: Primjer jednostavne fiksacije sesije

Otmica sesije

Otmica sesije (engl. *Session Hijacking*) predstavlja trenutak kada napadač iskoristi ukraden ili fiksiran identifikator sesije.[15] Obično rezultira impersonalizacijom korisnika. Još neke podvrste ovog napada su: *Session sniffing* i *čovjek u sredini* (engl. *Man-in-middle*). Na slici 2.7 vidimo jednostavni primjer *Session sniffing* napada.



Slika 2.7: Primjer jednostavnog *Session sniffing* napada.

Propust u implementaciji odjave

Riječ je o propustu u implementaciji funkcionalnosti odjave korisnika kada programeri omoguće samo brisanje kolačića na strani korisnika. Međutim, problem se javlja jer i dalje postoje validni identifikatori sesije na strani poslužitelja koje napadač može iskoristiti.

2.4 Napadi na baze podataka

Napad ugnježđenim SQL naredbama

Napad ugnježđenim SQL naredbama (engl. *SQL injection attack*) je napad koji omogućuje napadaču izvršavanje modificiranog SQL upita putem web aplikacije.[15] Ako napadač uspješno izvrši ovu vrstu napada, može otkriti osjetljive podatke iz baze podataka, modificirati podatke unutar baze, zaobići proces autentifikacije i autorizacije, te u nekim slučajevima izvršavati operacije dozvoljene samo administratorima. Uspješan napad ugnježđenim SQL naredbama može ugroziti tri bitna svojstva svake web aplikacije:

- *Tajnost* - aplikaciji su ukradeni osjetljivi podaci, koje je više nemoguće vratiti.
- *Integritet* - aplikaciji je narušen integritet ako napadač uspije izmjeniti ili dodati određene podatke unutar baze podataka.
- *Dostupnost* - aplikacija prestaje biti dostupna ako primjerice napadač uspije obrisati tablicu korisnika. [15]

U sljedećem primjeru pokazat ćemo napad na sva tri svojstva aplikacije. Zamislimo da aplikacija dozvoljava upite korisnika putem varijable *$upit*:

```
$upit = "Ivan";
$sql = "SELECT ime, prezime FROM User WHERE
       ime = '" . $upit . "'";
$result = $conn->multi_query($sql);
```

Napad na svojstvo dostupnosti mogao bi izgledati ovako:

```
$upit = "Ivan'; DROP TABLE User; --'";
$sql = "SELECT ime, prezime FROM User WHERE
       ime = '" . $upit . "'";
$result = $conn->multi_query($sql);
```

Napad na svojstvo integriteta može izgledati ovako:

```
$upit = "';UPDATE User SET prezime = 'Ivanovic'
        WHERE ID = '1'; --'";
$sql = "SELECT ime, prezime FROM User
       WHERE ime = '" . $upit . "'";
$result = $conn->multi_query($sql);
```

Napad na svojstvo tajnosti može izgledati ovako:

```
$upit = "';SELECT * FROM KreditneKartice; --'";
$sql = "SELECT ime, prezime FROM User
       WHERE ime = '" . $upit . "'";
$result = $conn->multi_query($sql);
```

Još jedan primjer ove vrste napada susreće se u aplikacijama koje provjeravaju korisničke podatke na temelju kriterija postoji li unutar baze barem jedan korisnik s tim podacima. Tada, ako je moguće izvesti napad ugnježdenim SQL naredbama, napadač može izvršiti sljedeći upit:

```
$sql = "SELECT * FROM User WHERE username = 'bla'
       AND password = 'bla' OR '1'='1'";
$result = $conn->multi_query($sql);
```

Zbog zadnjeg dijela upita *OR '1'='1'*, ovaj upit će uvijek biti istinit, te napadač može zaobići proces autentifikacije.

Slijep napad ugnježđenim SQL naredbama

Slijep napad ugnježđenim SQL naredbama (engl. *Blind SQL injection attack*) je podvrsta napada ugnježđenim SQL naredbama gdje napadač izvršava napad na temelju upućenih da/ne pitanja prema bazi podataka. Napadač obično upotrebljava ovu vrstu napada u slučajevima kada aplikacija ne ispisuje detaljne poruke o pogreškama.[15]

Uzmimo primjer neke web aplikacije koja omogućuje svojim članovima pregled uplata novčanih donacija. Aplikacija u slučaju iznimke ili kada SQL upit nema rezultata, preusmjerava korisnika na početnu stranicu. U slučaju da upit daje rezultate, aplikacija preusmjeri korisnika na stranicu s informacijama o iznosima novčanih donacija tog člana. Sljedeći dio koda opisuje taj slučaj:

```
$sql = "SELECT * FROM Donacije WHERE
       id_korisnika = '" . $id  .  "'";
try {
 $result = $conn->multi_query($sql);
        if($result->num_rows > 0){
          header('Location:
          http://www.primjer.com/donacije.php');
        }
        else {
          header('Location:
          http://www.primjer.com/index.php');
        }
} catch (Exception $e) {
          header('Location:
          http://www.primjer.com/index.php');
}
```

Odmah na prvi pogled, može se primjetiti da je prethodni dio koda podložan napadu ugnježđenim SQL naredbama jer upit ovisi o varijabli *$id* koju kontrolira korisnik. Napadač najprije može promijeniti upit na sljedeći način:

```
$id = "'";
$sql = "SELECT * FROM Donacije WHERE
       id_korisnika = '" . $id  .  "'";
```

U ovom slučaju, aplikacija će uhvatiti iznimku zbog nepravilno strukturiranog upita i vratiti napadača na početnu stranicu. Nakon toga, napadač može promjeniti SQL upit u:

```
$id = "' OR '1'='1";
$sql = "SELECT * FROM Donacije WHERE
```

```
        id_korisnika = '" . $id  .  "'";
```

U ovom slučaju, upit će uvijek biti istinit zbog dijela *OR '1'='1'* i napadač će biti preusmjeren na stranicu donacija. Naravno ovaj postupak daje ključnu informaciju napadaču da je aplikacija podložna napadu ugnježđenim SQL naredbama. Za kraj može još provjeriti sljedeći slučaj:

```
$id = "' AND '1'='2";
$sql = "SELECT * FROM Donacije WHERE
        id_korisnika = '" . $id  .  "'";
```

To će opet u konačnici rezultirati vraćanjem na početnu stranicu, pošto je dio izraza *AND '1'='2'* uvijek neistinit. Sada kada napadač zna ponašanje aplikacije, može primjeniti niz da/ne upita prema bazi podataka o informacijama vezanim uz tablice koje se nalaze unutar baze. Na primjer, može pitati ima li prva tablica u bazi prvo slovo 'C':

```
$id = "'OR MID((SELECT TABLE_NAME FROM
         information_schema.tables LIMIT 1),1,1)= 'C";
$sql = "SELECT * FROM Donacije WHERE
        id_korisnika = '" . $id  .  "'";
```

Na temelju ponašanja, odnosno preusmjeravanja, napadač pomoću ovih da/ne upita, s vremenom može otkriti važne informacije vezane uz bazu podataka.

2.5 Napadi na web preglednike

HTML injekcija

HTML injekcija (engl. *HTML injection*) je vrsta napada na web preglednike koji je moguće izvesti kada korisnik ima kontrolu nad unosom podataka i kada je u mogućnosti injektirati HTML kod unutar postojeće web aplikacije.

Na primjer, napadač može ubaciti formu za prijavu korisnika i *POST* metodom poslati unesene korisničke podatke svojoj web aplikaciji. Time bi korisnik bio potpuno ugrožen jer bi vrlo teško otkrio krađu svojih podataka.

Cross-Site Scripting

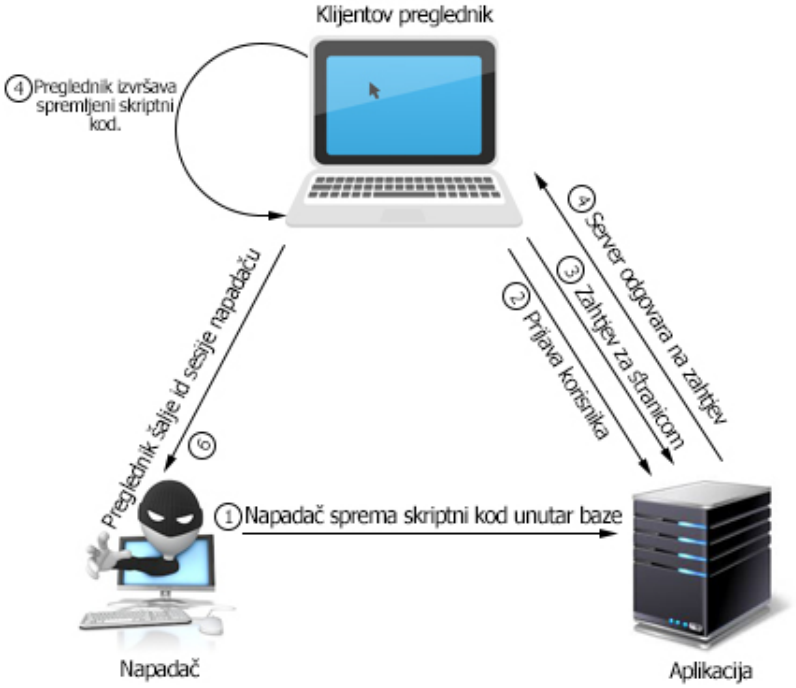
Cross-Site Scripting ili kraće XSS predstavlja propust koji omogućuje napadaču umetanje svog skriptnog koda unutar ranjive web aplikacije.[14] Ako napadač navede korisnika da pistupi toj stranici, web preglednik će preuzeti i pokrenuti zlćudnu skriptu.

Glavni uzrok ovog propusta jest to što aplikacija uzima korisnički unos bez validacije i šifriranja. Na slici 2.8 prikazan je primjer takve web tražilice.

Slika 2.8: Primjer web tražilice s XSS propustom

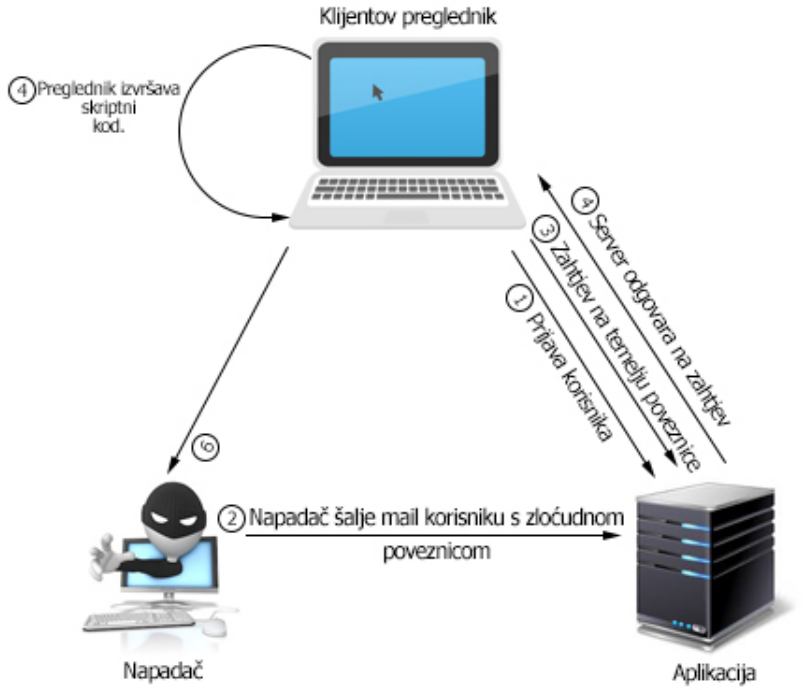
Postoje tri tipa napada na XSS propust:

- *Spremljen XSS napad* (engl. *Stored XSS attack*) - predstavlja najopasniju vrstu napada na XSS propust. Uključuje slučaj u kojem napadač može trajno pohraniti zloćudni skriptni kod unutar aplikacije (npr. unutar baze podataka) koji aplikacija kasnije prikazuje korisnicima bez ikakve provjere. Na primjer, uzmimo slučaj da aplikacija na stranicu ispisuje komentare svih korisnika. Ako napadač uspije spremiti skriptni kod unutar baze te ga aplikacija ne provjeri, korisnički preglednik će prilikom pokretanja stranice izvršiti skriptni kod i ugroziti korisnika. Na slici 2.9 je prikazan primjer mogućeg spremljenog XSS napada.



Slika 2.9: Primjer spremljenog XSS napada

- *Reflektiran XSS napad* (engl. *Reflected XSS attack*) - riječ je o napadu u kojem napadač umeće skriptu u web stranicu koja predstavlja odgovor poslužitelja na korisnikov zahtjev. Napadač obično navede korisnika da koristi zlćudnu poveznicu koja će potom umetnuti skriptni kod unutar dinamičke web stranice i omogućiti pristup korisničkim podacima. Na slici 2.10 je prikazan mogući primjer ove vrste napada.



Slika 2.10: Primjer reflektiranog XSS napada

- *DOM baziran XSS napad* (engl. *DOM-based XSS attack*) - tip napada koji je omogućen kada skripte na strani korisnika unose podatke od strane korisnika u DOM (engl. *Document Object Model*). Prilikom ove vrste napada umetnuti zloćudni skriptni kod se može izvršavati na korisničkom računalu s istim posljedicama kao i reflektiran XSS napad.

Cross-Site Request Forgery

Cross-Site Request Forgery ili kraće CSRF jest vrsta napada koja prisiljava korisnika koji je autentificiran od strane određene web aplikacije da izvrši neželjene radnje vezane uz tu aplikaciju. Ovaj napad se oslanja na činjenicu da je dovoljno prevariti korisnika da pošalje zahtjev ranjivoj aplikaciji bez potrebe odgovora.[15]

Na primjer, uzmimo da korisnik koristi ranjivu aplikaciju na adresi ”www.banka.com”. Neka je jedna od funkcionalnosti te aplikacije prebacivanje novčanih sredstava s računa korisnika na neki drugi račun. Također pretpostavimo da aplikacija ne koristi dodatne provjere autentifikacije, već se oslanja isključivo na provjeru identifikatora sesije. Postupak se odvija sljedećim koracima:

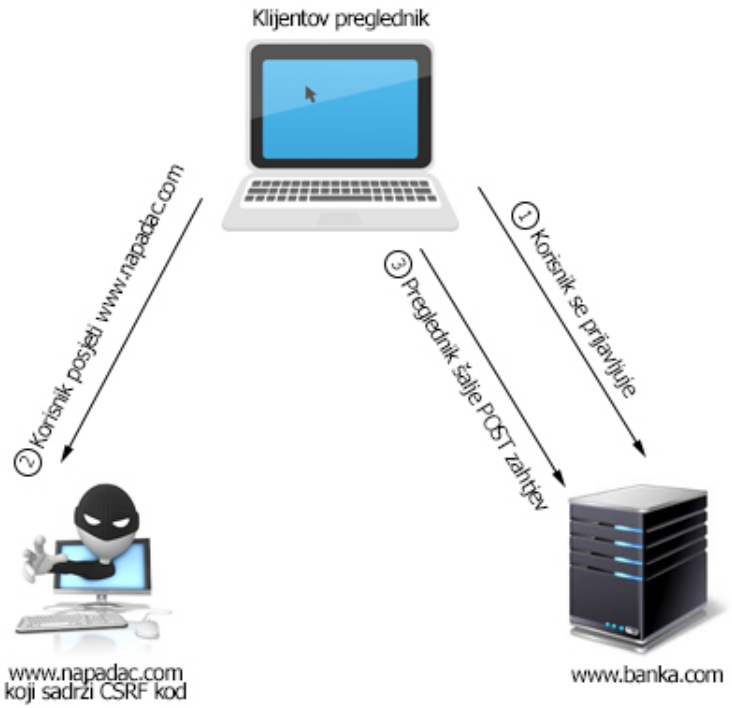
1. Korisnik se prijavljuje u aplikaciju ”www.banka.com” te mu se pridružuje identifikator sesije.

2. Korisnik pristupa stranici ”www.napadac.com” na način da mu napadač najprije pošalje poveznicu unutar e-mail poruke s porukom da je sretni dobitnik nagradne igre a zatim korisnik klikne na tu poveznicu. Na toj stranici nalazi se forma koja se automatski šalje putem POST metode aplikaciji ”www.banka.com”. Neka je izgled stranice sljedeći:

```
<html>
 <body onload="document.primjer.submit()">
   <form action="https://www.banka.com/transfer"
         name="primjer" method="post">
      <input type="hidden" name="br" value="10000" />
      <input type="hidden" name="acc" value="1231" />
   </form>
</body>
</html>
```

3. Korisnikov preglednik automatski šalje POST zahtjev aplikaciji ”www.banka.com” s identifikatorom sesije.

Pošto smo pretpostavili da nema dodatnih koraka autentifikacije, kao ponovna autentifikacija korisnika, korisnik ostaje bez 10000 novčanih sredstava. Ilustraciju ovog primjera možete vidjeti na slici 2.11.



Slika 2.11: Jednostavan primjer CSRF napada

Clickjacking

Clickjacking ili *UI Redress attack* je napad u kojem napadač koristi više transparentnih ili poluprozirnih slojeva s ciljem prevare korisnika da klikne na gumb ili poveznicu različite stranice od one koju je on namjeravao koristiti. Dakle, govorimo o krađi korisničkog klika.

Na primjer, neka napadač kreira stranicu na kojoj se nalazi gumb s natpisom "klikni ovdje kako bi preuzeo besplatnu nagradu". Međutim, na površini stranice, odnosno najgornjem sloju, napadač je kreirao *iframe* element koji sadrži mail aplikaciju. Također, poravnao je taj element tako da je operacija za brisanje svih poruka točno nad prije spomenutim gumbom. Ako korisnika prilikom posjeta ovoj stranici mail aplikacija unutar prozirnog *iframe* elementa autentificira i korisnik klikne na gumb, izvršit će se operacija brisanja svih poruka.

Poglavlje 3

Zaštita web aplikacije

U ovom poglavlju govorit ćemo o tehnikama i pravilima zaštite web aplikacije. Pojedine tehnike i pravila podijeljena su s obzirom na sigurnosni princip koji štite. U početku su opisane tehnike i pravila zaštite autentifikacije. Zatim slijede tehnike i pravila zaštite autorizacije, upravljanja sesijom te baza podataka. Na kraju poglavlja opisana su pravila zaštite web preglednika, odnosno pravila zaštite od XSS i CSRF propusta.

3.1 Zaštita autentifikacije

Zaštita lozinke

Većina današnjih autentifikacijskih mehanizama bazira se na upotrebi lozinke kao potvrde postupka autentifikacije. Stoga, nije ni čudo da je napad na lozinke jedan od najčešćih napada na web aplikaciju. Da bi se aplikacija, a i time sami korisnici, zaštitili od raznih vrsta napada vrlo je bitno da aplikacija brine o složenosti korisničkih lozinki. Neka od pravila kojih se moramo pridržavati kako bismo što bolje osigurali korisničke lozinke su:

- *Minimalna duljina lozinke* – vrlo važno pravilo koje određuje minimalan broj znakova od kojih se svaka lozinka mora sastojati. Iako u praksi mnoge web aplikacije zahtijevaju da se korisnička lozinka sastoji od minimalno 8 znakova, preporučena minimalna duljina trebala bi biti barem 12 ili 14 znakova. Potrebno je također napomenuti da je duljina lozinke bitnija od mogućeg skupa znakova koji se mogu nalaziti unutar lozinke.
- *Minimalni zahtjevi na složenost lozinke* – dobra lozinka sastoji se od barem jednog znaka iz sljedeće 4 kategorije znakova:

Kategorije	Znakovi
Velika slova	A, B, C, ..., X, Y, Z
Mala slova	a, b, c, ..., x, y, z
Brojevi	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Simboli	()'~!@#$%&/^*+-={}[]:;"<>,.? | \

Tablica 3.1: Tablica simbola

- *Rotacija lozinke* – pravilo koje preporučuje frekvenciju zamjene lozinke, time se smanjuje mogućnost uspješnog napada grubom silom, pošto ovakve vrste napada iziskuju dosta vremena. Preporuča se izmjena lozinke svakih 90 dana.
- *Jedinstvenost lozinke* – pravilo koje zahtjeva od korisnika da prilikom promjene lozinke ne upotrebljava prethodno korištene lozinke ni da koristi lozinke koje su identične korisničkom imenu.
- *Dobro generirani korisnički podaci* – ako aplikacija ne dozvoljava korisniku da sam postavi svoje korisničke podatke već ih sama generira, tada takve vrijednosti ne smiju biti predvidljive. [15]

Zaštita od napada grubom silom

Napad grubom silom predstavlja pokušaj otkrivanja korisničkih podataka isprobavanjem svih mogućih kombinacija slova, brojeva i simbola dok se ne pronađe točna kombinacija. Web aplikacija koja ima neki mehanizam za autentifikaciju nameće se kao vjerojatna meta napada grubom silom. Ako se napadača pravovremeno ne detektira i ne spriječi u izvođenju napada grubom silom, tada će vrlo vjerojatno otkriti korisničke podatke. U nastavku navodimo neka pravila za obranu od napada grubom silom:

- *Zaključavanje korisničkog računa* - nakon određenog broja propalih pokušaja prijave korisnika, aplikacija zaključa njegov korisnički račun na određeno vrijeme. Ovisno o stupnju sigurnosti koji zahtjeva aplikacija, važno je odrediti koliko pokušaja prijave je moguće prije zaključavanja, unutar kojeg vremenskog intervala te na koje vremensko razdoblje se korisnički račun zaključava. Ovo pravilo je vrlo učinkovito u borbi protiv napada grubom silom, no ako napadač posjeduje listu korisničkih imena, vrlo lako može zloupotrijebiti ovo pravilo u cilju zaključavanja korisničkih računa. Druga mana ovog pravila jest to što ne može spriječiti reverzni napad grubom silom gdje napadač isprobava jednu ili nekoliko lozinki s velikim brojem korisničkih imena. [15]

Stupanj sigurnosti	Broj dozvoljenih pokušaja	Interval mjerenja	Period zaključavanja
Niski	10	60 minuta	30 minuta
Visoki	5	30 minuta	neodređeno

Tablica 3.2: Preporučene vrijednosti zaključavanja korisničkog računa.

- *Zaštita korisničkog imena* - važno je da aplikacija štiti tajnost korisničkog imena i ne dopušta enumeraciju kroz njih jer tada napadač najprije mora pogoditi koja su korisnička imena validna. Ovo pravilo je važno jer u velikoj mjeri usporava običan i reverzni napad grubom silom. [14]
- *Upotreba CAPTCHA* - CAPTCHA (engl. *Completely Automated Public Turing test to tell Computers and Humans Apart)* jest proces koji na temelju određenog izazova traži odgovor. Upotrebljava se u borbi protiv automatiziranih napada grubom silom gdje zahtjeva odgovor na koji može odgovoriti samo čovjek. Iako je zbog nesigurne implementacije i računalnog prepoznavanja znakova, moguće zaobilaženje procesa CAPTCHA, to je vrlo zahtjevno i većina napača radije odustaje od napada grubom silom. [15]

Slika 3.1: CAPTCHA

- *Zaštita informacija o zaključavanju* - informacije o pravilima zaključavanja korisničkog računa moraju biti tajne jer u protivnom napadač može prilagoditi automatizirane napade s obzirom na broj dozvoljenih pogrešnih prijava i vremenski interval u kojem to aplikacija prati. Primjerice, dovoljno je korisniku ispisati ”Pokušajte kasnije”. [5]

Siguran prijenos korisničkih podataka

Komunikacija između klijenta i poslužitelja mora se odvijati putem šifriranog kanala kao što je SSL. Ako aplikacija za nezaštićene dijelove koristi HTTP protokol, potrebno je osigurati da se sama forma za prijavu učitava pomoću HTTPS protokola. Zahtjeva se izbjegavanje slanja korisničkih podataka putem URL parametara ili pomoću kolačića. Korisničke podatke potrebno je slati pomoću POST metode. [15]

Pravilna validacija korisničkih podataka

Pravilna validacija korisničkih podataka podrazumijeva potpunu provjeru svih podataka koji služe za autentifikaciju korisnika. Zahtjeva se provjera lozinke bez ikakvih modifikacija i hvatanje svih iznimki koje se mogu pojaviti u procesu autentifikacije. Ako proces autentifikacije zahtjeva odgovor na tajno pitanje, važno je osigurati da se za određenog korisnika pitanje ponavlja tako dugo dok odgovor nije točan. To postižemo na način da je pitanje spremljeno na strani poslužitelja.

Nadalje, u slučaju da je proces autentifikacije višeslojan potrebno je zadovoljiti sljedeća pravila:

- Svi podaci o tijeku procesa autentifikacije moraju se nalaziti na poslužitelju te ne smiju biti vraćeni korisniku.
- U svakom sloju procesa autentifikacije provjerava se jesu li prethodni slojevi zadovoljeni i korektni, u protivnom treba odbaciti takav pokušaj autentifikacije.
- Informacije o pogreškama na pojedinim slojevima potrebno je sakriti od korisnika. Najbolje je proces autentifikacije provesti do kraja te tada ispisati neku generičku poruku o pogreški.
- Provjereni podaci u prethodnim slojevima ne mogu mjenjati. [14]

Zaštita funkcionalnosti promjene lozinke

Već smo prije spomenuli važnost mogućnosti promjene lozinke unutar aplikacije. Promjena lozinke mora biti implementirana u svakoj aplikaciji kako bi omogućila korisnicima da zamjene svoje lozinke bez obzira na razlog. U nastavku navodimo pravila koja štite mehanizam promjene lozinke od zlouporabe:

- Promjeni lozinke mogu pristupiti samo prethodno autentificirani korisnici.
- Prilikom promjene ne smije biti moguće slanje korisničkog imena jer korisnici smiju mjenjati samo svoje lozinke.
- Nova lozinka mora biti unijeta dva puta, drugi put zbog potvrde nove lozinke kako bi se izbjegle slučajne pogreške. Također aplikacija prvo treba provjeriti podudaraju li se ta dva unosa.

Zaštita funkcionalnosti ”zapamti me”

Aplikacije koje zahtjevaju visok stupanj sigurnosti u pravilu ne smiju imati mogućnost ”zapamti me”. No, moguće je razmotriti mogućnost pamćenja korisničkog imena. Ako

aplikacija nema visok stupanj sigurnosti tada može imati opciju ”zapamti me”, ali mora upozoriti korisnika na moguće opasnosti.

Zaštita funkcionalnosti zaboravljene lozinke

Pravila zaštite funkcionalnosti zaboravljene lozinke su:

- Mehanizam oporavka od zaboravljene lozinke ne smije nuditi mogućnost savjeta (engl. *hint*) za lozinke. Razlog tome jest činjenica da korisnici često odabiru preočite savjete.
- Tajna pitanja ne smiju se izmjenjivati ako korisnik ne pruža točan odgovor. Najbolji način odabira tajnog pitanja jest iz skupa prethodno pripremljenih pitanja, gdje je skup mogućih odgovora dovoljno velik.
- Najbolji način zaštite jest mehanizam koji korisniku nakon što odgovori točno na tajno pitanje, pošalje jedinstveni, jednokratni i vremenski ograničen URL za oporavak putem e-mail poruke na mail adresu korisnika. Klikom na tu poveznicu, korisniku se omogućuje postavljanje nove lozinke. [14]

Nadzor i obavijesti procesa autentifikacije

Aplikacija mora bilježiti sve događaje vezane uz proces autentifikacije. To su: pogreške prilikom prijave korisnika, pogreške unutar funkcionalnosti vezanih uz lozinku i blokiranje korisničkih računa. Bilo kakva vrsta pogreške mora dati upozorenje te pokrenuti mehanizam obrane. Korisnici moraju biti obaviješteni o svim događajima kao što su primjerice promjena lozinke ili pak broj neuspjelih pokušaja prijave. Na taj način mogu pratiti jesu li žrtve napada.

3.2 Zaštita autorizacije

Crne i bijele liste IP adresa

Crne i bijele liste IP adresa služe nam kao prva crta obrane web aplikacije. Poslužitelj može provjeriti IP adrese sa koji dolaze zahtjevi i po potrebi ih filtrirati. *Crne liste IP adresa* (engl. *IP address blacklists*) sadrže IP adrese na koje poslužitelj u slučaju primanja zahtjeva odgovara pogreškom ili ih jednostavno ignorira. *Bijele liste IP adresa* (engl. *IP address whitelists*) sadrže raspon IP adresa ili IP adrese koje smiju slati zahtjev poslužitelju. Njima je moguće smanjiti raspon mogućih napadača. [15]

URL autorizacija

Ovisno o poslužitelju ili kostoru (engl. *framework*) web aplikacije moguće je ograničiti pristup određenom URL-u.

Pravila dobrog dizajna i implementacije autorizacije

Ključ sigurnosti procesa autorizacije leži u dobrom dizajnu i implementaciji mehanizma za autorizaciju. Pravila koja osiguravaju tu sigurnost su:

- Ako se dogodi pogreška prilikom autorizacije, potreban je dobar mehanizam za upravljanje iznimkama kako bi se izbjeglo nepredvidivo ponašanje aplikacije. Najbolje je resetirati proces i isti pokrenuti ponovo.
- Aplikacija na poslužitelju ne smije biti pokrenuta s administratorskim ovlastima. Potrebno je limitirati ovlaštenja na minimum a da i dalje aplikacija može normalno funkcionirati.
- Korisnički račun mora u potpunosti biti odvojen od administratora. Ako administrator mora obavljati poslove korisnika, tada se zahtjeva da upotrebljava korisnički račun. Korisnik nikada ne smije imati privilegije administratora.
- Definiranje strogih pravila.
- Upotreba jedinstvenih korisničkih računa. Pod ovim pravilom smatra se da samo jedan korisnik može upotrebljavati jedan korisnički račun. Ako se dozvoli djeljenje korisničkog računa, smisao autorizacije se gubi te ga je nemoguće u potpunosti sigurno provesti.
- Proces autorizacije mora se provesti za svaki zahtjev. Ovo pravilo spriječava napad prisilnog pregledavanja.
- Autorizacijski mehanizam mora biti centraliziran. Tako u slučaju promjena ili propusta lako možemo promjeniti određeni dio.
- Ako je to moguće, izbjeći dizajniranje i implementaciju vlastitog mehanizma autorizacije. Razlog tome jest činjenica da postoje već dobro testirani i pouzdani autorizacijski mehanizmi.
- Izbjegavanje nesigurnih autorizacijskih informacija od strane klijenta. Klijentu je nemoguće vjerovati jer nemamo kontrolu nad njihovim računalom. Informacije koje su dobro šifrirane i nalaze se na strani klijenta koji ne može znati ključ, mogu biti prihvaćene kao točne.

- Siguran poslužitelj je poslužitelj koji nikome ne vjeruje.
- Proces autorizacije siguran je jedino na strani poslužitelja. Na strani klijenta, autorizacija se javlja s ciljem poboljšanja performansi. [15]

3.3 Zaštita upravljanja sesijom

Apsolutno vremensko ograničenje sesije

Ako promotrimo problem beskonačnog trajanja sesije, primjetili bismo da to omogućuje napadaču beskonačni pristup aplikaciji na temelju ukradenog identifikatora sesije. Stoga je važno unutar aplikacije odrediti maksimalno vrijeme trajanja sesije i uništavanje svih sesija čiji je vremenski rok prekoračen. Obično se uništava identifikator sesije, dok se stanje sesije koje se nalazi na strani poslužitelja može sačuvati. Preporučeno vremensko ograničenje ovisi o razini sigurnosti potrebnoj za određenu web aplikaciju.

Za aplikacije visoke razine sigurnosti preporuča se upotreba ograničenja od jednog do dva sata, dok se općenito preporuča upotreba ograničenja od četiri sata.[15] Osim ograničenja koje možemo postaviti na kolačiću identifikatora sesije potrebno je pravilo provesti na strani poslužitelja, pošto klijent može manipulirati podacima na svojem računalu.

Vremensko ograničenje neaktivne sesije

Vremensko ograničenje neaktivne sesije rješava problem prestanka interakcije korisnika s web aplikacijom. Ako se korisnik ne odjavi iz aplikacije, vrlo je teško odrediti je li korisnik završio svoj rad. Važno je da aplikacija prati aktivnost određenog identifikatora sesije, primjerice razlikom između vremena ponovnog pojavljivanja.

Aplikacije visoke razine sigurnosti ograničavaju vrijeme neaktivnosti na 10 minuta, dok se općenito preporuča upotreba ograničenja od 20 minuta.[15]

Ograničenje konkurentnosti sesija

Neke web aplikacije omogućuju više aktivnih sesija vezanih za određenog korisnika. Na primjer, jedna sesija putem računala i jedna putem mobilnog uređaja u isto vrijeme. Ovisno o aplikaciji, u nekim slučajevima uvodi se ograničenje na broj aktivnih sesija čime se može spriječiti napadača od uporabe ukradenih korisničkih podataka. Aplikacije koje dozvoljavaju više sesija po korisniku, moraju obavijestiti korisnika o prijavi. [15]

Upotreba dobrih identifikatora sesije

Identifikator sesije ključan je u identifikaciji korisnika između različitih zahtjeva. Stoga je potrebna određena kontrola kvalitete generiranja tih identifikatora. Cilj je onemogućiti napadača da predvidi vrijednosti identifikatora na temelju većeg broja uzoraka prikupljenih od strane aplikacije. Preporuča se upotreba identifikatora koji ne sadrže informacije o korisniku, koji nisu strukturirani te koji predstavljaju slučajan niz znakova. Od programera se zahtjeva da prouče matematička svojstva pojedinih algoritama koji se koriste za generiranje slučajnih vrijednosti i upotrijebe samo one koji se smatraju dobrima. Kao dodatni korak može se upotrijebiti konkatenacija slučajne vrijednosti, nekih informacija vezanih uz zahtjev te niza znakova koje zna samo poslužitelj. Takav niz na kraju je moguće i šifrirati. [14]

Uništavanje nevažećih identifikatora sesije

Identifikatori sesije postaju nevažećima u slučaju da im je istekao rok upotrebe, zbog neaktivnosti ili zbog neovlaštene ponovne upotrebe. U nastavku navodimo korake pravilnog uništavanja identifikatora sesije:

1. Uništavanje identifikatora na strani poslužitelja.
2. Ako je poslužitelj u mogućnosti poslati odgovor klijentu, šalje mu odgovor koji sadrži *Set-Cookie* direktivu s vremenskim rokom trajanja u prošlosti.
3. Brisanje kolačića prilikom zatvaranja preglednika.

Implementacija funkcionalnosti odjave

Važno je da programeri prilikom dizajna aplikacije uzmu u obzir implementaciju funkcionalnosti odjave jer na taj način aplikacija može biti sigurna da je korisnik završio svoj rad te odmah nakon toga odbaciti identifikator sesije. Na taj način smanjuje se vremenski interval u kojem napadač može iskoristiti ukradeni identifikator sesije.

Generiranje novog identifikatora sesije prilikom autentifikacije

Ovo pravilo navodi da je potrebno prilikom svake autentifikacije korisnika generirati novi identifikator sesije. Drugim riječima, identifikator sesije ne smije se nikada ponovno upotrijebiti. Pomoću ovog pravila izbjegavamo napad fiksacije sesije zbog toga što će nakon autentifikacije korisnik dobiti novi identifikator, koji napadač neće znati.

Sigurna razmjena identifikatora sesije

Identifikatori sesije moraju se slati isključivo putem HTTPS protokola. Ako aplikacija koristi HTTP protokol za određene dijelove tada bilo kakvo slanje osjetljivih podataka mora biti preusmjereno putem HTTPS protokola. Ako se koriste HTTP kolačići za prijenos identifikatora sesije, potrebno je koristiti zastavicu *secure* koja osigurava da ih preglednik ne šalje putem HTTP protokola. Važno je također izbjegavati slanje identifikatora putem URL parametara jer time izlažemo identifikatore na brojnim mjestima, što napadaču širi područje mogućeg napada.

3.4 Zaštita baza podataka

Zaštita od napada ugnježđenim SQL naredbama

Glavni uzrok napada ugnježđenim SQL naredbama jest kreiranje dinamičkih SQL upita na temelju unesenih korisničkih podataka.[14] Kako bi se to onemogućilo potrebno je ne koristiti dinamičke SQL upite ili spriječiti unos podataka koji mogu sadržavati SQL naredbe. Glavni koraci zaštite su sljedeći:

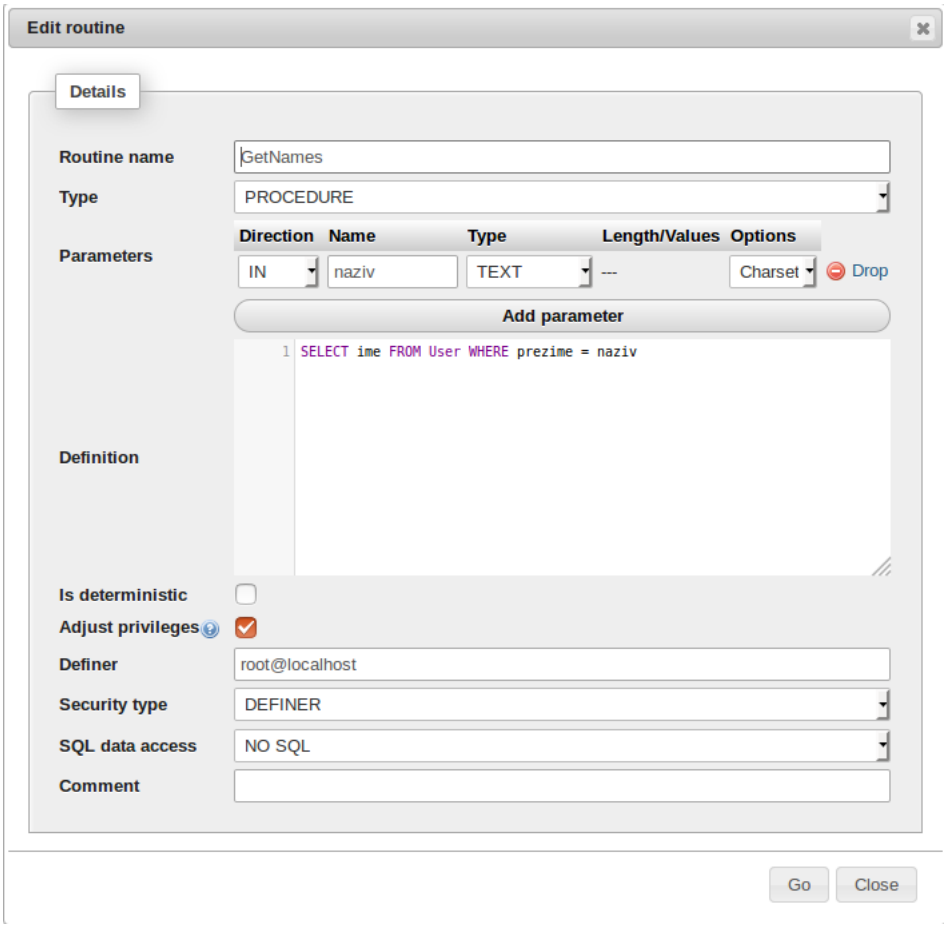
- Aplikacija u slučaju iznimke mora vraćati samo općenite poruke o pogreškama. Potrebno je spriječiti bilo kakvo informiranje korisnika o podacima vezanim uz bazu podataka.
- U slučaju validacije jednostavnijih unosa, primjerice brojeva kartice, preporuča se upotreba regularnih izraza. Posebnu pažnju treba obratiti da regularni izrazi budu pravilno zadani jer u protivnom napadač može izvršiti *ReDos* napad, odnosno napad u kojem je napadač unio takav podatak da se validacija izvršava u beskonačnoj petlji.
- Validaciju je potrebno provoditi na temelju bijelih lista (engl. *whitelists*).
- Podatke koje korisnik unosi potrebno je pretvoriti u odgovarajući tip.
- Upotreba pripremljenih naredbi (engl. *Prepared statement*) najbolji je oblik zaštite od svih vrsta napada ugnježđenim SQL naredbama. U ovom slučaju, konstrukcija SQL upita se provodi u dva koraka, specificira se struktura i rezervira mjesto za korisnički unos, a zatim se u drugom koraku specificira sadržaj koji dolazi na to mjesto. Sljedeći dio koda prikazuje primjer upotrebe pripremljenih naredbi:

```
//priprema
if (!($upit = $conn->prepare("SELECT * FROM User
                              WHERE ime = (?)"))) {
```

```
     echo "Greška u pripremi";
}
//vezanje parametra
$ime = "Marko";
if (!$upit->bind_param("s", $ime)) {
    echo "Greška u vezanju parametra";
}
if (!$upit->execute()) {
     echo "Greška u izvršavanju";
}
if (!($rezultat = $upit->get_result())) {
    echo "Greška u rezultatu";
}
```

- Alternativno upotrebi pripremljenih naredbi preporuča se upotreba spremljenih procedura (engl. *Stored procedures*). Tada je moguće korisnički račun koji pristupa bazi podataka ograničiti samo na izvršavanje takvih naredbi. Na slici 3.2 prikazan je primjer izrade procedure unutar *phpMyAdmin* alata. Primjer korištenja spremljene procedure vidimo u sljedećem kodu:

```
$vari = "Markovic";
$conn->query("SET @un = '" . $vari . "'");
$result = $conn->query("CALL GetNames(@un)");
```



Slika 3.2: Primjer kreiranja spremljene procedure unutar *phpMyAdmin* alata

3.5 Zaštita web preglednika

Zaštita od XSS propusta

XSS propusti vrlo često se susreću unutar već postojećih web aplikacija pa je ujedno i napad na takve propuste jedna od najčešćih vrsta napada na web aplikacije. Testiranje ove vrste propusta može biti vrlo teško i zahtjevno te se obično provodi pomoću automatiziranih alata.[5] Ako se korisnici i programeri pridržavaju sljedećih pravila, mogu spriječiti i najupornije XSS napadače:

- Korisnici moraju izbjegavati posjećivanje nesigurnih poveznica koje obično dobivaju putem sumnjivih mail poruka ili kojima pristupaju putem sumnjivih web stranica.

- Najbolja vrsta obrane od XSS napada jest kodiranje izlaza. Preporuča se upotreba već gotovih biblioteka kao primjerice OWASP ESAPI zbog velikog broja mogućih mjesta na kojima se ti izlazi upotrebljavaju (npr. HTML text, URL, XML, itd.).
- Preporuča se sanacija nesigurnog unosa podataka na način da se ukloni svaka mogućnost pojave skriptnog koda. Opet moguće je korištenje već gotovih biblioteka kao što je OWASP AntiSamy.
- Ako aplikacija zahtijeva mogućnost uređivanja i ukrašavanja korisničkih unosa, preporuča se upotreba tzv. *lightweight markup* jezika kao što je Wikitext.
- Preporuča se upotreba pravila sigurnosti sadržaja.
- Ako nije potreban pristup kolačiću od strane skripti na strani korisnika, potrebno je uključiti zastavicu *HttpOnly*.[15]

Zaštita od CSRF propusta

Prvi korak zaštite od CSRF propusta je zaštita od XSS napada. Razlog tome jest činjenica da ako napadač može podmetnuti skriptni kod koji preglednik izvrši, tada vrlo lako može kreirati skriptni kod koji će kreirat zahtjeve. Također ako je aplikacija podložna XSS napadima, moguće je zaobilaženje nekih pravila zaštite od CSRF napada. Nakon zaštite od XSS napada potrebno je slijediti sljedeća pravila:

- HTTP GET metoda smije se upotrebljavati isključivo za sigurne operacije.
- Ne preporuča se oslanjanje na *HTTP referer* dio zaglavlja.
- Ne preporuča se korištenje osjetljivih podataka unutar URL-a.
- Za osjetljive operacije, kao što je slanje novčanih sredstava, preporuča se reautentifikacija korisnika.
- Uz zaštitu od XSS napada, najbolja vrsta obrane jest implementacija *dijeljene tajne* (engl. *Shared Secret*). Dijeljena tajna funkcionira na način da kada se korisnik prvi put prijavi u aplikaciju, generira se kriptografski jak slučajan broj na strani poslužitelja te pridružuje tom korisniku unutar stanja sesije. Za svaki odgovor na zahtjev korisnika, poslužitelj uključuje taj broj unutar skrivene forme. Kada prima zahtjev, poslužitelj provjerava skrivenu formu i u slučaju nepoklapanja ili nedostatka forme zna da se radi o ovoj vrsti napada. Mana ovog pravila jest dodatno spremanje broja za svakog korisnika te nemogućnost rješavanja problema čovjeka u sredini. Prednost je da već postoje gotove biblioteke kao što je *OWASP CSRFGuard*.

- Analogno obrani pomoću dijeljene tajne može se upotrijebiti obrana zasnovana na *dvostruko predanom kolačiću* (engl. *Double-Submitted Cookie*). Ovakva vrsta obrane zasniva se na tajnosti identifikatora sesije. Prilikom odgovora, poslužitelj šalje korisniku identifikator unutar kolačića te ga također zapisuje unutar skrivene forme. Kada korisnik šalje zahtjev poslužitelju on uključuje kolačić i skrivenu formu. Na kraju, poslužitelj provjerava jesu li vrijednosti primljenih identifikatora identične. Ako aplikacija zahtjeva veću sigurnost, potrebno je osigurati jednosmjerno haširanje identifikatora prilikom slanja unutar skrivene forme. [15]

Poglavlje 4

Metode sigurnog razvoja web aplikacija

Integracija sigurnosti unutar razvoja web aplikacija ključan je korak koji se mora izvršiti u samom početku razvoja aplikacije. Time osiguravamo uštedu vremena i cijene razvoja te izbjegavamo loš pristup kao što je traženje propusta te njihovo ispravljanje. U ovom poglavlju, opisat ćemo dvije metode koje se koriste za siguran razvoj aplikacija. Najprije opisujemo *Security Development Lifecycle* tvrtke Microsoft, a potom *Comprehensive Lightweight Application Security Process* organizacije OWASP (*Open Web Application Security Project*).

4.1 Microsoft Security Development Lifecycle

Microsoft Security Development Lifecycle ili kraće SDL je proces razvoja softvera koji pomaže programerima u razvoju sigurnijeg softvera i usklađivanju sigurnosnih zahtjeva s ciljem jeftinijeg razvoja. Zasnovan je na modelu vodopada, odnosno odvija se kroz niz vremenski odvojenih faza.

Faze SDL procesa su:

1. Trening - svi članovi razvojnog tima moraju jednom godišnje prisustvovati barem jednoj univerzalnoj vježbi vezanoj uz sigurnosni razvoj sofvera.

2. Zahtjevi - potrebno je: definirati i integrirati zahtjeve sigurnosti, definirati najmanju prihvatljivu razinu sigurnosti te provesti sigurnosnu procjenu rizika.

3. Oblikovanje - potrebno je: utvrditi zahtjeve na dizajn, analizirati moguću površinu napada te korisiti modeliranje mogućih prijetnji.

4. Implementacija - potrebno je: objaviti listu odobrenih alata za korištenje, zabraniti nesigurne funkcije te analizirati izvorni kod.

5. Verifikacija - potrebno je: provesti dinamičku analizu, provesti testiranje *Fuzz* metodom te provesti ponovnu analizu moguće površine napada.

6. Izdavanje - potrebno je: izraditi plan odgovora na incidente, provesti finalnu provjeru sigurnosti te potvrditi i arhivirati izdanje.

7. Odgovor - provedba plana odgovora na incidente.

Sve prethodno navedene aktivnosti unutar pojedinih faza uvedene su također i u *SDL-Agile* metodi razvoja softvera. Ta metoda se zasniva na agilnom pristupu razvoja softvera te aktivnosti raspoređuje po iteracijama. Tablica 4.1 prikazuje aktivnosti i njihovu primjenu unutar pojedinih iteracija.

Aktivnost	**Vrijeme provedbe aktivnosti**
Definiranje zahtjeva sigurnosti	Početna iteracija
Definiranje najmanje prihvatljive razine sigurnosti	Moguća provedba kroz više iteracija
Sigurnosna procjena rizika	Početna iteracija
Definiranje zahtjeva na dizajn	Početna iteracija
Analiza mogućeg područja napada	Početna iteracija
Modeliranje mogućih prijetnji	Svaka iteracija
Objava liste odobrenih alata za razvoj	Svaka iteracija
Zabrana nesigurnih funkcija	Svaka iteracija
Analiza izvornog koda	Svaka iteracija
Dinamička analiza	Moguća provedba kroz više iteracija
Fuzz testiranje	Moguća provedba kroz više iteracija
Ponovna analiza mogućih područja napada	Moguća provedba kroz više iteracija
Izrada plana odgovora na incidente	Početna iteracija
Finalna provjera sigurnosti	Svaka iteracija
Potvrda i arhiviranje izdanja	Svaka iteracija

Tablica 4.1: Raspored aktivnosti unutar *SDL-Agile* metode.

U nastavku opisujemo aktivnosti:

- Definiranje zahtjeva sigurnosti - rano definiranje i integriranje zahtjeva sigurnosti omogućuje lakšu identifikaciju ključnih ciljeva te minimalizaciju poremećaja unutar planova razvoja.
- Definiranje najmanje prihvatljive razine sigurnosti - pomaže razvojnom timu da u samom početku razvoja razumije rizike vezane uz sigurnost te da identificira i popravi propuste tijekom razvoja aplikacije.
- Sigurnosna procjena rizika - omogućuje razvojnom timu identifikaciju dijelova projekta koji će zahtjevati modeliranje prijetnji i reviziju sigurnosti prije izdavanja aplikacije.
- Definiranje zahtjeva na dizajn - uključuje točne i potpune specifikacije dizajna i reviziju specifikacija. Pomaže u minimalizaciji rizika poremećaja rasporeda i troškova.
- Analiza mogućeg područja napada - potrebno je temeljito analizirati napadaču dostupno područje napada te onemogućiti ili umanjiti pristup uslugama sustava pomoću slojevite obrane i principa najmanjih privilegija.
- Modeliranje mogućih prijetnji - uključuje strukturirani pristup koji omogućuje timu efikasnije i jeftinije identificiranje propusta te određivanje rizika i sanacije takvih propusta.
- Objava liste odobrenih alata za razvoj - pomaže u automatizaciji i uspostavi provedbe sigurnosnih aktivnosti. Listu je potrebno redovito ažurirati.
- Zabrana nesigurnih funkcija - pomaže u smanjenju potencijalnih propusta. Preporuča upotrebu sigurnijih alternativa.
- Analiza izvornog koda - omogućuje provedbu politike sigurnog kodiranja i propusta unutar implementacije.
- Dinamička analiza - provodi se uz pomoć alata koji nadziru ponašanje prilikom izvođenja koda te služi za prikupljanje informacija koje nisu dostupne u analizi izvornog koda.
- Fuzz testiranje - predstavlja izazivanje rušenja aplikacije pomoću namjerno pogrešnih podataka kako bi se otkrili potencijalni propusti unutar aplikacije.
- Ponovna analiza mogućih područja napada - osigurava da su sve promjene unutar aplikacije uzete u obzir te analizira rezultate promjena koje su utjecale na područje mogućih napada.

- Izrada plana odgovora na incidente - izrada ove vrste plana ključno je za rješavanje novih napada. Uključuje identifikaciju hitnih kontakata i planova održavanja sigurnosti.
- Finalna provjera sigurnosti - uključuje provjeru modela prijetnji, rezultata upotrebe alata te reviziju sigurnosnih aktivnosti.
- Potvrda i arhiviranje izdanja - potvrda prije izdavanja osigurava da su zahtjevi sigurnosti postignuti. Arhiviranje je ključno za daljnje održavanje te uključuje svu specifikaciju, izvorne kodove, modele prijetnji, dokumentaciju, licence, planove odgovora te servisne uvjete.

Više informacija o prethodno navedenom moguće je pronaći na službenim stranicama *MS SDL-a* (vidi [6]).

4.2 OWASP CLASP

OWASP *Comprehensive Lightweight Application Security Process* ili kraće CLASP predstavlja metodu sigurnog razvoja softvera. Kao i prethodno opisana metoda, CLASP specificira aktivnosti koje članovi razvojnog tima moraju provesti kako bi razvili što sigurniji softver. Međutim, za razliku od MS SDL metode, CLASP metoda kategorizira aktivnosti po ulogama. Svaki suradnik unutar CLASP metode spada u jednu od sljedećih uloga:

- projektni menadžer,
- specifikator zahtjeva,
- arhitekt,
- dizajner,
- implementator,
- tester,
- revizor sigurnosti

Svaka od ovih uloga ima drugačije odgovornosti i zadatke koje provodi tijekom procesa razvoja. Tablica 4.2 prikazuje raspored aktivnost po pojedinim ulogama. Više informacija dostupno je na službenim stranicama organizacije *OWASP* (vidi [2]), a dodatni opisi pojedinih aktivnosti mogu se pronaći u knjizi ”Developing and evaluating security-aware software systems”. (vidi [10])

Aktivnost	Uloge
Uspostaviti svijest o sigurnosti programa	Projektni menadžer
Modeliranje mogućih prijetnji	Revizor sigurnosti
Kontrola razine sigurnosti izvornog koda	Revizor sigurnosti u suradnji s implementatorom i dizajnerom
Identificirati, implementirati i provesti testove sigurnosti	Tester
Verifikacija sigurnosnih atributa resursa	Tester
Istražiti i pridružiti sigurnosno stanje pojednim tehnološkim rješenjima	Dizajner
Identificirati globalnu politiku sigurnosti	Specifikator zahtjeva
Identificirati resurse i granice povjerenja	Arhitekt u suradnji sa specifikatorom zahtjeva
Identificirati uloge korisnika i mogućnosti resursa	Arhitekt u suradnji sa specifikatorom zahtjeva
Navesti operativno okruženje	Specifikator zahtjeva u suradnji s arhitektom
Opis detalja nepravilnog korištenja slučajeva	Specifikator zahtjeva
Identifikacija mogućih područja napada	Dizajner
Dokumentiranje zahtjeva vezanih uz sigurnost	Specifikator zahtjeva u suradnji s arhitektom
Uključivanje sigurnosnih principa u dizajn	Dizajner
Bilježiti klase dizajna sa sigurnosnim svojstvima	Dizajner
Implementirati i razraditi politiku resursa i sigurnosne tehnologije	Implementator
Implementirati sučelja ugovora	Implementator
Integrirati sigurnosnu analizu u izvor upravljanja procesom	Implementator
Potpisivanje izvornog koda	Implementator
Upravljanje procesom razotkrivanja sigurnosti	Projektni menadžer u suradnji s dizajnerom
Riješanje prijavljenjih sigurnosnih problema	Dizajner
Praćenje sigurnosti	Projektni menadžer
Specificiranje sigurnosnih postavki baze	Dizajner baze
Izrada operativnog sigurnosnog vodiča	Dizajner, arhitekt i implementator

Tablica 4.2: Podjela aktivnosti na uloge unutar CLASP metode.

Poglavlje 5

Studijski primjer

U ovom poglavlju najprije opisujemo alate i aplikaciju koja je korištena u svrhu izrade studijskog primjera. U nastavku opisujemo studijski primjer napada grubom silom pomoću *Burp Suite* skupine alata a potom primjer testiranja otpornosti aplikacije na napad ugnježđenim SQL naredbama.

5.1 Opis korištenih alata i aplikacije

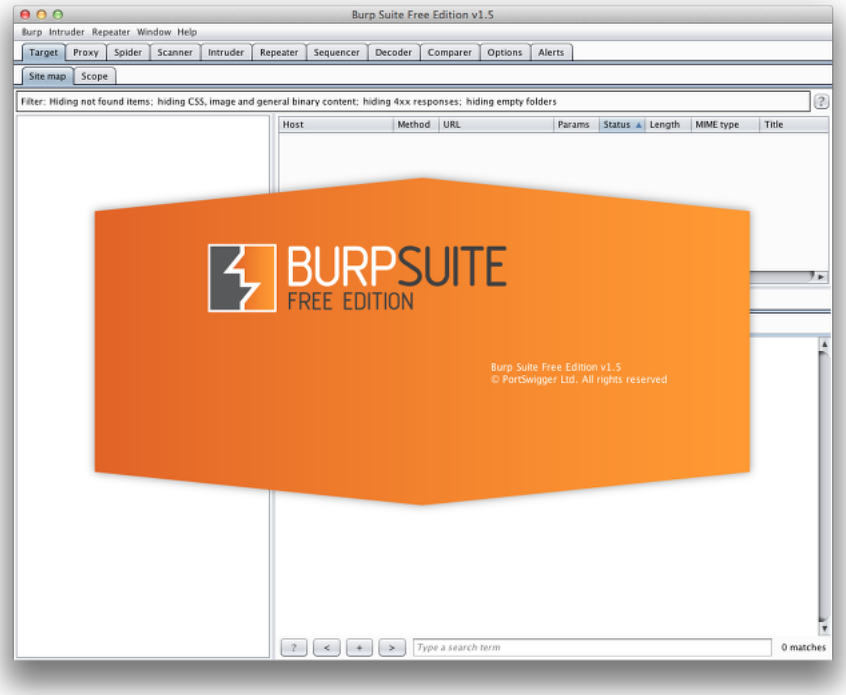
Za potrebe studijskog primjera koristit ćemo besplatnu aplikaciju *Damn Vulnerable Web App* (verzija 1.9) i besplatnu platformu *Burp Suite* (verzija 1.7.05) za testiranje sigurnosti prije spomenute web aplikacije. Primjere izvodimo koristeći Firefox web preglednik.

Damn Vulnerable Web App predstavlja ranjivu web aplikaciju koja koristi PHP i MySQL. Glavni cilj ove aplikacije jest testiranje alata i vještina programera u legalnom okruženju. Pomoću tih alata i vještina razvojni tim lakše može testirati određene propuste te ih i u konačnici naučiti ukloniti.

Slika 5.1: DVWA aplikacija

Glavni razlog upotrebe ove aplikacije unutar studijskog primjera jest mogućnost pregleda izvornog koda te mogućnost postavljanja razine sigurnosti web aplikacije. Razine sigurnosti koje su omogućene su: ”*low*”, ”*medium*”, ”*high*”, ”*impossible*”.

Burp Suite predstavlja platformu baziranu na Javi koja objedinjuje alate namijenjene za testiranje sigurnosti web aplikacije.



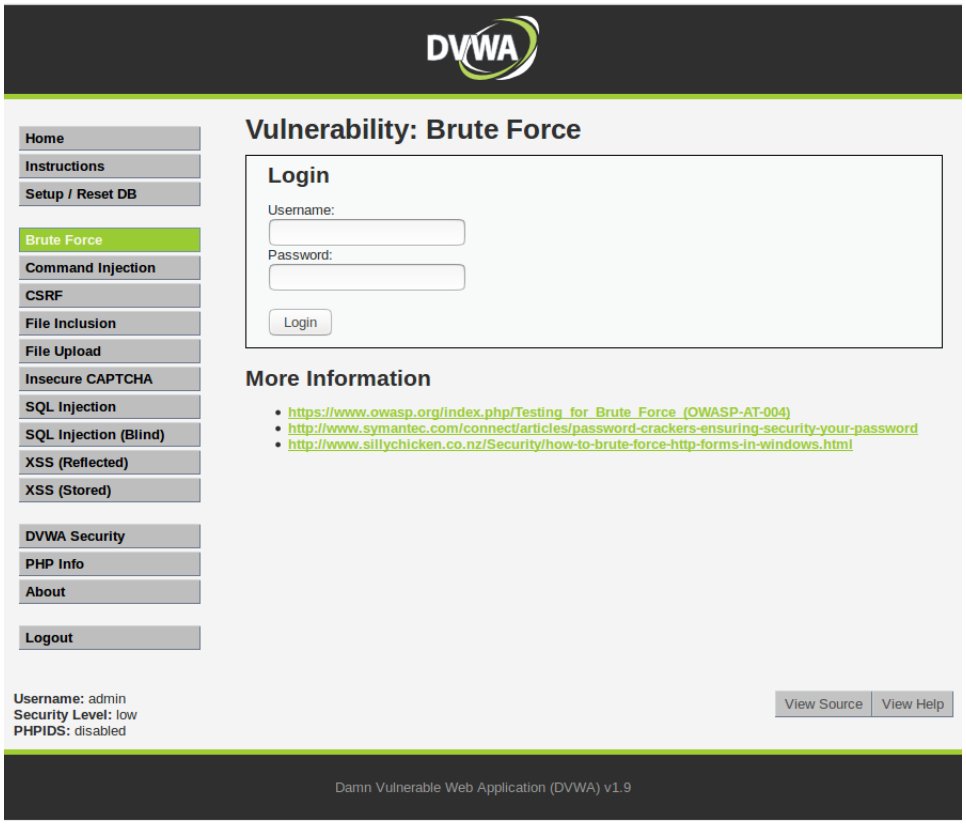
Slika 5.2: Burp Suite

Prije početka testiranja web aplikacije, potrebno je povezati *Burp Suite* i web preglednik Firefox. To radimo sljedećim koracima:

1. U Firefox pregledniku odlazimo na ”*Preferences*”/”*Advanced*”.
2. Odaberemo karticu ”*Network*” i kliknemo ”*Settings*”.
3. Označimo ”*Manual proxy configuration:*”
4. Za *HTTP proxy* unosimo vrijednost ”127.0.0.1”, dok za *Port* unosimo ”8080”.
5. Obrišemo sve iz *No Proxy for*.

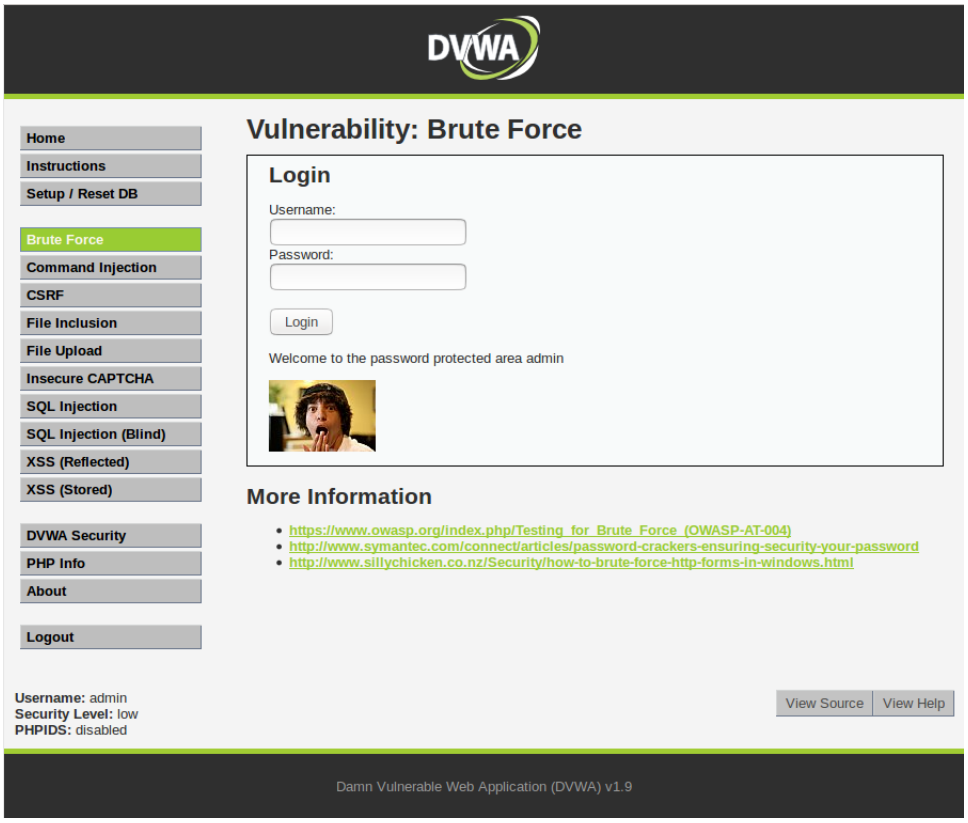
5.2 Primjer napada grubom silom pomoću Burp Suite

Napad izvodimo na prije spomenutoj aplikaciji s postavkama sigurnosti na razini: ”*low*”. Najprije otvorimo web aplikaciju u web pregledniku Firefox. Zatim kliknemo na poveznicu ”*Brute Force*” koja nas vodi do stranice nad kojom ćemo vršiti napad (vidi sliku 5.3).



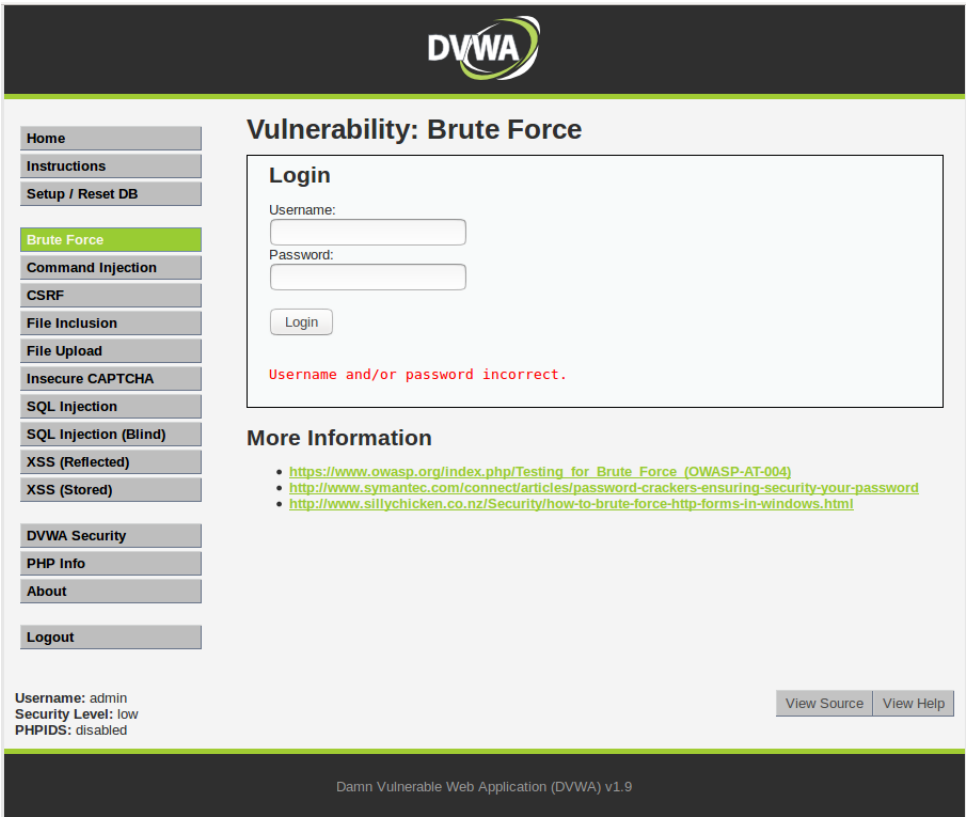
Slika 5.3: DVWA Login stranica za testiranje napada grubom silom

Nakon što smo pristupili traženoj stranici, primijenit će se test nad ponašanjem mehanizma autentifikacije. Najprije unosimo točne podatke i pratimo ponašanje aplikacije. U ovom slučaju koristimo korisničko ime ”admin” i lozinku ”password”. Na slici 5.4 prikazan je odgovor aplikacije.



Slika 5.4: Uspješna prijava korisnika

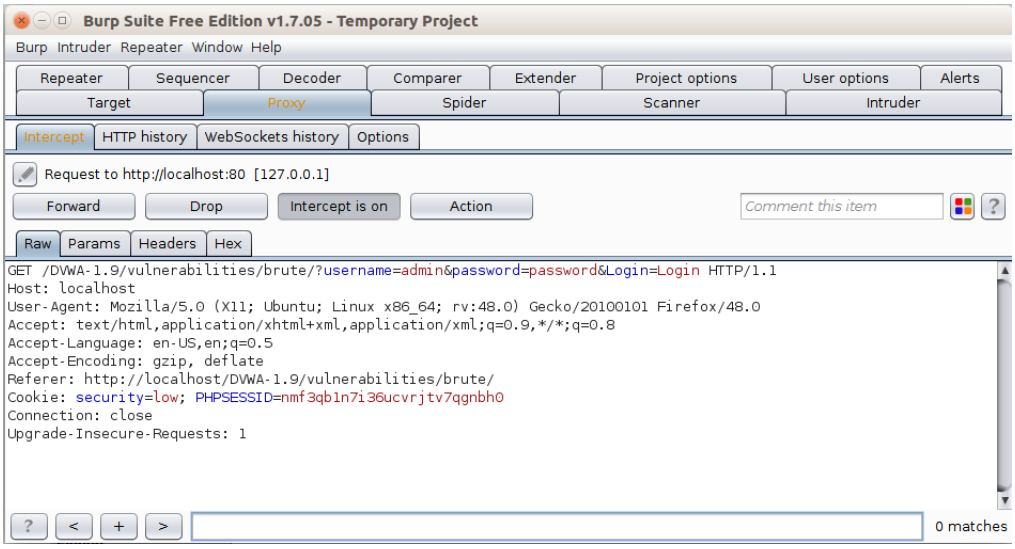
Sada kada nam je jasan odgovor za točne podatke, isti postupak ponavljamo s netočnim podacima. Na primjer, unosimo korisničko ime ”Goran” i lozinku ”npr123abc”. U tom slučaju dobivamo odgovor kao na slici 5.5.



Slika 5.5: Neuspješna prijava korisnika

Za kraj testiranja, ponovimo ovaj zadnji postupak unosa netočnih podataka za isto korisničko ime nekoliko puta. Aplikacija u svim slučajevima vraća isti odgovor što znači da nema dobar mehanizam zaštite od napada grubom silom. Postupak napada pomoću *Burp Suite* skupine alata izvršavamo sljedećim koracima:

1. Pristupimo stranici za prijavu kao na slici 5.3.
2. Pokrenemo Burp Suite gdje odaberemo karticu ”*Proxy*” i zatim karticu ”*Intercept*”.
3. Kliknemo na gumb ”*Intercept is off*”, tako da postane ”*Intercept is on*”.
4. Sada odlazimo opet na stranicu za prijavu korisnika te unesemo točne podatke za prijavu i pošaljemo zahtjev.
5. Unutar *Burp Suite* dobivamo podatke kao sa slike 5.6.



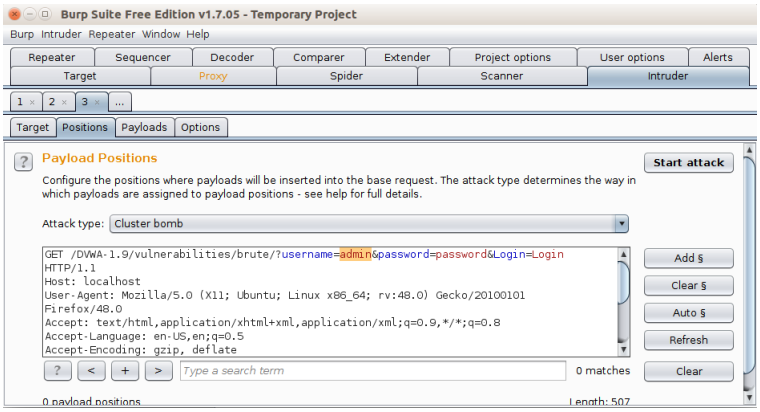
Slika 5.6: Burp intercept

6. Bilo gdje unutar sadržaja kartice ”*Raw*” kliknemo desnom tipkom miša i označimo ”*Send to Intruder*” te ugasimo presretanje zahtjeva.

7. Odlazimo na karticu ”*Intruder*” gdje odaberemo karticu ”*Positions*”.

8. Ovdje odaberemo tip napada ”*Cluster bomb*” te kliknemo na gumb ”*Clear*”.

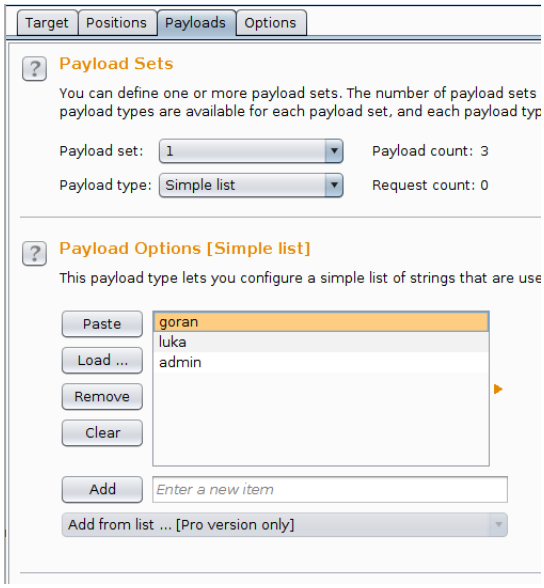
9. Označimo vrijednost ”admin” i kliknemo gumb ”*Add*” kao što je prikazano na slici 5.7.



Slika 5.7: Dodavanje pozicije podatka koji pogađamo

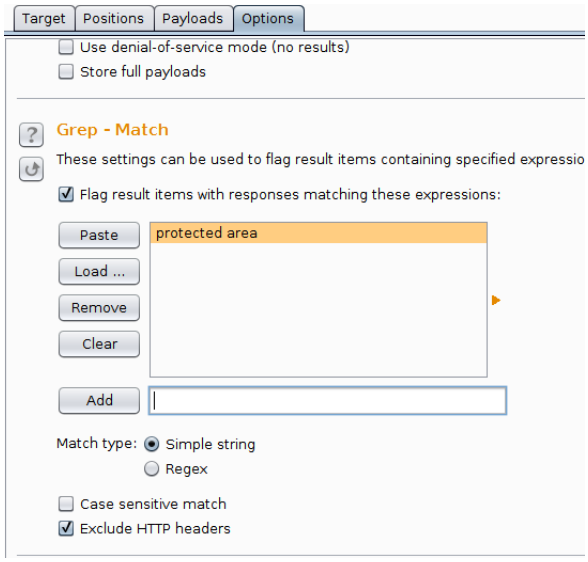
10. Postupak iz prethodnog koraka obavimo i za drugu vrijednost.

11. Odlazimo na karticu ”*Payloads*” gdje postavljamo skup vrijednosti za pogađanje svake od traženih informacija. Na slici 5.8 prikazan je skup vrijednosti za korisničko ime.



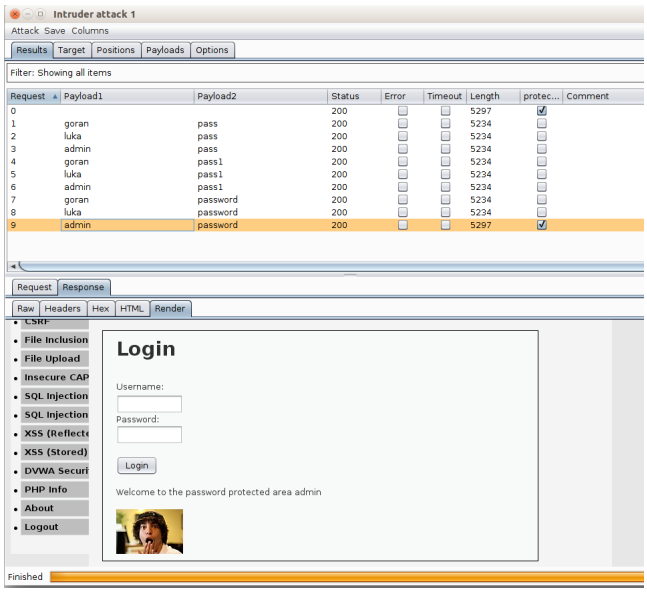
Slika 5.8: Dodavanje skupa vrijednosti pomoću kojeg pogađamo vrijednosti

12. Unutar kartice ”*Options*” možemo postaviti vrijednosti kao što su vremenski interval između dva napada. U našem slučaju (vidi sliku 5.4) znamo sadržaj poruke koja se ispisuje u slučaju ispravne prijave korisnika pa dio te poruke unosimo kao na slici 5.9. Na temelju toga određivat će se uspješnost napada.



Slika 5.9: Dodavanje skupa vrijednosti pomoću kojeg pogađamo vrijednosti

13. Na kraju pokrenemo napad klikom na gumb ”*Start Attack*”. Na slici 5.10 su prikazani dobiveni rezultati gdje možemo primjetiti da je za korisničko ime ”admin” i lozinku ”password” napad uspio.

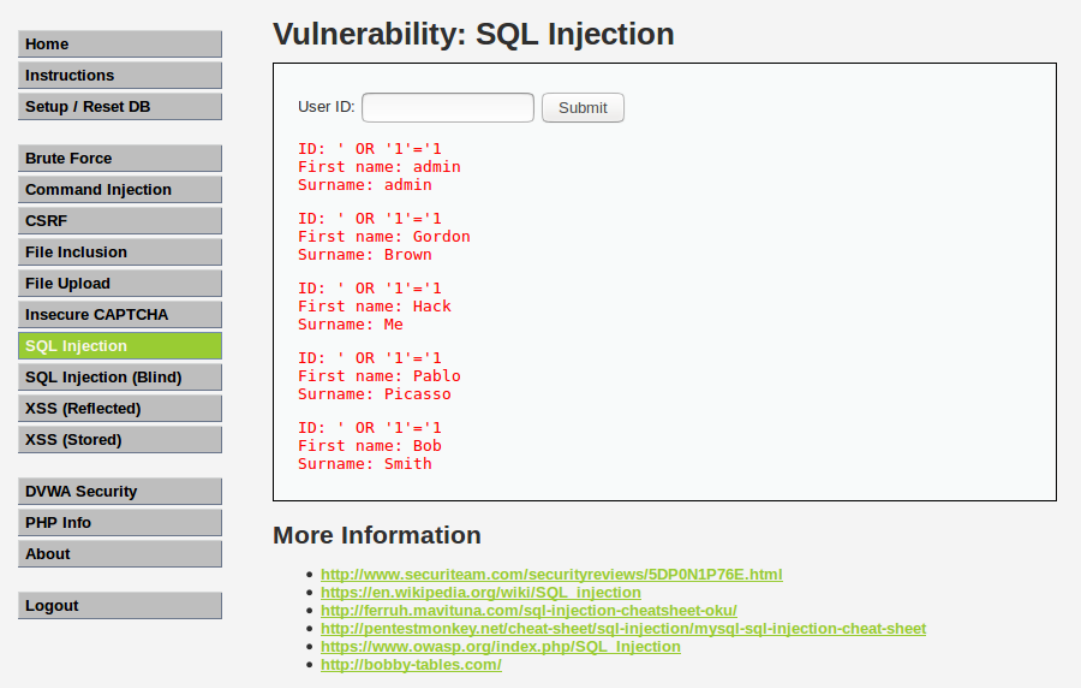


Slika 5.10: Uspješan napad grubom silom

5.3 Testiranje aplikacije na napad ugnježđenim SQL naredbama

Testiranje na napad ugnježđenim SQL naredbama započinjemo tako da pristupimo DVWA aplikaciji te kliknemo na poveznicu "*SQL Injection*". Testiranje provodimo na dvije razine sigurnosti aplikacije: "*low*" i "*impossible*", koje možemo promijeniti u postavkama aplikacije.

Promotrimo prvo primjer s "*low*" razinom sigurnosti. Test započinjemo tako da unesemo unutar forme "*User ID:*" vrijednost *' OR '1' = '1*. Slika 5.11 prikazuje rezultat tog unosa.



Slika 5.11: Aplikacija je podložna napadu ugnježđenim SQL naredbama

Odmah je vidljivo da je napad ugnježđenim SQL naredbama uspio za najjednostavniji test, štoviše ispisani su svi podaci korisnika.

Ako promijenimo razinu sigurnosti aplikacije na "*impossible*" i tada pokrenemo ovaj isti postupak, primjećujemo da napad nije uspio. Razlog tome vidljiv je na slici 5.12.

Slika 5.12: Izvorni kod aplikacije prikazuje upotrebu pripremljenih procedura

Vidimo da aplikacija na najvišem stupnju sigurnosti koristi pripremljene procedure kao vrstu obrane od napada ugnježđenim SQL naredbama. Stoga, nije ni čudo da naš napad nije imao efekta.

Zaključak

Cilj ovog diplomskog rada ”Sigurnost web aplikacija” bio je opis sigurnosnih principa koji se koriste za siguran rad web aplikacija, zatim klasifikacija i opis napada u vezi s pojedinim principima, zaštita principa i opis metoda razvoja sigurnih web aplikacija.

Klasifikacijom i opisom raznolikih vrsta propusta unutar web aplikacija istaknuta je važnost primjene i zaštite sigurnosnih principa. Iako je u posljednjih nekoliko godina razina svijesti o sigurnosti web aplikacija znatno porasla, i dalje smo svjedoci pojavljivanja web aplikacija koje ne udovoljavaju potrebnim sigurnosnim principima. Iz tog razloga potrebno je unutar procesa razvoja web aplikacija integrirati sigurnost. Pritom je moguće korištenje brojnih metoda za siguran razvoj web aplikacija, od kojih su dvije opisane u ovom radu.

Studijskim primjerom prikazane su dvije tehnike testiranja propusta web aplikacija. Pokazano je da pravilna upotreba alata i tehnike olakšava testiranje web aplikacija.

U budućnosti se očekuje daljnje smanjenje pojave već dobro poznatih napada kao što su napad ugnježđenim SQL naredbama te pojava novih vrsta napada koji prate evoluciju modernih web aplikacija. Navedeno će zasigurno rezultirati novim metodama i pravilima zaštite web aplikacija.

Bibliografija

[1] *Analiza XSS sigurnosnih propusta*, rujan 2016, http://www.cert.hr/sites/default/files/CCERT-PUBDOC-2006-05-157.pdf.

[2] *CLASP Concepts*, rujan 2016, https://www.owasp.org/index.php/CLASP_Concepts.

[3] *CSRF napadi*, rujan 2016, http://www.cert.hr/sites/default/files/NCERT-PUBDOC-2010-04-297.pdf.

[4] *Metode za poboljšanje sigurnosti web preglednika*, rujan 2016, http://www.cis.hr/www.edicija/LinkedDocuments/CCERT-PUBDOC-2009-09-276.pdf.

[5] *OWASP Testing guide v4*, rujan 2016, https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf.

[6] *Security Development Lifecycle*, rujan 2016, https://www.microsoft.com/en-us/sdl/.

[7] *Sigurnost sustava za upravljanje bazama podataka*, rujan 2016, http://www.cis.hr/www.edicija/LinkedDocuments/CCERT-PUBDOC-2006-11-171.pdf.

[8] J. D. Bokefode, S. A. Ubale, S. Apte Sulabha i D. G. Modani, *Analysis of DAC MAC RBAC Access Control based Models for Security*, International Journal of Computer Applications **104** (2014), br. 5, 6–13.

[9] J. Gregoire, K. Buyens, B. D. Win, R. Scandariato i W. Joosen, *On the secure software development process: CLASP and SDL compared*, Information and Software Technology **51** (2009), br. 7, 1152–1171.

[10] K. M. Khan, *Developing and evaluating security-aware software systems*, IGI Global, 2012.

[11] R. Manger, *Baze podataka*, Element, Zagreb, 2014.

[12] ———, *Softversko inženjerstvo*, Element, Zagreb, 2016.

[13] T. Pavic i L. Jelenkovic, *Autentifikacija i autorizacija korisnika na jednom mjestu*, Proceedings of the Information Systems Security, MIPRO 2007, 2007, str. 150–155.

[14] D. Stuttard i M. Pinto, *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*, John Wiley & Sons, Indianapolis, 2011.

[15] B. Sullivan i V. Liu, *Web Application Security, A Beginner's Guide*, McGraw Hill Professional, New York, 2011.

[16] H. Wu i L. Zhao, *Web Security: A WhiteHat Perspective*, CRC Press, 2015.

Sažetak

Sigurnosni principi web aplikacija zasnovani su na autentifikaciji, autorizaciji, upravljanju sesijom, sigurnom radu baze podataka te sigurnom radu web preglednika. Sesija objedinjuje proces autentifikacije i autorizacije, dok baze podataka predstavljaju nezaobilazan dio modernih web aplikacija koje su dostupne pomoću web preglednika. Napadi na web aplikaciju mogući su na bilo kojoj sastavnici njezinih sigurnosnih principa te sukladno pojedinoj vrsti napada postoje adekvatni načini zaštite. Osim toga, ključan korak koji se mora izvršiti u samom početku razvoja aplikacije jest integracija sigurnosti unutar razvoja web aplikacija. Na taj način se osigurava ušteda vremena i cijene razvoja te izbjegava loš pristup traženja propusta i njihovog ispravljanja. U radu se spominju Microsoftov *Security Development Lifecycle* te *Comprehensive Lightweight Application Security Process* organizacije OWASP. Provedeni studijski primjer napada grubom silom pokazuje primjer automatiziranog napada dok testiranje aplikacije na napad ugnježđenim SQL naredbama dokazuje pravilo da uporaba pripremljenih naredbi predstavlja dobru vrstu obrane od takvih napada.

Summary

Web applications security principles are based on authentication, authorization, session management, database secure work and web browser secure work. Session combines authentication and authorization process, whilst databases represent inevitable part of modern Web applications which are available through Web browser. Web application attacks are possible on any part of its security principles. For each particular type of attack there are adequate protection methods and procedures. Furthermore, the key step that ought to be firstly executed when developing application is security integration within the web application development. In that manner, development time and money saving is assured and unsatisfactory approach of gap finding and correcting is avoided. Hereunder are also mentioned Microsoft's Security Development Lifecycle and Comprehensive Lightweight Application Security Process by OWASP organization. Case study of brute-force attack points out a kind of automated attack, while application testing on SQL injection attack proves the rule that utilization of prepared statements represents a proper kind of defense for this type of attacks.

Životopis

Rođen sam 25. rujna 1990. godine u Čakovcu gdje sam pohađao 3. osnovnu školu Čakovec te gimnaziju "Josip Slavenski" matematičkog usmjerenja. Kroz osnovnoškolsko obrazovanje osvojio sam prvo mjesto na županijskom natjecanju iz tehničke kulture 2005. godine te nastupio na državnom natjecanju 2005. godine u Kraljevici. 2009. godine upisao sam Prirodoslovno-matematički fakultet odsjek matematika u Zagrebu na kojem sam 2014.godine stekao titulu sveučilišnog prvostupnika edukacije matematike. Nakon stjecanja titule prvostupnika upisao sam Diplomski sveučilišni studij Računarstvo i matematika na istoimenom fakultetu.

Od svoje rane mladosti bavio sam se redovito rukometom te sam osvajao brojna natjecanja širom Republike Hrvatske, ali i diljem Europe.

Engleski jezik koristim aktivno, a njemački jezik na razini A1.

Najdraži hobiji su mi bicikliranje, izrada 3D modela pomoću računala te rad u obiteljskom voćnjaku.