UDK: 007:004.056.5
343.533::004

Pregledni naučni rad

UPRAVLJANJE BEZBEDNOŠĆU U CYBER PROSTORU I MEHANIZMI ZAŠTITE WEB APLIKACIJA

Prof. dr Muzafer Saračević*
MSc Muhedin Hadžić*

Apstrakt

U ovom radu su razmatrani osnovni mehanizmi zaštite i upravljanja bezbednošću u Cyber prostoru. Osnovni problem zaštite web aplikacija se ogleda u činjenici da su svi unosi podataka od strane korisnika nesigurni i ne treba im verovati. Ovaj pristup je razvio veliki broj zaštitnih mehanizama koje aplikacije koriste da bi se odbranile od napadača. Najozbiljniji napadi na web aplikacije su napadi koji otkrivaju osetljive podatke ili omogućuju neograničen pristup sistemu i resursima.

Za mnoge kompanije svaki napad koji rezultuje zastoj sistema je kritičan događaj. Kada se govori o zaštiti i bezbednosnim rizicima, mora se znati da je to neprestana borba između onih koji otkrivaju propuste i napadaju aplikacije i onih koji prave mehanizme odbrane. Ono što je danas aktuelno kao metod napada, u bliskoj budućnosti može biti modifikovano ili zamenjeno nekom drugom metodom.

U ovom radu akcenat je stavljen na napade koji se mogu koristiti u cilju iscrpljenosti resursa servera na kojem se aplikacija nalazi, što ima za rezultat nemogućnost pristupa aplikaciji od strane korisnika.

Ključne reči: Cyber bezbednost, Mehanizmi zaštite, Napadi na internetu, Ranjivost web aplikacija.

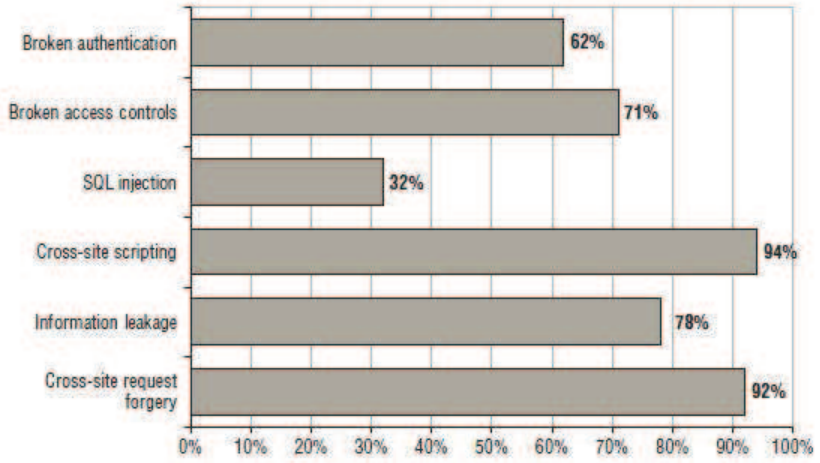
UVODNA RAZMATRANJA

Postoji široko rasprostranjena svest o problemima zaštite web aplikacija. Korisnici se često pozivaju na proveru sertifikata, jer žele da svoje informacije poveravaju ozbiljnim organizacijama koje njihove podatke čuvaju iza naprednih kriptografskih protokola. Na osnovu ispitivanja sigurnosnih propusta web aplikacija u periodu od 2007. do 2015. godine, možemo izdvojiti sledeće kategorije napada [7]:

- *Nedostaci autentifikacije* obuhvataju ugroženost i zloupotrebu raznih nedostataka mehanizma prijavljivanja na sistem. Ovde se najčešće koristi metoda nasumičnog nagađanja šifre korisnika.

* Departman za računarske nauke, Univerzitet u Novom Pazaru, e-mail: muzafers@uninp.edu.rs.
* Departman za računarske nauke, Univerzitet u Novom Pazaru, e-mail: muhedin.hadzic@uninp.edu.rs.

- *Nedostaci kontrole pristupa* predstavljaju slučaj kada aplikacija nije u mogućnosti da zaštiti pristup podacima i resursima. Ovde je napadaču omogućeno da dođe do osetljivih podataka korisnika na serveru, ili da izvršava privilegovane akcije.
- *SQL injection* omogućava napadaču da unosom izmeni SQL upit namenjen bazi podataka. Na taj način napadač može biti u mogućnosti da dođe do podataka koji se nalaze smešteni u bazi podataka.
- *Cross-site scripting* omogućava napadaču da napada druge korisnike aplikacije, kao i dobijanje pristupa njihovim podacima ili vršenje nedozvoljenih radnji u njihovo ime.
- *Curenje informacija* uključuje slučajeve gde aplikacija prikazuje osetljive informacije koje su korisne napadaču u razvoju mehanizama protiv same aplikacije, štampanjem poruka o greškama i drugo slično ponašanje.
- *Cross-site falsifikovanje zahteva* omogućava napadaču da izvršava neželjene akcije u ime korisnika aplikacije. Napad se realizuje na osnovu posete zlonamernog sajta od strane žrtve, nakon čega korisnički browser obavlja određene akcije koje korisnik ne namerava da uradi [8].



Slika 1. Učestalost ranjivosti web aplikacija u periodu od 2007. do 2016. Godine

U praksi sve aplikacije koriste konceptualno slične mehanizme odbrane, ali se razlikuju u efikasnosti i implementaciji. Mehanizam odbrane web aplikacija pokriva sledeće osnovne elemente:

- *Upravljanje korisničkim pristupom podacima,* da bi se izbeglo da napadač dođe do neovlašćenog pristupa.
- *Upravljanje podacima koje korisnik unosi u aplikaciju*, da bi se izbegao neželjen unos podataka i neželjeno ponašanje aplikacije.
- *Upravljanje napadačima*, da bi se osiguralo da se aplikacija ponaša normalno u slučajevima napada i ispravno koristi odgovarajući mehanizam odbrane.

- *Omogućavanje administratoru da prati aktivnosti na web aplikaciji*, da bi mogao da uvidi propuste na istoj i unapredi zaštitu na veći nivo bezbednosti.

UPRAVLJANJE KORISNIČKIM PRISTUPOM

Najznačajnija potreba koju svaka aplikacija mora da zadovolji je kontrolisanje korisničkih pristupa. U osnovi, postoji nekoliko različitih kategorija korisnika, na primer: anonimni korisnici, obični korisnici i administratori. Svaka grupa korisnika ima pristup različitim podacima i resursima aplikacije. Kao na primer, korisnici web mejl aplikacije bi trebalo da imaju pristup čitanja svojih mejlova, ali ne i mejlova drugih korisnika. Mnoge aplikacije upravljaju pristupima koristeći sledeća tri mehanizma: autentifikacija, upravljanje sesijama i kontrola pristupa.

Autentifikacija

Autentifikacija je najosnovniji oblik kontrole pristupa web aplikaciji. Ona se koristi da proveri identintet korisnika koji se predstavlja aplikaciji. Bez ovog mehanizma, aplikacija bi tretirala sve korisnike kao anonimne. Većina današnjih aplikacija koriste autentifikacioni model koji se ogleda u tome da korisnik dostavlja korisničko ime i lozinku, nakon čega aplikacija proverava validnost unetih podataka. Kod aplikacija kod kojih je potreban veći nivo sigurnosti ovaj model je proširen dodavanjem višestepenog procesa prijavljivanja. Ukoliko je potrebno još više osigurati pristup, neki drugi modeli autentifikacije se mogu koristiti koji su pre svega bazirani na klijentskom sertifikatu, kao što su pametne kartice, unos odgovora na sigurnosno pitanje i sl. Osim procesa prijave, autentifikacijski mehanizmi omogućuju mnoge druge funkcije, kao što su: registracija, oporavak računa, promena lozinke.

Bez obzira na jednostavnost, autentifikacioni mehanizmi pate od velikog broja grešaka u njihovom dizajnu i implementaciji. Te greške mogu omogućiti napadaču da dođu do podataka o korisničkim imenima drugih korisnika, nagađaju njihove lozinke, ili da zaobiđu funkciju prijavljivanja iskorišćavanjem logičkih grešaka.

Upravljanje sesijama

Nakon što provera identiteta korisnika na sistem prođe uspešno, korisniku će biti omogućen pristup raznim stranicama, funkcionalnostima i podacima slanjem HTTP zahteva. U isto vreme, web aplikacija prima zahteve i od drugih korisnika, neki od njih su prijavljeni na aplikaciju, dok su neki anonimni. Da bi aplikacija mogla da zna kojim korisnicima će omogućiti pristup određenim resursima, a kojima će tu privilegiju uskratiti, ona mora da identifikuje i obradi svaki poslan zahtev od strane korisnika.

U praksi sve web aplikacije obavljaju ove poslove kreiranjem sesija za svakog korisnika posebno izdavajući tokene koji identifikuju sesije. Token predstavlja zapis koji aplikacija mapira u sesiju. Kada korisnik prihvati token, browser ga automatski šalje nazad prema serveru. Ovo se ponavlja prilikom svakog zahteva od strane korisničkog web browsera. Na ovaj način se omogućuje aplikaciji da prepozna koji zahtev je došao od kog korisnika. Sesije imaju određeni rok trajanja i ako korisnički browser duže vreme ne šalje zahteve aplikaciji, sesija se prekida.

Sigurnost sesija u velikoj meri zavisi od sigurnosti tokena. Ukoliko je token ugrožen napadač se može predstavljati kao korisnik i može koristiti aplikaciju u ime legalnog korisnika. Mali broj aplikacija ne koristi tokene već identifikuje korisnike kroz višestruke HTTP zahteve. Tada browser automatski ponovo podnosi korisničke podatke sa svakim novim zahtevom i na taj način omogućuje aplikaciji da identifikuje korisnika. Postoje i aplikacije koje korisničke informacije ne čuvaju na serveru, nego na klijentskoj strani, i tada se ti podaci kriptuju [3].

Kontrola pristupa

Poslednji korak u procesu upravljanja korisničkim pristupom je ispravna odluka o tome da li će svaki od individualnih zahteva biti dozvoljen ili odbijen. Ukoliko mehanizam odlučivanja funkcioniše ispravno, aplikacija će znati identitet korisnika od koga je zahtev pristigao, i na osnovu toga će znati da li će korisniku obezbediti pristup podacima koje on potražuje. Mehanizmu kontrole pristupa je obično potrebno da se implementira naprednija logika sa naprednijim razmatranjima koji se odnose na kontrolu pristupa različitim područjima aplikacije, privilegijama i podacima.

Zbog kompleksne prirode zahteva kontrole pristupa, ovaj mehanizam je često izvor ranjivosti koje omogućuju napadaču neovlašćen pristup podacima i ostalim resursima aplikacije. Programeri vrlo često prave pogrešne pretpostavke u vezi interakcije korisnika sa aplikacijom, i vrlo često prave greške i propuste u proveri kontrole pristupa za pojedine funkcije aplikacije. Ispitivanje ovih ranjivosti je naporan posao, ali se na kraju uvek isplati.

UPRAVLJANJE PODACIMA U APLIKACIJI

Veliki broj različitih napada na aplikacije uključuju upravo prihvatanje neočekivanog unosa, koji ima cilj da proizvede ponašanje aplikacije koje nije osmišljeno od strane programera. U skladu sa tim, ključni uslov odbrane aplikacije je da aplikacija mora upravljati korisničkim unosima na siguran način. Ranjivosti zasnovane na osnovu unosa podataka se mogu javiti bilo gde u aplikaciji. Validacija unosa je jedna od najčešćih načina odbrane od ovog tipa napada.

Vrste i pristupi upravljanja unosima

Tipična web aplikacija obrađuje podatke koje je korisnik uneo na razne načine. Neka vrsta ulazne validacije može biti neizvodljiva ili poželjna za sva polja koja korisnik popunjava u formi aplikacije. U mnogim slučajevima, aplikacija može koristiti vrlo stroge provere za svaki od unosa. Sa druge strane, aplikacija mora da dozvoli veliki opseg mogućih unosa. Na primer, polje *adresa* u formi sa ličnim podacima može sadržati slova, brojeve, razmake i ostale karaktere. Za ovu stavku još uvek postoje određena ograničenja koja se mogu uvrstiti (npr. dužina unetog teksta ne sme biti veća od 40 karaktera, i ne sme koristiti nikakav HTML kod). U nekim situacijama, aplikacija će morati da prihvati proizvoljan unos od strane korisnika. Na primer, korisnik blog sajta može kreirati blog čija je tema "*napad na web aplikacije*". Postovi i komenatri koji se nalaze na blogu mogu imati štetne zapise u svom sadržaju (izvorni kodovi sa kojim je moguće izvršiti napad). Ono što se očekuje od aplikacije je da sačuva ove zapise u bazi podataka, zapiše ih na hard disku, i prikaže ih korisnicima na siguran način.

Uz sve različite tipove unosa koji korisnici mogu uneti koristeći formu aplikacije u svom browseru, tipična aplikacija prihvata i podatke čiji životni vek započije na serveru, pa se šalju klijentu tako da ih klijent može preneti nazad serveru sledećim zahtevom. To uključuje stvari kao što su *cookie* i skrivena polja unutar formi koja nisu vidljiva običnim korisnicima aplikacije, ali koje napadači mogu videti i menjati njihov sadržaj. U tim slučajevima, aplikacija često može izvesti veoma opširne provere dobijenih podataka. Na primer, od parametara se može zahtevati da sadrži neke specifične vrednosti kao što su *cookie* koji pokazuju informacije o jeziku korisnika ili ID broj kupca. Kada aplikacija detektuje da su podaci koje je napravio server menjani na način koji nije moguć od strane legitimnog korisnika i web browsera, to ukazuje na to da je korisnik ispitivao ili tražio ranjivost u aplikaciji. U takvim situacijama aplikacija bi najverovatnije trebalo da odbaci zahtev i zabeleži mogući napad.

Postoji veliki broj pristupa koji se odnose na problem rukovanja unosima. Svaki od njih je poželjno koristiti za neke tipove korisničkih unosa. Kombinacije više pristupa mogu biti poželjne u sprečavanju napada na aplikaciju. Postoji pet pristupa upravljanja unosima [6]:

1. Odbaci ono što se zna da je loše,
2. Prihvati ono što se zna da je dobro,
3. Saniranje,
4. Bezbedno rukovanje podacima,
5. Provera semantike.

Odbaciti ono što se zna da je loše

Ovaj pristup često sadrži "*crnu listu*" koja sadrži grupu reči ili obrazaca za koje se zna da se mogu iskoristiti u napadima. Validacioni mehanizam blokira bilo koji podatak koji se poklapa sa nekim od podataka iz crne liste i dopušta sve ostalo. Generalno, ovaj pristup se smatra za najmanje efikasnim iz dva razloga. Prvo,

ranjivost u aplikaciji se može iskoristiti korišćenjem različitih oblika unosa podataka, koji mogu biti kodirani ili predstavljeni na različite načine. Ovaj pristup je zadovoljavajući samo u jednostavnim slučajevima napada. Drugo, tehnike zloupotrebe aplikacija se stalno unapređuju i razvijaju. Nove tehnike napada koje se razvijaju najverovatnije neće biti blokirane od strane aplikacije koja koristi ovaj pristup, pošto se ne nalaze u crnoj listi.

Prihvatiti ono što se zna da je dobro

Ovaj pristup sadrži "*belu listu*" koja u sebi sadrži grupu reči, obrazaca ili grupu kriterijuma za koje se zna da odgovaraju dobroćudnom unosu. Validacioni mehanizam dopušta podatke koje odgovaraju podacima bele liste i blokira sve ostalo. Na primer, pre nego što dođe do podataka o kodu proizvoda u bazi podataka, aplikacija će izvršiti validaciju koja će proveriti da li zahtev sadrži samo alfanumeričke karaktere, i da li je unos dug tačno šest karaktera. Ovaj tip pristupa se koristi za rukovanje sa potencijalno opasnim unosima. Nije moguće uvek koristiti ovakav način zaštite, jer neka polja za unos imena i prezimena moraju podržavati neke druge znakove a to može biti iskorišćeno za napade.

Saniranje

Ovaj pristup je osmišljen za situacije gde je potrebno prihvatiti podatke za koje ne možemo da garantujemo da su sigurni. Umesto da odbaci unos, aplikacija ga sanira na nekoliko načina da bi sprečila mogućnost bilo kakvih neželjenih efekata. Potencijalni štetni karakteri će biti uklonjeni iz podatka, ostavljajući samo podatke za koje se zna da nisu štetni, ili se mogu kodirati ili "izbeći" pre dalje obrade.

Pristup baziran na saniranju je veoma efektivan i u mnogim situacijama se može koristiti kao osnovno rešenje u borbi protiv zlonamernih unosa. Na primer, ovo je uobičajena odbrana od *cross-site scripting* napada u HTML kodiranju kritičnih karaktera pre nego se implementiraju u stranici aplikacije. Efektivno saniranje može postati otežano u situacijama kada je više potencijalno zlonamernih podataka potrebno biti sadržano u jednom unosu.

Bezbedno rukovanje podacima

Veliki broj ranjivosti aplikacija je uzrokovano od strane nesigurne obrade unetih podataka. Ranjivost se često može izbeći ne samo proverom unosa nego i osiguravanjem da je obrada nad podacima sigurna. U nekim situacijama postoje određene sigurnosne programske metode koje su dostupne i osmišljene da izbegnu ovaj tip problema. Na primer, *SQL injection* napad može se sprečiti kroz pravilnu upotrebu parametarizovanih upita za pristup bazi podataka. Ovaj pristup se ne može koristiti nad svim zadacima koje aplikacija treba da zadovolji, ali je jedan od osnovnih mehanizama odbrane gde ga je moguće implementirati.

Provera semantike

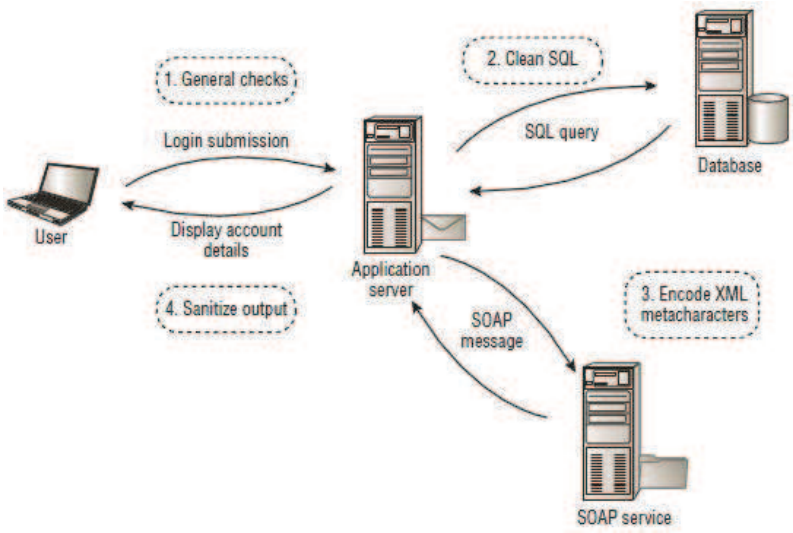
Pristupi koji su do sada objašnjeni odnose se na odbranu aplikacije od raznih tipova loših unosa podataka koji su osmišljeni sa ciljem da ometaju rad aplikacije. U nekim situacijama, ranjivost koju je napadač implementirao može biti identična unosu običnog korisnika. Ono što ga čini zlonamernim je okolnost pod kojom se on unosi. Na primer, napadač može tražiti pristup bankovnom računu drugog koristnika menjajući broj računa koji se nalazi u skrivenom polju u formi. Da bi se ovaj vid napada sprečio, aplikacija mora proveriti da li ID računa pripada korisniku koji je podneo zahtev za određenu akciju.

Granica validacije

Ideja o proveri validnosti podataka preko sigurnih granica je prihvatljiva. Provera validnosti unetih podataka implementirana na klijentskoj strani može povećati performanse aplikacije, sa druge strane ovaj pristup ne može osigurati zaštitu podataka koji se nalaze na serveru. Drugim rečima, provere sigurnosti podataka se ne mogu odvijati samo prilikom unosa podataka u poljima forme. Kompletna zaštita od svih napada koji su mogući je po nekad neizvodljiv.

Efektivniji model zaštite koristi koncept granične validacije. Ovde svaka individualna komponenta ili funkcionalna celina *server-side* aplikacije tretira njene ulaze kao da dolaze od potencijalnog zlonamernog izvora. Provera ispravnosti podataka je uključena na svakoj od ovih granica između klijenta i servera. Svaka komponenta je u stanju da brani sebe protiv specifičnih tipova ulaznih podataka koji mogu da budu zlonamerni. Kako podaci prolaze kroz različite komponente, validacija može biti sprovedena nezavisno od vrednosti koji je podatak imao u nekim pređašnjim proverama. Korisničko prijavljivanje na sistem prolazi kroz nekoliko faza obrade. U svakoj fazi se koristi odgovarajuća provera unetih podataka [4]:

- *Faza 1: Aplikacija dobija podatke od korisnika* - rukovodilac forme vrši proveru da li svaki od unetih podataka sadrži samo dozvoljene karaktere (da li podaci imaju dozvoljenu dužinu, sadrže samo dozvoljene karaktere itd.)
- *Faza 2: Aplikacija zatim obavlja SQL upit da bi potvrdila korisnikov identitet* - da bi izbegli SQL injection napad, svaki karakter u korisnikovom unosu koji se može uoptrebiti za napad na bazu podataka je potrebno izbeći pre nego se upit napravi.
- *Faza 3: Aplikacija šalje podatke od korisničkog računa SOAP servisu da bi preuzela dodatne informacije o profilu* - da bi se zaštitili od SOAP injection, svaki XML meta karakter koji je sadržan u korisničkom profilu je potrebno kriptovati.
- *Faza 4: Aplikacija prikazuje informacije o korisničkom profilu u pretraživaču* - da bi izbegli cross-site scripting napad [9], aplikacija kriptuje sve podatke od strane koristnika koje prikazuje na stranici.



Slika 2. Granica validacije u više koraka provere

Višekoračna validacija i kanonizacija

Najčešći problemi kod mehanizma za upravljanje unosom se javljaju kada se unetim podacima manipuliše kroz nekoliko koraka validacije. Ako se validacija ne upotrebljava pravilno, napadač može konstruisati unos da se maliciozni podaci mogu provući kroz mehanizam validacije. Jedan tip problema nastupa kada aplikacija pokuša da sanira unos korisnika uklanjajući ili kodirajući neke od znakova ili izraza. Na primer, aplikacija može pokušati da se odbrani od XSS napada tako da iz podataka koje je korisnik uneo ukloni izraze tipa *<script>*. Napadač u ovom slučaju može da zaobiđe filter tako što unosi sledeći podatak: <scr<script>ipt>. Kada se blokirani izraz ukloni, podaci koji ga okružuju se stapaju u jedan i opet formiraju maliciozni izraz. Razlog tome je taj što se filter ne koristi rekurzivno.

Slično ovome, ako se koristi više od jedne validacije unetih podataka od strane korisnika, napadač može da iskoristi redosled obavljanja samih validacija da bi zaobišao filter. Na primer, ako aplikacija prvo uklanja *script* tagove rekurzivno, a potom uklanja apostrofe, sledeći iskaz bi mogao da porazi validaciju: <scr"ipt>

Drugi problem nastaje kod kanonizacije podataka. Kada se uneti podatak pošalje sa korisničkog browser-a, on se može kriptovati na razne načine. Šeme kriptovanja postoje kako bi obični znakovi, slova i binarni podaci mogli biti sigurni preko HTTP protokola. Kanonizacija je proces koji se koristi za pretvaranje ili dekriptovanje podataka u normalne znakove i slova. Ako je bilo koja kanonizacija uspešna nakon što je primenjen filter za podatke unosa, napadač može biti u mogućnosti da koristi prikladnu šemu kriptovanja i da zaobiđe mehanizam za

validaciju. Na primer, aplikacija može pokušati odbranu od nekog SQL injection napada tako što uklanja apostrofe iz unetog podatka.

Uklanjanje problema sa višekoračnom validacijom i kanonizacijom može biti po nekad veoma teško, i ne postoji jedinstveno rešenje za ovaj problem. Jedan način je da se saniranje obavlja rekurzivno i završi tek onda kada se nema više šta modifikovati. Međutim, ako saniranje koristi zaobilaženje problematičnih karaktera, ovo može rezultovati beskonačnu petlju.

UPRAVLJANJE AUTENTIFIKACIJOM

Autentifikacija je prva linija u odbrani protiv neovlašćenog pristupa. Ako napadač može prevariti ovu zaštitu, biće u mogućnosti da neovlašćeno dođe do podataka i kontrola aplikacije. Bez snažne provere autentifikacije, nijedan drugi zaštitni mehanizam ne može biti efikasan. Bilo ko može uneti proizvoljne reči u login formi i nagađati password. Sa druge strane, suptilni defekti se mogu javiti u načinu na koji aplikacija obrađuje unete podatke.

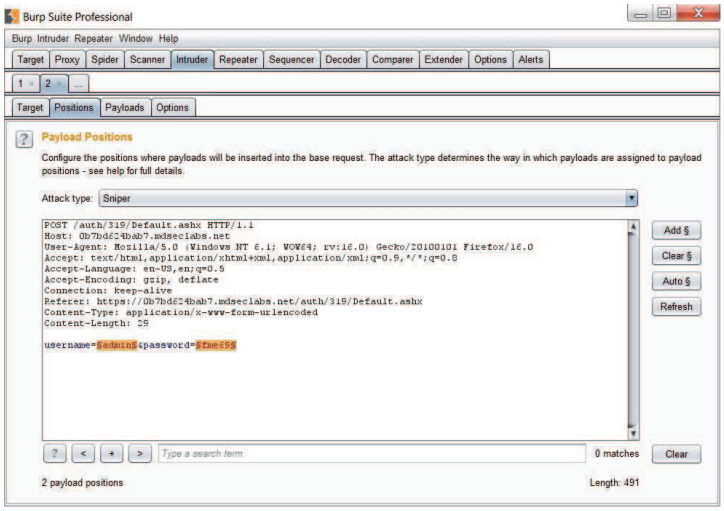
Loše lozinke

Mnoge aplikacije nemaju ili imaju veoma loše izgrađenu kontrolu kvaliteta korisničkih lozinki. Često se mogu naći aplikacije koje dopuštaju lozinke koje su veoma kratke, sadrže proste reči koje je lako pogoditi ili iste kao korisničko ime. Kod ovakvih tipova lozinki, napadač može sa lakoćom pogoditi korisničku lozinku, što će mu omogućiti neovlašćen pristup aplikaciji.

Brute Force

Login funkcionalnost predstavlja otvoreni izazov napadaču da pokuša pogoditi korisničko ime i lozinku i na taj način omogući sebi pristup aplikaciji. Ukoliko aplikacija dozvoljava napadaču da pokušava prijavljivanje više puta sa drugačijim podacima, dok ne pogodi prave podatke, ovo predstavlja veoma veliku ranjivost. Čak i napadač amater može odraditi ovaj vid napada.

Svaki ozbiljan napadač koji koristi ovaj vid napada će se koristiti automatizovane tehnike za nagađanje lozinki. One koriste listu mogućih lozinki. Gledajući današnju brzinu Interneta i performanse računara, moguće je napraviti na stotine zahteva prijave u jednoj minuti. U nekim aplikacijama, postoje kontrolni mehanizmi koji ne dozvoljavaju napadaču da nagađa lozinku više puta. Na primer, može se kreirati *cookie* koji u sebi nosi informaciju o broju neuspešnih prijavljivanja od strane jednog korisnika. Kada broj neuspešnih prijavljivanja dostigne svoj limit, aplikacija će u tom slučaju onemogućiti korisniku (napadaču) dalje prijavljivanje. Ovaj tip zaštite može osigurati aplikaciju samo u slučaju ako se za napad koristi web browser, ali se ovaj tip zaštite takođe može i zaobići u napadima. Na slici je prikazan uspešno obavljen *Brute Force* napad koristeći ***Burp Intruder*** alat.



Slika 3. Uspešno okončano nagađanje korisničkog imena i lozinke

Kada logovanje protekne neuspešno, neke od aplikacija prikazuju korisniku koja informacija je bila netačna prilikom podnošenja zahteva prijave. Na taj način napadač može doznati koje od podataka je potrebno izmeniti. Neke aplikacije zahtevaju i neke dodatne informacije kao što su: datum rođenja, mesto ili PIN.

Promena lozinke

Mnoge web aplikacije ne dozvoljavaju korisnima mogućnost promene lozinke. Ova funkcionalnost je neophodna u dobro dizajniranim autentifikacionim mehanizmima iz dva razloga:

- Periodična promena lozinke smanjuje mogućnost otkrivanja lozinke od strane napadača. Na ovaj način se smanjuje mogućnost napada sa automatizovanim alatima koji nagađaju lozinku.
- Korisnici koji su saznali da je njihova lozinka otkrivena imaju potrebu za brzom zamenom iste i na taj način onemoguće napadaču nedozvoljeni pristup njihovim podacima.

Iako je promena lozinke neophodan deo autentifikacionog mehanizma, on takođe može biti i veoma ranjiv. Slabe tačke koje su namerno bile osigurane u kreiranju funkcije prijavljivanja se mogu javiti u funkciji za promenu lozinke. Funkcije promene lozinke u mnogim web aplikacijama su dostupne bez ikakve autentifikacije i omogućavaju sledeće:

- Prikazuju poruke o greškama i na taj način odaju informacije da li je korisničko ime validno,
- Omogućavaju neograničen broj nagađanja lozinke,

- Proveravaju da li je nova lozinka ista kao i stara, i na taj način omogućava napadaču da otkrije staru lozinku.
-

Zaboravljena šifra

Kao i funkcije promene lozinke, mehanizam za oporavak lozinke takođe može biti iskorišćen od strane napadača. Sam dizajn funkcija za oporavak lozinke predstavlja jedan od najslabijih delova autentifikacije. Postoji nekoliko propusta na koje možemo naići:

- U slučajevima zaboravljene lozinke korisnik treba da odgovori na neka zaštitna pitanja. Na ovaj način napadaču je puno lakše da dođe do nečijeg korisničkog računa.
- Kao što je bio slučaj kod promene lozinke, ukoliko su programeri ograničili broj zahteva za prijavu od strane jednog korisnika na glavnoj stranici za prijavu, napadač može odraditi brute force napad nad poljem za odgovor.
- U nekim aplikacijama, oporavak lozinke koristi umesto odgovora na postavljeno pitanje nagoveštaj lozinke koji korisnik unosi prilikom registracije. Korisnici najčešće koriste previše očigledne nagoveštaje, čak ima i onih kod kojih su nagoveštaji potpuno identični sa lozinkom. Prilikom kreiranja nagoveštaja korisnici misle da će nagoveštaj biti vidljiv samo za njih. I ovde se može koristiti brute force metoda za nagađanje.
- Neke aplikacije dozvoljavaju korisnicima da resetuju lozinke odmah nakon što uspešno odgovore na zaštitno pitanje a da pritom ne pošalju obaveštenje na e-mail korisnika. Ovo znači da napad korisničkog računa neće biti primećen sve dok pravi vlasnik ne pokuša da se prijavi na aplikaciju. U nekim slučajevima žrtva i ne primeti da je napadač koristio aplikaciju u njeno ime. Napadač koji je već jednom ušao u tuđ korisnički račun može čuvati podatke o obnovi lozinke (nagoveštaj lozinke, odgovor na sigurnosno pitanje) i s vremena na vreme može koristiti aplikaciju.

"Upamti me" funkcionalnost

Neke aplikacije imaju implementirane "*Upamti me*" funkcije kao pogodnost koju pružaju korisnicima. Na ovaj način, korisnici ne unose korisničko ime i lozinke svaki put kada koriste aplikaciju sa specifičnog kompjutera. Ove funkcije su često veoma nesigurne i ostavljaju korisnike izložene napadu kako lokalnom, tako i od strane drugih kompjutera.

Neke "Upamti me" funkcije su impelementirane koristeći jednostavne stalne *coookie*, kao što je npr. *Korisnik=Muzafer.* Kada je ovaj cookie dostavljen stranici aplikacije, aplikacija veruje *cookie*-u i autentifikuje korisnika, zatim kreira sesiju za korisnika, zaobilažeći prijavu. Napadač može koristiti listu najčešćih korisničkih imena da bi omogućio sebi pristup aplikaciji bez ikakve autentifikacije. U ovu svrhu se može koristiti bilo koji alat preko koga možemo menjati sadržaj cookie-a (kao npr.

Cookie Manager ekstenzija za Mozilla Firefox web browser). Neke "Upamti me" funkcije ne čuvaju korisničko ime u cookie-u već identifikator sesije, kao npr. *Korisnik=8421*. Kada se identifikator prosledi stranici, stranica pretražuje korisnika sa tim identifikatorom i kreira sesiju.

NAPAD NA KONTROLE PRISTUPA

U osnovi mehanizma zaštite, kontrole pristupa u aplikacijama zauzimaju značajno mesto. Kontrola pristupa je često zasnovana na upravljanju autentifikacijom i sesijama. Glavni razlog zbog koga aplikacije rade poslove kontrole pristupa je potreba za odlukom da li korisniku omogućiti ili zabraniti pristup određenim resursima.

Kontrola pristupa predstavlja kritičan mehanizam odbrane zato što je upravo on odgovoran za donošenje ovih odluka. Kada postoji propust u kontroli pristupa, napadač može biti u stanju da naruši rad cele aplikacije, može da sebi pridobije administratorske prvilegije i da dođe do poverljivih podataka. Kontrola pristupa se može podeliti u tri kategorije [1]:

1. Vertikalna kontrola pristupa dozvoljava različitim tipovima korisnika da koriste neke delove aplikacije. U najjednostavnijem slučaju, vertikalna kontrola pristupa pravi podelu između dva tipa korisnika: obični korisnici i administratori. U nekim kompleksnijim sistemima vertikalna kontrola pristupa može u sebi sadržati mehanizam po kom svakom korisniku dozvoljava pristup tačno određenim funkcijama i resursima.
2. Horizontalna kontrola pristupa omogućava korisnicima da pristupe određenom podskupu šireg opsega resursa istog tipa. Na primer, web mail nam može dozvoliti da čitamo naše mail poruke, ali ne i mailove koji pripadaju drugim korisnicima, ili banka nam može dozvoliti da vršimo prenos novca i sredstava upravljajući samo našim računom, ali ne i tuđim računima.
3. Kontrola pristupa zasnovana na kontekstu ograničava korisniku pristup u zavisnosti od trenutnog stanja aplikacije. Na primer, ako korisnik izvršava iste akcije više puta (pristupa nekom resursu 20 puta), aplikacija će brojati korisnikove zahteve za pristupom, i nakon što se limit ispuni, aplikacija će zabraniti korisniku sledeći novi pristup tom resursu. U nekim slučajevima, sa nedostatkom kontrole pristupa, napadač može doći do osetljivih funkcija samo preko URL adrese. Na primer, kod nekih aplikacija bilo koji korisnik koji poseti specifičnu URL adresu može doći do administratorskih privilegija: www.sajt.com/administrator/

U ovom primeru www.sajt.com/1245874/admin545.php, aplikacija prikazuje link do administratorskog panela samo jednoj vrsti korisnika: administratorima. Drugim korisnicima ovaj link nije prikazan. Po nekad nagađanje URL adrese koja dozvoljava pristup administratorskim funkcijama može biti otežano i tu se može iskoristiti određeno šifrovanje lokacije. Ovde je pristup stranici osiguran pretpostavkom da napadač neće znati i neće biti u stanju da otkrije tačnu URL adresu

stranice. Napadač u nekim slučajevima može doći do lokacije stranice gledajući u client-side kod. Mnoge aplikacije koriste JavaScript da bi omogućile dinamičan korisnički interfejs svojim korisnicima. Napadač jednostavno može izmeniti vrednost "admin" varijable u "true", zatim pozvati funkciju "proveri" i na taj način doći do administratorske stranice, unosom sledećeg koda u adresnoj liniji browsera: javascript:admin=true; proveri();

Naravno postoji i drugi način. U kodu se već vidi lokacija admin stranice, napadač može kopirati lokaciju u svom browseru i na taj način doći do željene lokacije. Ukoliko se u funkciji nalaze naredbe za dodeljivanje *cookie* ili sesije koji će korisniku omogućiti pristup admin stranici, u tom slučaju je prihvatljivije koristiti prvi način sa pozivom funkcije "*proveri*". Neke aplikacije koriste nesigurne kontrole pristupa u kojima se odluke o pristupu donose na osnovu podnešenih parametara od strane klijenta. Drugim rečima, takve aplikacije su u kontroli napadača. Postoje sledeće tri kontrole pristupa:

- Kontrola pristupa bazirana na parametrima
- Referer bazirana kontrola pristupa
- Kontrola pristupa bazirana na lokaciji

Kontrola pristupa bazirana na parametrima

Aplikacija određuje nivo korisnikovog statusa od vremena kada se korisnik prijavi, i od tog trenutka pa na dalje prenosi tu informaciju preko klijenta u skrivenim poljima ili preko parametra u URL adresi. Uvek kada se neki zahtev obradi, aplikacija čita parametar, i na osnovu njega određuje da li će korisniku omogućiti ili uskratiti pristup resursima. Na primer, administrator koji je trenutno ulogovan sa svojim administratorskim privilegijama može videti sledeći URL: www.sajt.com/login/index.php?administrator=true

Ovaj URL se prikazuje samo administratoru, ali ne i drugim korisnicima. Drugi korisnici mogu imati drugačiji parametar. Svaki korisnik koji zna šta "administrator" parametar predstavlja može jednostavno da doda parametar u svojoj URL adresi i na taj način da dođe do administratorskih funkcija. Napad na ovaj tip kontrole pristupa je često vrlo teško otkriti bez korišćenja aplikacije u visoko privilegovanom modu da bi se identifikovalo odakle je određeni zahtev pristigao.

Referer bazirana kontrola pristupa

Neke aplikacije koriste *Referer* HTTP zaglavlje kao osnovu donošenja odluka o kontroli pristupa. Na primer, aplikacija može striktno kontrolisati pristup do nekih delova administratorskog menija na osnovu privilegija korisnika. Ali ako korisnik pošalje zahtev za pojedinačnu administracijsku funkciju, aplikacija može jednostavno proveravati samo da li je zahtev poslat iz administratorske stranice. Na taj način aplikacija može pretpostaviti da korisnik ima određene privilegije i dopušta mu korišćenje i ostalih funkcija koje mu nisu dozvoljene. Ovaj model je generalno

veoma nesiguran, obzirom da je *Referer* zaglavlje pod kontrolom korisnika, i može mu se menjati vrednost.

Kontrola pristupa bazirana na lokaciji

Mnoge aplikacije imaju regulativu ili poslovnu politiku po kojoj uskraćuju pristup resursima u zavisnosti od geografske lokacije korisnika. Ograničenja se ne odnose na finansijske sektore, ali uključuju vesti, pristup multimedijalnim sadržajima itd. U ovim situacijama, kompanija mora da uvrsti nekoliko metoda da bi locirala korisnika, najčešće se koristi otkrivanje geo-lokacije preko IP adrese korisnika. Kontrolu pristupa baziranoj na lokaciji je relativno lako zaobići od strane napadača. Neki od načina zaobilaženja su:

- Korišćenje proxy servera koji se nalaze na specifičnoj lokaciji koju aplikacija dozvoljava,
- Korišćenje VPN koji odgovara dozvoljenoj lokaciji,
- Korišćenje mobilnog telefona koji podržava roaming,
- Direktnom manipulacijom client-side mehanizma geo-lokacije.

Na Internetu postoji veliki broj web sajtova koji održavaju liste besplatnih proxy adresa koji se mogu koristiti. Proxy serveri omogućuju klijentima korišćenje njihove IP adrese kako bi se na Internetu predstavljali anonimno. Jedno od osnovnih stvari koje napadač uradi pre nego što krene u misiju napadanja putem Interneta je da pronađe odgovarajući proxy server na koji će se povezati, pa tek onda krenuti u napad anonimno, da ne bi otkrio svoj identitet.

SQL Injection

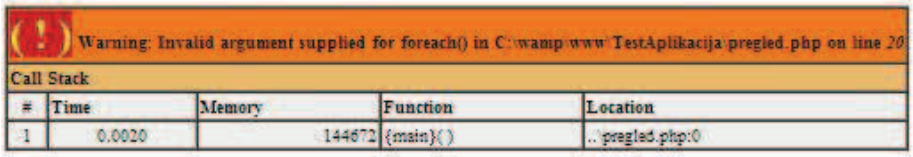
Većina web aplikacija koriste bazu podataka da bi čuvale različite vrste informacija. Jezik koji se koristi za pristup podacima u bazi podataka je Structured Query Language (SQL). Web aplikacije obično konstruišu izraze u vidu SQL izraza preko kojih dolaze do korisničkih podataka. Ako se ovo odradi na nesiguran način, tada aplikacija postaje ranjiva na SQL injection napad.

U nekim ozbiljnijim napadima, SQL injection može da omogući napadaču da čita i modifikuje sve podatke koji se čuvaju u bazi, čak može da omogući napadaču potpunu kontrolu nad serverom u kome je implementirana baza podataka. Kako se svest o zaštiti web aplikacija razvila, SQL injection napadi su sve manje upotrebljivi i mnogo je teže upotrebiti ovaj vid napada na aplikacijama današnjice. Pre nekoliko godina SQL injection napadi su bili učestali pošto nije bilo dovoljno dobro razvijenog mehanizma zaštite. Napadač je izvršavao SQL injection napad jednostavnim unosom apostrofa u polje forme. SQL injection napadači koriste da bi došli do neovlašćenog pristupa bazi podataka i prikupili podatke iz nje. Principi koji se koriste u ovim napadima su jednostavni za izvršavanje i upravljanje [5].

SQL injection je napad prilikom kojeg napadač umetanjem SQL koda pokušava iskoristiti ranjivost web aplikacije. Prilikom tog umetanja, menja se

sintaksa originalnog SQL upita i aplikacija umesto originalnog upita izvršava modifikovani upit. Prvo šta će napadač tražiti na aplikaciji je da pronađe način na koji će biti u mogućnosti uneti podatke koji će narušiti SQL upit u aplikaciji. Uzmimo za primer da jedan prosečan korisnik pregleda korisnika čiji ID je 3 na stranici *pregled.php*. Link sa parametrom stranice će izgledati ovako:
http://localhost/TestAplikacija/pregled.php?id=3

SQL upit koji će aplikacija odraditi je sledeći: select * from login where ID_login = 3. Na ovaj način aplikacija prikazuje podatke iz tabele *login* gde je vrednost ID primarnog ključa 3. Ukoliko napadač izmeni link, i doda još ' or 1=1 desno od vrednosti id parametra, aplikacija će javiti sledeću grešku:


Warning: Invalid argument supplied for foreach() in C:\wamp\www\TestAplikacija\pregled.php on line 20

Call Stack

#	Time	Memory	Function	Location
1	0.0020	144672	{main}( )	..\pregled.php:0

Na ovaj način aplikacija će izvršiti sledeći SQL upit:
select * from login where ID_login = 3 ' or 1=1

U ovom slučaju, interpreter dobija podatke na isti način kako je to i pre objašnjeno. On uzima podatke, koji su odvojeni jednostrukim apostrofom, prihvata podatak 3, zatim dolazi do podatka ' or 1=1 i tu nailazi na problem, obzirom da imamo otvoren apostrof, i nigde ga kasnije nismo zatvorili pre "*or*" naredbe. Kada se aplikacija ponaša na ovaj način ona je veoma ranjiva na SQL Injection napad. Napadač može ukloniti apostrof iz linka i tako sebi osigurati izvršavanje SQL upita bez greški, onda može dodati drugi proizvoljan SQL da bi izmenio originalni upit. U gore navedenom primeru, napadač može da izmeni upit i dobije sve korisnike koji se nalaze u tabeli "*login*", tako što unosi sledeći iskaz: OR 1=1--. Ovo za posledicu ima da aplikacija izvrši sledeći upit: select * from login where ID_login = 3 or 1=1--

Na ovaj način se modifikuje originalni WHERE izraz i dodaje drugi zahtev. Baza podataka će proveriti svaki red u tabeli "*login*" i izvaditi sve podatke gde "*ID_login*" kolona ima vrednost "*3*" ili gde je 1 jednako 1. Zato što je 1 uvek jednako 1, baza podataka će izvaditi sve podatke koje se nalaze u tabeli "*login*" i biće prikazani napadaču. U gore navedenom primeru "--" znaci kazuju interpreteru da je reč o komentarima i da treba da zanemari sve što se nalazi sa desne strane od ovih znakova. Na primer, ako aplikacija izvršava upit:
SELECT * FROM login WHERE Username = 'admin' and Password = 'admin', unosom sledećeg iskaza za password: admin' OR 'a' = 'a, aplikacija će izvršiti sledeći upit: SELECT * FROM login WHERE Username = 'muzafer' and Password = 'saracevic' OR 'a' = 'a'.

U nekim situacijama alternativni način da se uspešno izmeni SQL upit bez korišćenja znakova komentara jeste primena metode "*ravnoteže apostrofa*".

Mnoge aplikacije koje imaju implementirane *form* bazirane login funkcije koriste bazu podataka za čuvanje korisničkih podataka i koriste jednostavne SQL

upite za proveru validnosti unetih podataka prilikom prijave. Tipičan primer ovog SQL upita, koji koristi naš test primer u aplikaciji je: select * from login where Username = 'muzafer' and Password= 'saracevic'

Ovaj upit zahteva od baze podataka da proveri svaki red u tabeli *login* gde kolona *Username* ima vrednost *muzafer* i kolona *Passowrd* ima vrednost *saracevic*. Ukoliko se red sa ovim podacima u tabeli pronađe, prijava korisnika je uspešna i aplikacija kreira sesiju za korisnika. Kao u gore navedenom primeru, napadač može da umetne u polje namenjeno za unos korisničkog imena, ili u password polje svoj tekst tako da izmeni upit koji je kreiran od strane programera i obori logiku aplikacije. Na primer, ako napadač zna da je korisničko ime administratora "*admin*", on se može uspešno prijaviti na aplikaciju koristeći bilo koju lozinku umetanjem sledećeg teksta u polje namenjeno unosu korisničkog imena u test aplikaciji. Na ovaj način aplikacija izvršava sledeći SQL upit:
select * from login WHERE Username = 'admin' -- ' AND Password = 'jdgdf'. Kada odstranimo komentar koji aplikacija neće izvršiti, konačni SQL upit izgleda ovako:
SELECT * FROM login WHERE Username = 'admin'

U velikom broju aplikacija, prvi korisnički račun je upravo administratorski, zato što je ovaj korisnički račun u velikom broju slučajeva prvi kreiran u bazi i to manuelno, pa se tek kasnije preko njega kreiraju ostali računi. Ukoliko upit vraća detalje više korisnika, mnoge aplikacije će jednostavno obraditi samo prvog korisnika na koji naiđe. Napadač može iskoristiti ovo ponašanje da bi se prijavio sa podacima prvog korisnika u bazi podataka, tako što za korisničko ime unosi sledeći iskaz: ' OR 1=1 --. Na ovaj način aplikacija obrađuje sledeći upit:

SELECT * FROM login WHERE Username = ' ' OR 1=1 --' AND Password = 'hfds'

Automatizovani alat za proveru SQL ranjivosti

Postoji veliki broj alata za za proveru SQL ranjivosti. Neki od njih se mogu naći u BackTrack operativnom sistemu zasnovanom na Linux-u, kao što je *Sqlmap*. Predstavićemo alat koji poseduje GUI i kreiran je za Windows platformu. To je alat pod imenom "*HAVIJ*". Havij je automatizovani SQL injection alat koji pomaže u testiranju i otklanjanju SQL injection ranjivosti web aplikacija. Koristeći ovaj alat, korisnik može da otkrije bazu podataka koju koristi aplikacija, tabele u bazi, podatke koji se nalaze u tabelama, dodaje, briše i menja sadržaj podataka u tabelama, izvršava SQL upite i izvršava *shell* komande. Osim toga, ovaj alat ima implementiranu pretragu admin stranice, koja se bazira na nagađanju naziva (koristi listu najviše korišćenih imena za admin stranice) i čitanju statusnog broja koji vraća server.

HAVIJ podržava GET i POST metode. POST metoda je malo zahtevnija, pošto napadač mora da gleda u izvorni kod stranice, pokupi imena polja za unos podataka, kao i stranicu kojoj se šalju podaci POST metodom.

Curenje informacija

Ovaj tip ranjivosti, kao što i samo ime kaže, odaje napadaču informacije koje može koristiti za zloupotrebu drugih tipova ranjivosti. Osetljive informacije se obično otkrivaju kroz zaboravljene komentare unutar programskog koda ili HTML stranica, zbog omogućene opcije listanja sadržaja foldera i neispravnog rukovanja greškama.

Često se otkrivanje osetljivih informacija može odraditi čitanjem samog HTTP zaglavlja u odgovoru servera. Unutar HTTP zaglavlja obično se odaju informacije kao što su: tačna vrsta i verzija web servera i vrsta programskog jezika u kojem je napisana web aplikacija. Otkrivanje osetljivih informacija preko HTTP zaglavlja nije direktno vezano za ranjivost web aplikacija, ali i te informacije mogu pomoći napadaču pri izvođenju napada na web aplikaciju.

Osetljive informacije se nekad mogu naći i u zaboravljenim komentarima unutar programskog koda ili koda HTML stranica. Takvi komentari mogu otkriti različite vrste informacija. Obično se u takvim komentarima nalaze opisi različitih delova programskog koda, opis načina funkcionisanja neke programske skripte, mrežna imena servera itd. U nekim slučajevima, unutar komentara u programskom kodu mogu se nalaziti i korisnička imena i lozinke koji su se koristili tokom razvoja aplikacije. Ta korisnička imena i lozinke mogu biti važeći i nakon što je završena razvojna faza same aplikacije, pa ih napadač jednostavno može iskoristiti za prijavu na aplikaciju. Sve ove informacije mogu napadaču biti od koristi, pa je zbog toga potrebno izbrisati suvišne i nepotrebne komentare iz programskog koda.

Web serveri obično imaju opciju listanja sadržaja foldera ukoliko se u folderu ne nalazi početna stranica. Ranjivošću se smatra ako je listanje sadržaja nekog foldera uključeno, a da se to nije htelo. U tom slučaju listanjem sadržaja foldera, mogu se otkriti različite vrste datoteka koje možda ne bi trebale biti prikazane. Potrebno je napomenuti da napadač nezavisno o tome da li je uključena opcija listanja sadržaja foldera, može pristupiti tim datotekama ako nije implementirana kontrola pristupa.

Web aplikacija može odavati osetljive informacije ako neispravno rukuje greškama [2]. Postoje dva tipa grešaka koje mogu nastati tokom rada web aplikacije, a to su: očekivane i neočekivane greške. Očekivane greške nastaju tokom normalnog korišćenja aplikacije. Jedna očekivana greška može biti neuspešna prijava korisnika na aplikaciju. Stranica sa greškom koja opisuje neuspelu prijavu morala bi biti ista nezavisno o tome da li je upisana samo pogrešna korisnička lozinka ili je upisano pogrešno i korisničko ime i lozinka. Aplikacija sa greškom otkriva da se unešeno korisničko ime nalazi u bazi, ali da je lozinka koju je korisnik uneo pogrešna. U tom slučaju nepotrebno se odaju informacije o postojećim korisničkim imenima u bazi podataka. Ovakvo ponašanje aplikacije napadač može iskoristiti kako bi skupio listu ispravnih korisničkih imena, koju kasnije može iskoristiti u napadu na web aplikaciju.

Kako web aplikacija tokom napada dobija veliki broj neočekivanih HTTP zahteva, za očekivati je da će barem jedan izazvati neželjeno dejstvo i onemogućiti normalan rad aplikacije. Ako ove greške nisu propisno sanirane mogu odavati veliki broj osetljivih informacija koje napadaču mogu biti od velike koristi i olakšati mu zloupotrebu drugih ranjivosti. Ove greške obično odaju osetljive informacije o unutrašnjoj strukturi web aplikacije, pa se zbog toga najviše koriste kako bi se prikupile informacije potrebne za iskorišćenje ranjivosti. Uz poruke o greškama koje izdaje sam server mogu se otkriti i mnogi drugi napadaču korisni podaci, kao što su: ime baze, ime korisnika baze, verzija servera, imena tabela itd.

Automatizovanim alatima je moguće otkriti curenje informacija. To su alati koji pokušavaju izazvati neočekivano ponašanje aplikacije kako bi izazvali grešku, slanjem posebno pripremljenih HTTP zahteva itd. Ranjivost se otkriva analizom dobijenog odgovora. U dobijenom odgovoru se obično traže ključne reči koje se obično pojavljuju u većini opisa nastalih greški, kao što su: *runtime error, exception, illegal, invalid, fail, stack, ODBC* itd.

ZAŠTITA KONTROLE PRISTUPA

Kontrola pristupa je jedan od najlakših delova za razumevanje u web aplikaciji. Kao mere zaštite, prvo se moraju izbeći neke određene zamke koje često nastaju iz neznanja o osnovama mehanizma kontrole pristupa i iz pogrešnih pretpostavki o tome kakve tipove zahteva će korisnici slati aplikaciji. Neke od osnovnih stvari na koje treba obratiti pažnju su [7]:

- Treba poći od pretpostavke da korisnik poznaje sve URL adrese aplikacije i treba osigurati da kontrola pristupa aplikacije može da onemogući neovlašćen pristup.
- Ne oslanjati se na neznanje korisnika da izmene sadržaj URL adrese ili identifikatora koji se koriste da bi precizirali resurse aplikacije, kao što su brojevi računa i identifikacija dokumenata.
- Ne treba verovati nijednom parametru od strane korisnika (kao što je bilo navedeno u primeru *administrator = true* i sl.)
- Ne treba pretpostavljati da će korisnici pristupati stranicama aplikacije na podrazumevani način.
- Ne treba pretpostavljati da korisnici ne mogu doći do stranice koja im se ne nudi u samom korisničkom interfejsu browsera.
- Ne verovati da korisnik neće vršiti izmene podataka koji se šalju od strane klijenta. Ako podaci prođu validaciju od strane client-side provere, ne treba verovati pristiglim podacima pre nego što ponovo ne prođu proces validacije na serveru.

Zaštita autentifikacije

Implementacija zaštite autentifikacije uključuje korišćenje nekoliko ključnih zaštitnih faktora, što u mnogim slučajevima ima za posledicu degradiranje drugih faktora kao što su funkcionalnost i upotrebljivost. U mnogim slučajevima "veća sigurnost" može imati kontra efekat. Na primer, zahtevanje od korisnika da koriste duge lozinke i da menjaju iste s vremena na vreme može napraviti situaciju da korisnici zaboravljaju svoje lozinke.

Neki od načina odbrane autentifikacije od napada su:

A) Korišćenje jakih akreditiva:

- Korišćenjem filtera za proveru kvaliteta lozinke. Ovde je moguće koristiti zakone o minimalnoj dužini lozinke i koje znakove korisnik može upotrebiti za lozinku, kao što su: brojevi, karakteri, slova, korišćenje malih i velikih slova, izbegavanje korišćenja reči koje napadač može lako pogoditi, korišćenje imena ili nekih uobičajenih lozinki, ne dozvoljavanje da se za lozinku koristi korisničko ime i sl. Od korisnika bi trebalo da se traži kreiranje jakih lozinki - duge lozinke, širok asortiman karaktera i sl.

B) Tajno rukovanje akreditivima:

- Svi akreditivi bi trebalo biti kreirani, čuvani i isporučivani na način da budu nedostupni drugim neovlašćenim pristupima.
- Sve client-server komunikacije bi trebalo biti čuvane koristeći dobru kriptografsku tehnologiju, kao sto je SSL. Aplikacije bi trebale koristiti HTTPS protokol umesto HTTP. Jedino POST zahteve bi trebalo koristiti za prenos akreditiva do servera. Akreditive nikad ne bi trebalo prenositi putem URL parametara ili cookie-a.
- Sve server-side aplikacione komponente bi trebalo da čuvaju akreditive na način da ne dozvoljavaju pristup njihovom sadržaju neovlašćenim korisnicima.
- Client-side "Upamti me" funkcije bi trebalo generalno da čuvaju samo informacije koje nisu tajne, kao što su korisnička imena. Korisnicima bi trebao biti dozvoljen sistem dojave u slučaju da napadač fizički dođe do kompjutera, ili koristi kompjuter daljinski.
- Promena lozinke bi trebalo da bude omogućena korisnicima. Korisnici bi trebali menjati svoje lozinke periodično.

Zaštita od Brute-Force napada

Neke aplikacije jednostavno onesposobe korisnički račun nakon relativno malog broja pokušaja logovanja (npr. nakon tri pokušaja). One takođe zahtevaju od korisnika profila da prođe kroz veći broj provera svoje identifikacije da bi povratio

svoj korisnički račun, kao što je pozivanje korisničke podrške putem telefona i davanje odgovora na bezbednosna pitanja.

Loša strana ovog pristupa je to što se na ovaj način omogućuje napadaču da odbija servise legitimnom korisniku tako što iznova onemogućuje pristup korisničkom računu, uvek kada korisnik vrati svoj račun. Prihvatljiviji način zaštite je onemogućiti pristup korisničkom računu na neki kratak period (kao npr. 30 minuta) ukoliko korisnik nije u mogućnosti da se uspešno prijavi na sistem nakon određenog broja neuspelih prijava (npr. tri puta). Ovo usporava bilo koji napad koji se zasniva na nagađanju informacija.

Da bi se izbegao Brute-Force napad trebalo bi koristiti odgovarajuće sisteme provere korišćenja automatizovanih alata [10]. Veoma je bitno korišćenje nepredvidivih korisničkih imena i sprečavanje njihovog nagađanja koristeći automatizovane alate.

Zaštita aplikacije od SQL injection napada

Glavni razlog SQL injecton ranjivosti je nedovoljna validacija ulaznih podataka. U sprečavanju ovog napada veoma je bitno filtriranje i validacija podataka koje korisnik unosi. Osim ispitivanja veličine, tipa i sadržaja ulaznih podataka potrebno je koristiti i pripremljene procedure. Ukoliko je aplikacija odrađena u PHP jeziku, postoje interfejsi baza podataka koji imaju funkcionalnost "prepare statement", to su *MySQLi* i *PDO* [11]. Na ovaj način aplikacija unapred zna strukturu SQL upita i očekuje prijem samo podataka, a ne SQL naredbi.

MySQLi je razvijen samo za MySQL baze podataka i samo sa MySQL može raditi. Postoje dve varijante ovog drajvera, proceduralni i objektno orijentisan.

PDO je objektno orjentisan, i u mogućnosti je da radi sa 12 različitih baza podataka. PDO ima ugrađenu proveru za apostrofe, tako da iste ne treba unositi u SQL upitu. Sa druge strane PDO je sporiji u izvršavanju naredbi od MySQLi-a.

ZAKLJUČAK

Nakon svih ovih analiza i razmatranja, možemo zaključiti da su najranjiviji delovi aplikacije oni gde aplikacija prihvata ulazne podatke od strane korisnika i tu treba posvetiti najviše pažnje. Kada se otkrije ranjivost, relativno brzo se nađu i odgovarajuće ispravke za aplikaciju. Napadi koristeći se client-side propustima su oni koji se najviše koriste i razvijaju u poslednjih nekoliko godina. Napadi na baze podataka (SQL Injection) polako nestaju, s obzirom da se prave veoma jake zaštite koje je nemoguće zaobići. Korišćenjem *PDO prepare PHP* naredbe, aplikaciji je omogućeno da unapred realizuje strukturu SQL upita pre nego prihvati ulazne parametre korisnika, tako da kasnije napadaču neće biti omogućena izmena i zloupotreba upita.

Postoji veliki broj programera koji nemaju dovoljno znanja o zaštiti i sigurnosti aplikacija koje razvijaju. Međutim, postoje i oni koji imaju dovoljno znanja

i u mogućnosti su da zaštite aplikaciju na najbolji mogući način ali su jednostavno lenji misleći da neće njegova aplikacija pasti u ruke napadaču. Pri razvoju aplikacije, programer mora razmišljati o najgorem mogućem događaju koji se može odigrati prilikom napada i implementirati zaštitu za sve vrste napada na najbolji mogući način.

LITERATURA

[1] Mike Shema, *Hacking Web Apps, Edition: 1,* Wiley publication, ISBN-13: 978-1597499514, ISBN-10: 159749951X, 2010.
[2] Rich Cannings, Himanshu Dwivedi, Zane lackey, *Hacking Exposed Web 2.0,* Edition: 1, ISBN-13: 978-0071494618, ISBN-10: 0071494618 December 17, 2007.
[3] Joel Scambraz, Vincent Liu, Caleb Sima, *Hacking Exposed Web Application 3*, ISBN: 9780071740647, Division: Professional, Pub Date: 2010.
[4] Ćamil Sukić, *Sigurnost računarskih sistema,* udžbenik, ISBN 978-86-84389-37-6, COBISS.SR-ID 189645068, Univerzitet u Novom Pazaru, 2012.
[5] Justing Clark, *SQL Injection Attack and Defense,* Wiley publication, ISBN: 978-1-118-36218-1, 552 pages, December 2012.
[6] Dafydd Stuttard, Marcus Pinto, *The Web Application Hacker's Handbook,* Edition: 2nd, ISBN-13: 978-1118026472, ISBN-10: 1118026470, 2012.
[7] Ryan C. Barnett, *Web Application Defender's Cookbook: Battling Hackers and Protecting Users*, Edition: 1st, ISBN-13: 978-1118362181, 2012.

INTERNET IZVORI:

[8] https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
[9] http://www.salebab.net/cross-site-scripting-xss-sigurnost-php-aplikacija-4-deo/
[10] http://www.infosecpro.com/applicationsecurity/a52.htm
[11] http://php.net/mysql_real_escape_string

SECURITY MANAGEMENT IN CYBER SPACE AND PROTECTION MECHANISMS OF WEB APPLICATIONS

PhD Muzafer Saracevic
MSc Muhedin Hadzic

This paper examines the basic mechanisms of security and security management in the Cyber Space. The basic problem of protecting web applications is reflected in the fact that all data entries by users are uncertain and should not be trusted. This approach has developed a number of security mechanisms that applications use to defend themselves against attackers. The most serious attacks on web applications are attacks that reveal sensitive data or allow unrestricted access to the system and resources.

For many companies, any attack that results in system downtime is a critical event. When it comes to protection and security risks, it must be known that this is a continuous struggle between those who detect flaws and attacks applications and those who make defense mechanisms. What is nowadays actual as an attack method, in the near future can be modified or replaced by some other method.

In this paper, emphasis is placed on attacks that can be used to exhaust the resources of the server on which the application is located, resulting in the inability to access the application by the user.

Keywords: Cyber Security, Protection Mechanisms, Attacks on the Internet, Vulnerability of Web Applications.

Article history:
Received: 11. 1. 2017.
Accepted: 4. 9. 2017.