Godina I
Broj I
Sveska
1/2013

TRENDOVI U POSLOVANJU

BEZBEDNOST VEB AJAX APLIKACIJA POSLOVNIH INFORMACIONIH SISTEMA

Dr Bojan Milosavljević
Visoka hemijsko tehnološka škola strukovnih studija - Kruševac
e-mail: bmilosavljevic@gmail.com

Sažetak

Primena veb interfejsa za poslovne aplikacije danas je de facto standard koji se nameće zbog različitih prednosti, kao što su: dostupnost centralizovanog poslovnog informacionog sistema, koji integriše sve aktivnosti preduzeća u lancu vrednosti, sa bilo koje lokacije na internetu; mogućnosti korišćenja aplikacije kao usluge, u "računarskom oblaku", čime se eliminišu potrebe institucije za specijalizovanim IT osobljem za instalaciju, održavanje i administriranje hardverske, softverske i mrežne infrastrukture; veb aplikacija je najčešće i osnova programiranja aplikacija za pristup informacionom sistemu sa mobilnih uređaja, čime se obezbeđuje neposredna, stalna i potpuna povezanost informacionog sistema i svih izvršilaca poslovnih procedura u preduzeću.

Međutim, sve privlačne prednosti primene veb interfejsa za poslovne informacione sisteme mogu postati zamka za nedovoljno oprezne programere i administratore, jer veb nije kreiran kao podrazumevano bezbedno okruženje. To može izazvati nesagledive posledice ako se poverljivi poslovni podaci izlože neovlašćenom korišćenju i izmeni. Posebno se polje zlonamernog dejstva povećalo primenom tehnologije Ajax za naknadno asinhrono obraćanje klijenta veb serveru sa već učitane stranice, pomoću koje se postiže interaktivnost veb aplikacija približno kao desktop aplikacija. U ovom radu se analizira pitanje bezbednosti veb Ajax aplikacija, sistematizuju se vrste, mesta i postupci zlonamernih dejstava i predlažu načini sprečavanja ugrožavanja bezbednosti veb aplikacija.

Abstract

Business applications having web interface is de facto standard nowadays, imposed by various advantages, such as: availability of a centralized business information system integrating all enterprise value-chain activities from any location on the Internet; possibilities for utilization of software-as-a-service model in a cloud, thus eliminating institution's demands for specialized IT stuff involved in installation, maintenance and administration of hardware, software and network infrastructure; web applications represent a common programming framework for mobile applications providing in-time, persistent and complete connection of business procedures in the enterprise with the information system.

These attractive advantages of web business information systems may become entrapments for not focused developers and administrators, because Web has not been created with security in mind. Serious consequences could arise if confidential business data would be exposed to unauthorized usage and modification. The attack surface has become even larger since Ajax technology emerged for sending

asynchronous client requests to web server from already-loaded webpage, thus achieving comparable interactivity to that of desktop applications. Ajax web application security issues are analyzed in this paper. Various attacks are clasified according to their place in application execution cycle and malware activities being used, and methods are proposed to prevent those security threats successfully.

Ključne reči: *bezbednost veb aplikacije, Ajax, poslovni informacioni sistemi*

Keywords: *web application security, Ajax, business information systems*

1. Uvod

Primena veb interfejsa za aplikacije poslovnih informacionih sistema danas je *de facto* standard. Takav smer razvoja aplikacija omogućava i diktira tehnologija Ajax (*Asynchronous JAvascript And Xml*). Ajax zapravo predstavlja skup tehnologija koje se zasnivaju na primeni jednog ili više tehnoloških rešenja za ponovno obraćanje veb serveru bez ponovnog učitavanja veb stranice u celini i ostvarivanje programskih akcija i/ili preuzimanje sadržaja sa servera koji menjaju pojedine delove već učitane veb stranice. Time se postiže interaktivnost veb stranica koja se može porediti sa interaktivnošću stonih (desktop) aplikacija. Ovo tehnološko dostignuće pokrenulo je kreiranje i primene različitih, gotovo uvek besplatnih programskih biblioteka koje olakšavaju razvoj aplikacija na vebu. Mogućnosti za brz i jeftin razvoj dovele su do masovne pojave veb aplikacija. To je jedan od glavnih preduslova *Web 2.0*, aktuelne paradigme unapređenog korišćenja veb servisa interneta, po kojoj su korisnici, a ne programeri, glavni kreatori, propagatori i merilo vrednosti dostupnih sadržaja i aktivnosti za društveno umrežavanje. Sa tim je u vezi prebacivanje na veb velikog broja aktivnosti i aplikacija koje se obavljaju na računarima, uključujući i poslovanje. Ovako masovna primena pojedinih veb aplikacija omogućila je i razvoj modela korišćenja veb aplikacije kao usluge (*Software as a Service* - *SaaS*), u tzv. "računarskom oblaku" (*cloud computing*). Time se eliminiše potreba za specijalizovanim IT osobljem u preduzeću koje bi bilo zaduženo za instalaciju, održavanje i administriranje hardverske, softverske i mrežne infrastrukture.

Jedna ista veb aplikacija najčešće predstavlja i osnovu za programiranje aplikacija za mobilne uređaje različitih proizvođača i hardverskih arhitektura, na platformama kao što su *PhoneGap* ili *Titanium Studio*. Mobilni uređaji mogu biti korišćeni za pristup informacionom sistemu na svakom mestu, čime se obezbeđuje neposredna, stalna i potpuna povezanost informacionog sistema i svih izvršilaca poslovnih procedura u preduzeću.

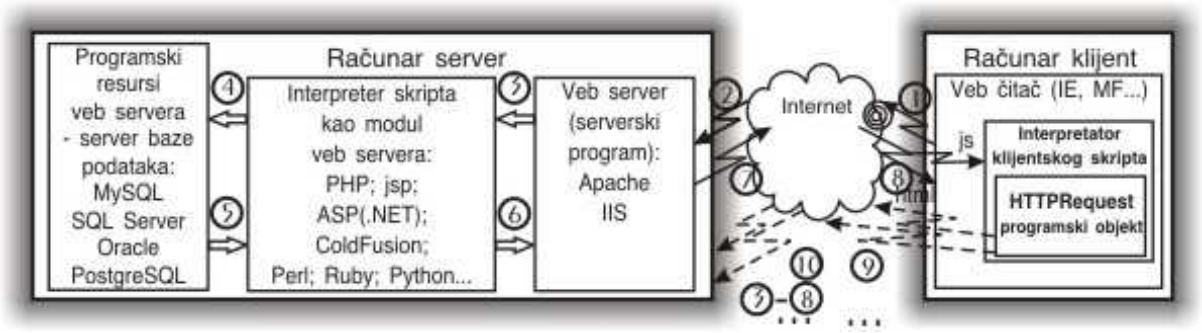
Međutim, sve privlačne prednosti primene veb interfejsa za poslovne informacione sisteme mogu postati zamka za nedovoljno oprezne programere i menadžere, jer veb nije kreiran kao podrazumevano bezbedno okruženje. To može izazvati nesagledive posledice ako se poverljivi poslovni podaci izlože neovlašćenom korišćenju i izmeni. Posebno se polje zlonamernog dejstva povećalo primenom tehnologije *Ajax*, između ostalog i zato što se poslovna logika prebacuje jednim delom i na klijentski skript koji je vidljiv u svakom veb čitaču, a takođe i zato što se otvaraju nove mogućnosti za napade umetanjem zlonamernog klijentskog skripta, jer se pri korišćenju klijentskog *Javascript*-a može zaobići *same-domain policy* - pravilo izvršavanja sadržaja samo sa istog domena na kom je i glavna veb stranica.

U ovom radu se analizira pitanje bezbednosti veb Ajax aplikacija, sistematizuju se vrste, mesta i postupci zlonamernih dejstava i predlažu načini sprečavanja ugrožavanja bezbednosti veb Ajax aplikacija.

2. Tehnologija Ajax i RIA (bogate internet aplikacije)

Na slici 1 ilustrovana je primena tehnologije Ajax za naknadne izmene pojedinih elemenata veb stranice posle njenog učitavanja. Komunikacija između klijentskog programa (veb čitača) i veb servera za sadržaj na iza-

branoj veb adresi (URL - *Uniform Resource Locator*) odvija se na internetu prema protokolu aplikacijskog sloja HTTP (*HyperText Transfer Protocol*). Sam proces učitavanja stranice prikazan je na ovoj slici od koraka 1 do 8 i sadrži sledeće aktivnosti:



Slika 1: Prikaz faza u razmeni informacija između klijenta i veb servera uz primenu tehnologije Ajax

1. Klijentski program šalje zahtev veb serveru na zadatoj veb adresi

2. Veb server prima zahtev preko interneta i obrađuje ga prema tipu traženog resursa (ekstenziji datoteke)

3. Datoteke ekstenzija .php, .jsp, .asp(x) i dr. "aktivne" stranice sadrže tekst programskog koda koji bi trebalo izvršiti na veb serveru, i veb server ih prosleđuje jednom od svojih programskih modula, odgovarajućem interpretatoru programskog skripta

4. Interpretator skripta se preko svojih programskih ekstenzija obraća resursima veb servera (serveru baza podataka, sistemu datoteka, serveru elektronske pošte i dr.)

5. Resursi na strani veb servera odrađuju akcije koje su im zadate u skriptu na serveru

6. Rezultati dobijeni iz baza podataka i drugih serverskih resursa se pomoću odgovarajućih komandi serverskog skripta ugrađuju u *html* kod, koji opisuje sadržaj veb stranice za prikaz u veb čitaču;

7. Generisani sadržaj veb stranice šalje se klijentu preko interneta

8. Veb čitač (klijent) prihvata sadržaj sa servera; pored .html koda sa uputstvima (u obliku tekstualnih oznaka, tagova) koje elemente stranice i sa kojim sadržajem prikazati, sadrži i podatke o vizuelnom izgledu i razmeštaju ovih elemenata na stranici (u .css datotekama, sa kojima se .html stranica povezuje pomoću oznake `<link>`), kao i tekst programskog *Javascript* koda za izvršavanje u klijentskom programu (veb čitaču) u .js datotekama, sa kojima se .html stranica povezuje u oznaci (tagu) `<script src="URL_js_datoteke"></script>`. Javaskript klijentski kôd definiše logiku rada stranice (ponašanje, *behaviour*), pre svega reakciju na akcije korisnika na interaktivnim elementima veb stranice. Pozivanje na elemente stranice kao na programske objekte omogućava DOM (*Document Object Model*).

Naknadna obraćanja klijenta (korak 9) serveru preko interneta (korak 10) sa stranice pokreću se u klijentskom *Javascript* kodu, najčešće korišćenjem programskog objekta veb čitača `XMLHttpRequest`. Naknadno obraćanje veb serveru sa već učitane stranice iz *Javascript* koda može se pokrenuti i dinamičkim kreiranjem i dodavanjem elemenata `<script>`, `<img>` (slike) ili elemenata `<iframe>` (za učitavanje druge stranice kao dela osnovne stranice), jer imaju atribut `src` (skraćeno od *source*), koji definiše URL izvora, sadržaja za učitavanje. Prilikom ispunjavanja naknadnog zahteva klijenta, veb server ponavlja korake 3 do 8.

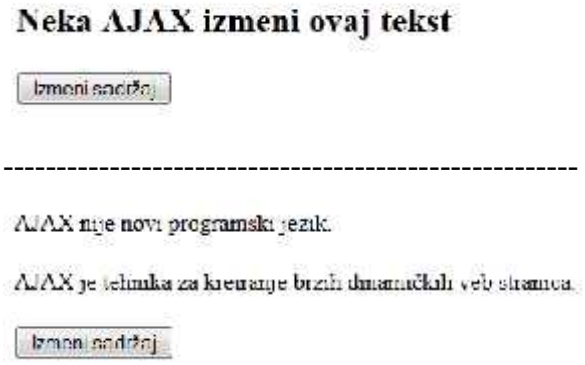
Ažuriranje pojedinih delova ranije učitane veb stranice obavlja se **asinhrono**, ne u osnovnoj niti izvršavanja koda, već kao reakcija na događaje uspeha ili greške realizacije zahteva. Reakcija se definiše u *callback* funkcijama `OnSuccess`, `OnComplete`, `OnError` koje se mogu izvesti iz opšte *callback* funkcije `onreadystatechange` na osnovu ispitivanja parametara `readyState` i `status` za objekat `XMLHTTPRequest`.

2.1. Osnovni primer primene tehnologije Ajax

U .html stranici čiji kôd sledi omogućava se izmena sadržaja elementa veb stranice `<div id= "myDiv">` sadržajem datoteke `ajax_info.txt`, naknadno učitane sa servera (posle učitavanja stranice u celini).

```
<!DOCTYPE html>
<html>
<head>
<title> Osnovni primer primene
tehnologije Ajax </title>
<meta
<script>
var xmlhttp;
function loadXMLDoc(url,cfun)
{
if (window.XMLHttpRequest)
  {/*  kôd  za  IE7+,  Firefox,
Chrome, Opera, Safari */
  xmlhttp=new XMLHttpRequest();
  }
else
  {// kôd za IE6, IE5
  xmlhttp=new
ActiveXObject("Microsoft.XMLHTTP
");
  }
xmlhttp.onreadystatechange=cfun;
xmlhttp.open("GET",url,true);
xmlhttp.send();
}
function myFunction() {
loadXMLDoc("ajax_info.txt",funct
ion()
  {
  if  (xmlhttp.readyState==4  &&
xmlhttp.status==200)
    {
document.getElementById("myDiv")
.innerHTML=xmlhttp.responseText;
    }
  });
}
</script>
</head>
<body>
<div  id="myDiv"><h2>Neka  AJAX
izmeni ovaj tekst</h2></div>
<button  type="button"  onclick=
"myFunction()">Izmeni  sadržaj
</button>
</body>
</html>
```

Izgled stranice pre i posle klika na komandno dugme *Izmeni sadržaj* prikazan je na slici 2.



Slika 2: Veb stranica osnovnog primera primene tehnologije Ajax pre (slika iznad) i posle klika (slika ispod) na komandno dugme Izmeni sadržaj

Programski *Javascript* kôd iz prethodnog primera ne nalazi se u posebnoj .js datoteci, već je umetnut u okviru oznake `<script>`. Preporuka je da se *Javascript* kôd izvršava u okviru oznake zaglavlja `<head>`, jer se onda ne sprečava generisanje veb stranice i učitavanje DOM dok traje izvršavanje ovog kôda.

Iz komentara u okviru primenjene funkcije `loadXMLDoc` može se zaključiti da primenu programskog objekta `XMLHttpRequest` Javaskripta za naknadno obraćanje veb serveru sa već učitane stranice treba prilagoditi različitim verzijama različitih veb čitača za različite platforme, uključujući i mobilne uređaje. Takođe, postoji potreba i za dodatnim olakšicama za programere u korišćenju pojedinih ugrađenih Javaskript funkcionalnosti, kao što je npr. skraćenje imena funkcije `getElementById` na `$`, uz istovremeno svođenje konteksta izvršavanja `document` na opšti kontekst, primenljiv i u *callback* funkciji koja predstavlja reakciju na događaj `domReady` uspostavljanja (dostupnosti) celog objektnog modela elemenata stranice, kada zapravo može otpočeti izvršavanje dodatnog klijentskog kôda. Radi izbora elementa čiji se objekat referencira, ova funkcija `$` može pored `id` imena elementa prihvatiti i CSS selektore elemenata. Drugi primer olakšica je ugrađivanje funkcionalnosti objekta `XMLHttpRequest` u posebnu funkciju koja olakšava njegovu

primenu, uz navođenje pomenutih *callback* funkcija `OnSuccess`, `OnComplete`, `OnError` koje se aktiviraju zavisno od uspešnosti zahteva. Zbog stvaranja sličnih dodatnih pogodnosti i olakšica za programere formirano je mnoštvo programskih biblioteka, čitavih okruženja za brzo programiranje veb aplikacija, što je veoma doprinelo velikoj ekspanziji veb aplikacija i koncepta *Web* 2.0 primene veba. Možda najpoznatija i najpopularnija takva programska biblioteka je *jQuery*, koja ima veoma malu veličinu kôda, a mogu se dodati posebne nadogradnje za grafičke kontrole i za programiranje mobilnih aplikacija.

2.2. Bogate internet aplikacije - RIA (*Rich Internet Application*)

Ajax je jedna od tehnologija koja je omogućila i koncept bogatih internet aplikacija - RIA (*Rich Internet Application*). Ovaj koncept podrazumeva da se celokupna veb aplikacija izvršava samo sa jednim učitavanjem celokupne stranice, a da se nadalje čitav rad aplikacije odvija primenom *Ajax* u *Javascript* kôdu ili dodatku veb čitača (npr. u *ActionScript* kôdu za upravljanje radom dodatka veb čitaču *Adobe Flash*), što onda predstavlja potpunu analogiju sa stonim aplikacijama.

3. Bezbednost veb aplikacija

Na osnovu opšte šeme funkcionisanja veb aplikacija sa primenom tehnologije *Ajax* mogu se izdvojiti odgovarajući nivoi i komponente koje učestvuju u radu veb aplikacije, pa svaka od ovih komponenti može biti i mesto zlonamernih aktivnosti ugrožavanja bezbednosti veb aplikacije. U tako sveobuhvatnom pristupu bezbednosti veb aplikacije treba posebno posvetiti pažnju bezbednosti sledećih komponenti koje učestvuju u radu veb aplikacije:

1. Internet mrežna infrastruktura servera – ruteri, linkovi, adresiranje, zaštitni zid (*firewall*), DNS serveri i dr.;
2. Operativni sistem serverskog računara – provera identiteta korisnika i njihova ovlašćenja za pristup, instalacije servera za veb, DNS, e-poštu, baze podataka, potrebnih serverskih protokola, zaštitnih zidova i antivirusnih programa itd.
3. Server baze podataka – korisnici servera i njihova ovlašćenja za baze podataka, povezivanje sa veb serverom i podrška za pristup bazama iz skript jezika itd.
4. Veb server – konfigurisanje, podešavanje naloga, uspostavljanje sigurne enkriptovane veze SSH (*Secured SHell*) / HTTPS (*Hypertext Transfer Protocol Secure*) i administratorskog interfejsa itd.
5. Mreža provajdera pristupa Internetu za klijenta – ruteri, zaštitni zid, ograničavanje i naplata mrežnog saobraćaja itd.
6. Operativni sistem klijentskog računara – antivirusni programi, zaštitni zid itd.
7. Veb čitač (klijentski program) – dodaci (*plug-in*) i proširenja (ekstenzije), kolačići (*cookies*) itd.
8. Veb aplikacija – provera identiteta i ovlašćenja korisnika aplikacije, obraćanje serveru, veb kontrole korisničkog interfejsa itd.

3.1. Bezbednost serverâ za veb i baze podataka

Mrežne infrastrukture servera i provajdera pristupa internetu za klijenta, kao i operativni sistemi servera i klijenta predmet su angažovanja specijalizovanog osoblja (administratora mreže), i obično ne zavise od programera veb aplikacije i nisu predmet ovog razmatranja. S druge strane, jedan od problema u korišćenju veb aplikacija svakako je činjenica da njihovi administratori i programeri nemaju mogućnost uticaja na klijentsku stranu, odnosno korisnik treba sam da vodi računa o bezbednosti računara i veb čitača sa kog pristupa veb aplikaciji, posebno o dodacima i proširenjima koje koristi. Slično, postoji specijalizovano osoblje i za administraciju servera za veb i baze podataka, ali budući da su ovi serveri komponente koje su u tesnijoj vezi sa samom veb aplikacijom, a takođe i zbog činjenice da je server baze podataka kao centralizovano mesto u kome se čuvaju svi poslovni podaci najpodložniji napadima, biće istaknuti najvažniji praktični saveti za bezbednost ovih servera. Isto tako, zbog važnosti bezbednosti svojih poslovnih poda-

taka, mnoge institucije se radije odlučuju da same hostuju i obezbeđuju svoje veb aplikacije poslovnih informacionih sistema, pa je verovatnije da će se rukovoditi ovim savetima.

Radi bezbednosti servera baza podataka treba se rukovoditi sledećim praktičnim savetima [1]:

1. *Odvojiti na poseban računar server baza podataka od servera za veb*, da bi se izbeglo izlaganje podataka iz baze u slučaju zlonamernog zaposedanja administrativnog naloga veb servera
2. *Uraditi enkripciju čuvanih datoteka*, jer se obično u tekstualnim datotekama serverskog skripta nalaze parametri za pristup bazi podataka
3. *Uraditi enkripciju i datoteka rezervnih kopija*, jer je moguće da će potencijalni napadač baš biti neko od najpoverljivijih službenika preduzeća
4. *Redovno praćenje i ugrađivanje sigurnosnih zakrpa*, što administratori često izbegavaju, zbog bojazni da menjaju već funkcionalnu konfiguraciju
5. *Korišćenje aplikacija sa strane treba svesti na minimum*, jer se ne može biti siguran da li su bez sigurnosnih propusta i da će biti redovno unapređivane i održavane
6. *Ne koristiti deljene servere za veb i baze podataka*, iako je lakše i jeftinije, jer se time veliki deo bezbednosti poslovnih podataka prebacuje na hosting provajdera, ali i zavisi od njegove stručnosti i poštenja
7. *Koristiti bezbednosne kontrole koje omogućava server baze podataka*, kao što su kontrola pristupa, provera identiteta korisnika servera baza podataka, nadgledanje aktivnosti na serveru, enkripcija, kontrole integriteta podataka, rezervne kopije – automatske ili na zahtev, ili sistemi za proveru ranjivosti na napade.

Praktični saveti za bezbednost veb servera obuhvataju sledeće [2,3]:

1. *Instalirati i pokrenuti samo potrebne module veb servera*
2. *Koristiti dozvole datotečnog sistema operativnog sistema servera* – dati odgovarajuća ograničenja korisniku na nivou operativnog sistema kome su pridružene dozvole veb servera
3. *Redovno pratiti i primenjivati sigurnosne zakrpe*, najkasnije jedan ili dva sata po njihovom objavljivanju na sajtu proizvođača softvera veb servera, operativnog sistema i administratorskog panela (pretplatiti se na listu za automatsko obaveštavanje o sigurnosnim unapređenjima)
4. *Pratiti aktivnosti korisnika* koji mogu nenamerno prouzrokovati sigurnosne propuste
5. *Obezbediti izveštaje o aktivnostima*, posebno administratorskim, sa korisničkim imenom, datumom i vremenom aktivnosti
6. *Ograničiti pristup administratorskom delu veb servera* samo za pojedine IP adrese i sa bezbednim protokolom HTTPS
7. *Redovno održavati i testirati upotrebljivost rezervnih kopija*
8. *Veb server izdvojiti izvan zaštitnog zida* da ne bi ugrozio poslovne podatke koji su zaštičeni zidom (konfiguracija "žrtvenog jagnjeta")
9. *Pažljivo konfigurisati veb server* – ograničiti izvršne datoteke samo na određene direktorijume i da se ne može preuzeti njihov izvorni kôd, isključiti sva dodatna automatizovana svojstva koja nisu planirana za korišćenje, za oznaku `<img>` zabraniti preuzimanje slika izvan sopstvenog domena.

3.2. Bezbednost lozinki i provera identiteta

Lozinke je potrebno koristiti za formiranje naloga za korišćenje svih servera, kao i za proveru identiteta korisnika u samoj veb aplikaciji. Zbog njihove sveprisutnosti u radu veb aplikacije potrebno je poštovati preporuke za njihovo formiranje, kao što su:

1. *Koristiti lozinke sa najmanje 8 znakova*
2. *Koristiti lozinke koje pored slova sadrže brojeve, simbole i znake interpunkcije*
3. *Ne koristiti za lozinke reči sa značenjem* (iz rečnika), uključujući i lične informacije, kao što je dan rođenja, jer ih mogu lakše upotrebiti programi za pogađanje lozinke (napad primenom "grube sile")

4. *Ne ponavljati iste nizove znakova*

5. *Ne čuvati lozinke na prenosivim uređajima* (laptop računarima, pametnim telefonima, tabletima), koji se mogu izgubiti

6. *Koristiti generator bezbednih lozinki*, koji automatski kreira lozinke slučajnim izborom znakova po prethodnim pravilima

7. *Koristiti upravljače lozinkama (Password Manager) za bezbedno beleženje lozinki*, programe koji pomažu korisnicima da organizuju svoje lozinke, tako što ih enkriptovane čuvaju u lokalnoj bazi podataka ili datoteci i često ih mogu automatski popuniti u obrascima u drugim aplikacijama

8. *Koristiti dvostruku (two-factor) proveru identiteta* uvek kada je to moguće – pored provere identiteta pomoću korisničkog imena i lozinke (onog što korisnik *zna*), koristiti i PKI (*Private Key Infrastructure*) tokene, pametne kartice i sl. (ono što korisnik *poseduje*) ili biometrijske podatke (ono što korisnik *jeste*) kao dodatnu zaštitu

3.3. Bezbednost klasičnih veb aplikacija

Opšta načela bezbednosti veb aplikacija (bez obzira na primenu tehnologije *Ajax*) su sledeća:

1. *Redovno ažurirati veb aplikaciju* (pretplatiti se na automatska obaveštenja ukoliko se koristi aplikacija treće strane)

2. *Koristiti programe* (kao što je *Nessus* [4]) *za udaljeno testiranje ranjivosti veb aplikacija*

3. *Koristiti zaštitne zidove veb aplikacija*, hardver ili softver (često kao poseban server između veb aplikacije i klijenata, npr. *dotDefender*[5]) koji štiti veb aplikacije na sličan način na koji zaštitni zid štiti mrežu – kontroliše podatke i korisnike koji ulaze u veb aplikaciju i izlaze iz veb aplikacije, ali su "svesni" funkcija aplikacijskog sloja, ispituju svaki HTTP, HTTPS zahtev, uključujući i zahteve za veb uslugama SOAP (*Simple Object Access Protocol*) ili XML-RPC (*eXtensible Markup Language Remote Procedure Call*) i imaju logiku za prepoznavanje i sprečavanje napada na veb aplikacije o kojima će u nastavku biti reči

4. *Obaviti testove poljâ za prosleđivanje datoteka* da ne mogu sadržati programski skript i ograničiti slanje ovih datoteka samo u određene direktorijume na serveru

5. *Koristiti programske biblioteke i okruženja koja imaju proverenu praksu u obezbeđivanju sigurnosti*

6. *Ne pouzdati se samo u teško prepoznatljiva imena, ni u relativne putanje* foldera i datoteka, uvek postavljati osnovni (*base*) direktorijum

7. *Ograničiti pristup administratorskom delu veb aplikacije* samo za pojedine IP adrese i sa bezbednim protokolom HTTPS

8. *Obraditi (proveriti ispravnost) ulazne podatke od korisnika*, tako da se spreče napadi koji ubacuju zlonamerni kôd u ulazne podatke korisnika

9. *Ostaviti datoteke koje sadrže osetljive podatke izvan javno dostupnih foldera*, ili im ograničiti pristup u datotečnom sistemu

10. *Ne verovati polju Referer HTTP zahteva*, jer se može prepraviti

11. *Koristiti metod POST (a ne GET) za preuzimanje ulaznih podataka korisnika*, da se ne bi osetljivi podaci prenosili u URL

12. *Kreirati bezbedne poruke o greškama koje ne otkrivaju poverljive informacije*

13. *Biti pažljiv kojim se podacima iz kolačića (cookies) veruje*, jer se njima može manipulisati

14. *Vršiti enkripciju konfiguracionih datoteka* koje sadrže poverljive podatke o korisnicima i serverima

15. *Ograničiti dužinu (broj znakova) poljâ za unos teksta*, kako bi se sprečili napadi otkazivanja usluge (DOS - *Denial of Service*) na aplikacijskom nivou

16. Ukoliko nije neophodno, *onemogućiti otvaranje datoteka koje se zadaju URL putanjom*

17. Ukoliko je moguće, *uključiti ograničenja bezbednog režima (safe mode) i iz kojih direktorijuma se mogu uključivati datoteke programskog kôda*

18. *Pažljivo izlagati javnom pristupu i imenovati datoteke sa ekstenzijama .bak, .txt, .inc*

19. *Pažljivo koristiti programske alate za upravljanje verzijom* u direktorijumima sa javnim pristupom

20. *Postaviti podrazumevanu adresu e-pošte za Reply-to na nivou veb aplikacije i pratiti e-poštu koja stiže na bounce adresu*, za prihvatanje pošte upućene na netačnu adresu za server e-pošte koji se koristi za aplikaciju

21. *Koristiti sisteme za upravljanje verzijama, praćenje programskih grešaka i izveštavanje o izmenama*

4. Bezbednost na strani servera veb aplikacija

Bezbednost na strani servera veb aplikacija podrazumeva moguće napade za koje se odbrana mora organizovati u programskom ili skript kodu koji se izvršava na serveru veb aplikacije.

4.1. Napadi sa umetanjem zlonamernog skript kôda

Provere ispravnosti podataka koji dolaze od korisnika (*input validation*) **izuzetno je važna i osnovna mera bezbednosti koju svaki programer veb aplikacija mora sprovoditi. Ubacivanje** (*injection*) **zlonamernog kôda** (najčešće preko prihvatanja ulaznih parametara) je prema najnovijem izveštaju specijalizovane svetske organizacije OWASP (*Open Web Application Security Project*) [6] najveća i najzastupljenija bezbednosna pretnja u današnjim veb aplikacijama. Suština dejstva napada je ubacivanje zlonamernog programskog kôda u polja za unos teksta koja je programer namenio za preuzimanje podataka od korisnika za funkcionisanje svoje aplikacije. Najefikasniji napad ove vrste svakako je **umetanje SQL koda** (*Structured Query Language Injection*) za direktno upravljanje radom baze podata-ka, jer vrši neposredan uticaj na sve podatke koji su centralizovani u bazi podataka poslovnog informacionog sistema, pa je zato svaki takav napad ukoliko uspe potencijalno najpogubniji za preduzeće. Osnovni primer za ilustraciju funkcionisanja ubacivanja SQL kôda je npr. unos teksta `' OR '1'='1` u polje sa HTML atributom npr. `id="un"`, u kom programer očekuje unos parametara naloga za prijavljivanje korisnika i obrađuje ga na sledeći način u programskom kôdu (na jeziku Java u primeru) u delu teksta koji će proslediti kao komandu na jeziku SQL bazi podataka na koju se povezao:

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("un") +"'";
```

Posle direktnog umetanja preuzete vrednosti za parametar un u tekst SQL komande, ova komanda će izgledati ovako:

```
SELECT * FROM accounts WHERE custID='' OR '1'='1'
```

što predstavlja uvek ispunjen uslov za preuzimanje parametara svih naloga (`'1'='1'` u uslovu `OR` u SQL) iz tabele `accounts`.

Moguće je za uticaj na SQL kôd koristiti i znake `--` ili # koji čine da je za interpretator jezika SQL komentar sve što sledi iza njih, čime je recimo moguće u polju za unos teksta korisničkog imena iz prethodnog primera ukloniti deo uslova koji bi se odnosio na podudarnost lozinki. I ostali specijalni znaci jezika SQL, kao što su `||` za konkate-naciju (nadovezivanje) tekstualnih podataka, `%` kao džoker (*wildcard*) znak za označavanje proizvoljnog broja proizvoljnih znakova u SQL konstrukciji `LIKE` za ispitivanje da li dati tekstualni podatak odgovara šemi vredn-osti, npr. `un LIKE '%ojan%'` ispunjavaju korisnička imena *Bojan*, *Bojana*, *Stojan*, *Stojana*, *Stojanka* itd. Takođe, umetanjem `IN(podupit_subquery)` moguće je zaobići željeno i dobiti novo filtriranje podataka.

U poljima u kojima se vrši pretvaranje podataka iz URL moguće je napad obaviti i sa URL kodovima, npr. koji predstavljaju razmak (`%20` običan razmak, `%09` tabulator, `%0D` prenos u novi red), npr. `%09select`.

Možda još opasniji napadi mogu biti sa akcionim upitima (koji menjaju podatke). Npr., u polju na obrascu za izmenu lozinke, unosom teksta `novalozinka'--` moguće je promeniti lozinku *svim* nalozima, jer je SQL iskaz za izmenu lozinke sa prihvaćenom novom lozinkom **`UPDATE accounts SET pwd= 'novalozinka'--' WHERE custID= usr AND pwd='staralozinka';`** (sve posle `--` se

smatra da je komentar!).

Kod umetanja SQL kôda u uskladištenu proceduru (*stored procedure*) moguće je promeniti njeno izvršavanje dodavanjem u ulazne parametre komandi `,@promenljiva` ili `PRINT @@promenljiva` za dodavanje novih lokalnih `@` i globalnih `@@` promenljivih i prikaz njihovih vrednosti.

Jedno od Kodovih 12 pravila potpuno relacionog sistema baza podataka je i da se meta-podaci o strukturi (definiciji) tabela i ostalih objekata moraju takođe nalaziti uz ostale podatke i njima se takođe može pristupiti iz SQL kôda. Zato je i struktura baze podataka podložna napadima umetanja SQL.

Zbog izuzetnog značaja provere ispravnosti ulaznih podataka, u programske biblioteke i proširenja koji služe za pristup bazama podataka danas su već ugrađeni mehanizmi za proveru ispravnosti ulaznih podataka u tzv. *pripremljenim iskazima* (*prepared statement*) *jezika SQL*. Slede primeri primene priprem-ljenih iskaza u biblioteci PDO (*PHP Data Objects*) za pristup podacima iz baza podataka u skript jeziku PHP:

```
$dbh = new PDO ('mysql:dbname=testbp;host=localhost', 'korisBP', 'lozinka');

$sql = 'SELECT ime, boja, kalorije FROM voce WHERE kalorije < :kal AND boja = :boja';

$sth = $dbh->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_FWDONLY));

$sth->execute(array(':kal' => $_REQUEST["Kalor"], ':boja' => $_REQUEST["boja"]);

$voce = $sth->fetchAll();
```

Druga varijanta pripremljenog SQL iskaza je (ostali kôd je isti kao u prvom primeru):

```
$sql = 'SELECT ime, boja, kalorije FROM voce WHERE kalorije < ? AND boja = ?';//...

$sth->execute(array($_REQUEST["Kalor"],$_REQUEST["boja"]);
```

Napad umetanjem SQL ne sprečava se ukoliko neki od parametara pripremljenog SQL iskaza zamenjuje ime tabele (npr. iza `FROM`).

Sve pomenute slučaje treba razmatrati pri testiranju bezbednosti veb aplikacija.

Pored umetanja podataka korisnika veb aplikacije u tekst SQL komande, moguće je umetanje i u bilo koji drugi interpretirani jezik, npr. u tekst komandi za LDAP (*Lightweight Directory Access Protocol*) server za proveru identiteta, XML, XML Path za izdvajanje XML podataka i dr.

Umetanje datoteke (slike) može biti neophodno u informacionim sistemima (npr. fotografija korisnika). Standardna veb kontrola za izbor datoteke takođe može biti način za umetanje zlonamernog kôda koji se obič-no nalazi u datoteci slike koja se postavlja na server.

4.2. Razbijanje provere identiteta i sesije

Veb stranice koriste objekat skripta na strani servera koji se zove `Session` (sesija) da bi čuvale informacije za različita učitavanja (istih i ostalih) veb stranica na istom serveru od strane istog veb čitača, jer ova funkcija nije predviđena protokolom HTTP. U PHP skriptu to je globalni niz `$_SESSION` sa tekstualnim indeksima (asocijativni niz). Najvažnija primena sesije je kod provere identiteta korisnika za sva obraćanja veb serveru (i u veb ajax aplikacijama). Ukoliko je veb aplikacija loše programirana, provera identiteta može dati napadaču mogućnost da dođe do lozinki ili na drugi način iskoristi ovlašćenja legitimnih korisnika. Ova vrsta napada zauzima 2. mesto u izveštaju [6]. Važno je da programeri budu svesni da pored anonimnih, mogu postojati napadači i među legitimnim korisnicima. Nije uvek lako ni izabrati da li koristiti neki provereni(ji), ali i potencijalnom napadaču poznatiji koncept (u nekoj programskoj biblioteci), ili primeniti originalno rešenje provere identiteta koje može imati neproverene ranjivosti na napade.

Napadi koji kradu lozinke najčešće se svode na umetanje SQL kôda, ali postoje i napadi na sesiju, koja se obično čuva u kolačiću veb čitača ili u URL. U parametru sesije `sessionid` čuva se kriptografski slučajni

broj koji je jedinstven za uspostavljenu vezu jednog veb čitača sa serverom (najčešće od uspešne provere identiteta korisnika do njegove odjave). Ukoliko se čuva u URL, broj sesije se mora prosleđivati u okviru svakog obraćanja veb serveru (u GET zahtevu ili stalno prepisivati u POST klijentskim skriptom ili pomoću skrivenih HTML polja). Broj sesije je jedino što se eventualno čuva u veb čitaču i služi kao ključ do promenljivih sesije, koje se isključivo čuvaju na serveru (npr. ključno polje prijavljenog korisnika). Napadi koji pokušavaju da ukradu broj sesije nazivaju se *otmica sesije* (*session hijacking*). Čuvanje broja sesije u kolačiću smatra se kao uopšte bezbednije rešenje, mada je podložno pokušajima sa drugih sajtova da ukradu kolačić, koji je, kao i ostalo što je vezano za veb čitač, dostupan iz Javascript kôda preko DOM objekta `document.cookie`. Pored postupaka na nivou mrežne infrastrukture za presretanje saobraćaja, kao što su korišćenje zadate putanje rutiranja (*source-routed IP*) ili napad "čoveka u sredini" ("*man-in-the-middle*"), postoje četiri načina otmice sesije:

1. **fiksiranje sesije**, kada napadač postavi neki njemu poznati broj sesije, recimo slanjem e-poruke korisniku koja sadrži link koji menja broj sesije i onda samo čeka prijavu tog korisnika
2. **otimanje sesije po strani**, korišćenjem manjkavosti bezbednosti mrežne infrastrukture, ali i nekorišćenja HTTPS u svim veb stranicama, napadač krade kolačić sa brojem sesije, pri čemu su posebno ranjive bežične pristupne tačke sa nedovoljno bezbednosti
3. **fizičko otimanje sesije**, kada napadač ima pristup datotekama i memoriji servera
4. **skript sa drugog sajta** (biće opisan u odeljku 5.1), kada napadač prevari veb čitač korisnika da izvrši zlonamerni kôd koji otkrije sadržaj kolačića

Metodi za sprečavanje napada na sesiju su:

1. enkripcija saobraćaja između servera i veb čitača (HTTPS) za celokupni sajt, nije dovoljno za sprečavanje fiksiranja sesije
2. korišćenje dugačkog slučajnog broja sesije koji otežava pogađanje
3. kreiranje novog broja sesije posle uspešne prijave, što sprečava fiksiranje sesije, jer napadač ne zna broj sesije posle prijavljivanja korisnika
4. neki sajtovi pri svakom obraćanju veb serveru traže dodatnu proveru identiteta korisnika, recimo istu IP adresu iz prethodnog obraćanja, što može biti neprijatnost korisnicima ukoliko imaju dinamičku IP adresu
5. provera vrednosti kolačića sa svakim novim zahtevom serveru dosta smanjuje prostor dejstva i olakšava otkrivanje napada, ali može uzrokovati mnoge tehničke probleme (npr. dva uzastopna legitimna zahteva mogu izazvati grešku tokena)

4.3. Nebezbedne direktne reference na objekte

Napadač, kada je ovlašćeni korisnik sistema, može promeniti vrednost parametra koji direktno ukazuje na sistemski objekat i tako koristiti drugi objekat za koji nije ovlašćen. Za sve takve moguće slučajeve programer mora predvideti sistem provere ovlašćenja i koristiti indirektne reference na nivou sesije. Ova vrsta napada zauzima 4. mesto u [6].

4.4. Neproverena preusmeravanja i upućivanja (prolazak do direktorijuma)

Na 10. mestu u izveštaju OWASP [6] nalazi se grupa napada koja nastaje usled neproverenog preusmeravanja na druge sajtove ili upućivanja na lokalni URL čija je putanja zapravo zabranjena za javno korišćenje (zaobilaženje kontrole pristupa datotekama).

Posebnu grupu čine tzv. *phishing* **napadi** (od *fishing* - pecanje). Oni nisu ostvarljivi samo u veb aplikacijama, već i u ostalim servisima interneta. Kod veb aplikacija, umetnuti zlonamerni kôd je najčešće link koji lakovernog korisnika kamuflirano vodi do veb stranice napadača, dok korisniku izgleda da je na legitimnoj veb stranici, npr. `<a href=" www.zlonamernisajt.com ">www.legitimnisajt.com</a>` (stvarna URL do koje vodi link zadaje se u atributu `href` oznake `<a>`). Mogućnosti za umetanje sličnih linkova su veće sa primenom Web 2.0 principa povezivanja veb stranica (RSS dovoda - *Really Simple Site feed*, spravica - *widget* / *gadget* na ličnim početnim stranicama i poslovnim *mashup* -

"mešavinama"), koji svi zaobilaze pravila izvršavanja samo veb sadržaja sa istog domena. Izvori *phishing* napada mogu biti i lažni baneri (koji liče na legitimne, a lako se postavljaju legitimnom procedurom kod pružalaca usluge oglašavanja) ili lažni iskačući (*popup*) prozori umetanjem odgovarajućeg Javascript kôda, koji preuzimaju funkcije na legitimnim sajtovima. Budući da se ovi napadi u velikoj meri zasnivaju na nepažnji korisnika, veb čitači dosta preuzimaju zaštitu na sebe i njihovi programeri su najodgovorniji za propuste ukoliko se npr. preko veb čitača preuzmu razni virusi (recimo *key logger* - beležnik šta je korisnik otkucao na tastaturi). Programer veb aplikacije treba da izbegava druge vrste propusta u svom kôdu koji bi omogućili umetanje lažnih adresa.

Prolazak do direktorijuma (*directory traversing*) je primer zaobilaženja kontrole pristupa "zbunjivanjem" veb servera u određivanju putanje. Do ovog zbunjivanja može doći zbog višeznačnog beleženja istih URL, npr. po apsolutnoj i relativnoj putanji, ili istih znakova (recimo ćirilično i latinično *A*) u kodiranjima promenljive dužine, gde spada i najzastupljeniji prikaz skupa znakova (*encoding*) UTF-8. U svim slučajevima višeznačnosti veb server primenjuje postupak svođenja na *kanonični*, najprostiji ekvivalentni oblik. Ukoliko to ne uradi na pravi način, onda je ranjiv na ovu vrstu napada. Jedan primer bio bi zabrana izvršavanja datoteka van foldera `C:\inetpub\wwwroot\cgi-bin` a dođe do izvršavanja na putanji `C:\inetpub\wwwroot\cgi-bin\..\..\..\Windows\System32\cmd.exe`, koja vizuelno podseća na poddirektorijum zadatog direktorijuma, ali je zapravo sa znacima za relativnu putanju njen kanonični oblik `C:\Windows\System32\cmd.exe`.

Za sprečavanje cele grupe napada najbolje je ne koristiti preusmeravanje. Ukoliko se ipak koristi, onda kod izračunavanja putanje ne uzimati u obzir podatke od korisnika. Ako je neophodno da korisnik utiče na izbor putanje, onda je bolje da mu se predstavi izbor od nekoliko ponuđenih putanja.

5. Bezbednost veb ajax aplikacija

Bezbednost veb Ajax aplikacija (u užem smislu) obuhvata sve vrste napada koje treba sprečavati u klijentskom *Javascript* kôdu. Pored pomenutih principa povezivanja, otvoreni interfejsi za programiranje *Web* 2.0 aplikacija (API - *Application Programming Interface*) u klijentskom Javaskriptu dodatno povećavaju rizik od napada, jer napadač može steći uvid, ali i izmeniti funkcije API.

5.1. Napad skript kôdom sa drugog sajta

Napad skript kôdom sa drugog sajta (CSS ili XSS - *Cross-Site Scripting*) koriste korisnički definisanu URL ili neprovereni unos teksta. Najčešće su rezultat lošeg programiranja veb aplikacije. XSS nastaje kada veb aplikacija preuzme i prosledi veb čitaču neproverene sadržaje i to omogući napadaču da izvršava skriptove u veb čitaču žrtve, čime mogu preuzeti parametre prijavljenog korisnika (sesije), izmeniti (*deface*) veb stranice (npr. dodati slojeve nevidljive korisniku koji mogu primati događaje stranice - otmica klikova, *clickjacking*), ili preusmeriti korisnika na zlonamernu veb stranicu. Do ovog napada može doći čak i na statičnim veb stranicama. Ova bezbednosna pretnja nalazi se na 3. mestu po ozbiljnosti i zastupljenosti napada [6]. Postoje dve vrste XSS napada:

1. kada su podaci privremeno *sačuvani* na napadnutom serveru, u bazi podataka, komentarima, izveštaju o aktivnostima, a žrtva dobije zlonamerni kôd sa servera kada zatraži sačuvanu informaciju
2. kada se podaci *reflektuju* sa servera, recimo poruke o grešci, rezultati pretrage, a korisniku se nametne link koji sa drugog sajta koristi te podatke

Po mestu korišćenja neproverenih podataka:

1. XSS na serveru
2. XSS pomoću DOM u Javascript kôdu nebezbednim pozivom dodaju se neproverene podaci u DOM ranije učitane veb stranice

Moguće su sve 4 kombinacije po vrsti XSS i mestu korišćenja neproverenih podataka. Sačuvani XSS ima više šanse da uspe od reflektovanog, ali inače su podjednako opasni. Rizici su isti bez obzira i na mesto korišćenja podataka, ali je na strani (Ajax) klijenta

teže otkrivanje pomoću automatizovanih testova i uopšte. Treba obezbediti da svi podaci koji se vraćaju u veb čitač budu provereni.

5.2. CSRF - Napad sa drugog sajta krivotvorenjem HTTP zahteva

CSRF (*Cross-Site Request Forgery*) nalazi se na 8. mestu u [6]. Koristi krivotvorenu verziju legitimnog HTTP zahteva, koju napadač obično pošalje u linku u okviru e-poruke legitimnom korisniku. Obično se koristi ranjivost veb aplikacije da prihvata obe vrste HTTP transporta (POST i GET), ali nije dovoljno dopuštati samo HTTP POST metod transporta, čak ni provera polja `Referer`, ni transakcije u više koraka. Pravo rešenje mogu biti sistemi CAPTCHA za sprečavanje mašinskog prijavljivanja bez korisnika ili još bolje **jednokratni sigurnosni tokeni** koje aplikacija dodaje u *Javascript* kôdu za atribute `href`, `src` i u skrivenim poljima na svim formama. Treba istaći da se u konzolama veb čitača (izmenom osobine `prototype`) mogu krivotvoriti Javaskript objekti, pa i *ajax* zahtev `XMLHTTPRequest`, protiv čega *nema odbrane*, ali se može posvetiti pažnja sprečavanju posledica, pre svega na serverskoj strani.

6. Zaključak

Ovaj rad predstavlja uvodni pregled i klasifikaciju mogućih bezbednosnih pretnji koje se stvaraju (neadekvatnom) primenom veb aplikacije za interaktivni korisnički interfejs poslovnih informacionih sistema. Cilj je pre svega da se upozore programeri ovog interfejsa i informacionih sistema u celini da obrate dovoljno pažnje pitanjima bezbednosti. Smernice koje bi programeri uvek trebalo da poštuju su da treba obraditi svaki ulazni podatak korisnika obavezno na serverskoj strani, a obradu izlaznih podataka koji se šalju veb čitaču korisnika treba uraditi i na klijentskoj i na serverskoj strani radi sprečavanja XSS. Za rad aplikacije trebalo bi koristiti HTTPS da bi se enkripcijom saobraćaja otežala otmica sesije i podataka prijavljenog korisnika. Ubuduće treba razmotriti i pitanja bezbednosti koja donosi standard HTML 5 i ispitati bezbednost pojedinih biblioteka kôda.

Bibliografija:

1. http://www.applicure.com/blog/database-security-best-practice, *Database Security Best Practices*
2. http://www.rackaid.com/resources/server-security-tips, *79 Web Server Security Tips*
3. http://www.pcmag.com/article2/0,2817,11525,00.asp, *Web Server Security Best Practices*
4. http://www.tenable.com/products/nessus, *Nessus Vulnerability Scanner: Fueled by Nessus ProfessionalFeed*
5. http://www.applicure.com/Products/dotdefender, *Web Application Firewall | PCI DSS Compliance*
6. https://www.owasp.org/index.php/Top_10_2013-T10, *Top 10 2013-Top 10 – OWASP*

Istorija rada:
Rad primljen: 06.05.2013.
Prva revizija: 22.05.2013.
Prihvaćen: 26.05.2013.