UDC: 004.738.5:004.896 INFO M: str. 42-52

PRIJEDLOG RJEŠENJA ZA DETEKCIJU I KLASIFIKACIJU SIGURNOSNIH PROPUSTA WEB APLIKACIJA
SOLUTION PROPOSAL FOR DETECTION AND CLASSIFICATION OF WEB APPLICATIONSECURITY VULNERABILITIES

Ognjen Joldžić, Zoran Đurić

REZIME: U ovom radu su analizirani sigurnosni aspekti web aplikacija u informacionim sistemima. Izloženi su tipovi arhitektura web aplikacija i značajne osobine vezane za sigurnost informacionih sistema. Prikazane su osobine najzastupljenijih napada na web aplikacije, kao i načini zaštite za svaki od opisanih napada. Opisani su načini testiranja web aplikacija, tehnike za otkrivanje sigurnosnih propusta, načini klasifikacije testova prema osobinama i načinu izvođenja i dat je pregled postojećih rješenja za testiranje sigurnosti web aplikacija. U drugom dijelu rada detaljno je opisan WASTT - novo rješenje za otkrivanje i klasifikaciju sigurnosnih propusta u web aplikacijama, razvijeno od strane autora ovog rada. Prikazana je modularna struktura sistema, način upotrebe i mogućnosti razvijenog sistema.
KLJUČNE REČI: sigurnosni aspekti web aplikacija, tipovi arhitektura web aplikacija, načini testiranja web aplikacija, WASTT
ABSTRACT: This paper presents an extensive analysis of security aspects in modern web applications and information systems. An overview of web application architecture types and parts of their functionalities that specifically relate to information security is also given. Some of the most commonly found attacks against web applications are shown, along with respective solutions for each of the described attacks. The paper also explains the techniques for web application testing and vulnerability detection, and classifications according to features and execution procedures, with a special overview of the available solutions in this field. The second part of the paper contains a detailed review of WASTT – a new solution aimed for detection and classification of security vulnerabilities in web applications, developed by the authors of this paper. The review contains a comprehensive description of the modular structure, usage procedures and features of the developed system.
KEY WORDS: security aspects in web applications web application architecture types, techniques for web application testing, WASTT

1. UVOD

U posljednjih nekoliko godina web aplikacije sve više dobijaju na značaju. Pored standardnih web-baziranih aplikacija kao što su web prodavnice ili aplikacije za *online* bankarstvo, pojavljuju se i nove vrste web aplikacija (poput Google Docs ili Google Calendar) koje uspješno zamjenjuju tradicionalne aplikacije. Kako se web aplikacijekoriste za različite poslovne primjene, postalesu i veoma atraktivan cilj za potencijalne napadače. Iz tog razloga njihova sigurnost je postala veoma važan segment, kako pri njihovom projektovanju, tako i pri implementaciji. Rezultati narušavanja sigurnosti web aplikacija mogu biti različiti: od narušavanja ugleda vlasnika ili korisnika aplikacije, do velike finansijske koristi za potencijalnog napadača. Veoma često ciljevi različitih vrsta napada su web aplikacije kod kojih narušavanje sigurnosti može rezultirati otkrivanjem različitih povjerljivih informacija, poput brojeva kreditnih kartica ili informacija o korisničkim nalozima.

Posljednjih godina primjetno je značajno povećanje broja napada na web aplikacije. Prema izvještaju Cenzic-a za posljednji kvartal 2009. godine, a na bazi informacija iz NIST, MITRE, SANS, US-CERT, OSVDB i drugih baza o ranjivostima, 82% ranjivosti odnosi se na web aplikacije i vezane tehnologije [1]. Brojni sigurnosni incidenti rezultirali su otkrivanjem brojeva kreditnih kartica miliona korisnika.Mnoge ranjivosti web aplikacija rezultat su nepravilne validacije ulaznih parametara. Primjeri ovakvih ranjivosti su SQL Injection (SQLI) i Cross-Site Scripting (XSS).

Konvencionalnetehnike mrežne zaštite, poput firewall uređaja, različitih kriptografskih tehnika ili korištenja HTTPS (*Hypertext Transfer Protocol Secure*) protokola za pristup web aplikaciji, nisu dovoljne. Dodatni problem sigurnosti web aplikacija predstavljaju programerske greške nastale u toku razvoja samih aplikacija. Iako je veliki broj ranjivosti web aplikacija moguće jednostavno razumjeti i eliminisati u toku razvoja same aplikacije, mnogi razvojni timovi nemaju adekvatno znanje na ovom polju. Kao razultat njihovog rada, na internetu se pojavljuju brojne ranjive web aplikacije.

U ovom radu predstavljen je WASTT, modularni alat razvijen od strane autora ovog rada namijenjenza automatsku analizu sigurnosti i detekciju ranjivosti web aplikacija. Alat posjeduje module za skeniranje web aplikacija, module za analizu i detekciju SQLI i XSS ranjivosti, kao i module za izvještavanje. WASTT alat je uspješno korišten za detekciju većeg broja ranjivih web aplikacija.

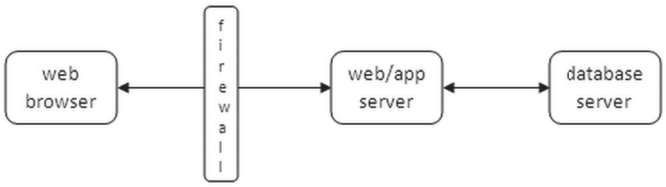
2. WEB APLIKACIJE

Web aplikacije su vrsta aplikacija koje se izvršavaju na aplikativnom serveru i koje putem HTTP (*Hypertext Transfer Protocol*) ili HTTPS (*Hypertext Transfer Protocol Secure*) protokola konzumiraju „tanki“ klijenti (web čitači). Navigacija kroz aplikaciju vrši se putem linkova ili unošenjem URL (*Uniform Resource Locator*) adresa u Web čitač. Prenos parametara od klijenta na stranu servera vrši se, najčešće, putem web formi. Svaka URL adresa se mapira u odgovarajući kod

na strani servera, a rezultat izvršavanja koda na strani servera se, u formi formatiranog HTML (*Hypertext Markup Language*) koda, vraća web čitaču, gdje se renderuje i prikazuje.

HTTP je protokol aplikativnog nivoa OSI (*Open Systems Interconnection*) referentnog modela koji ne pamti stanja, tj. protokol kod kojeg se svaki zahtjev procesira nezavisno, bez bilo kakve informacije o prethodnom zahtjevu. U cilju praćenja sesije korisnika web aplikacija mora da implementira neke od sljedećih tehnika: korištenje kolačića, prepisivanje URL-a i korištenje skrivenih polja u formama [2].

Na slici 1 data je klasična arhitektura troslojne Web aplikacije sa mrežnim firewall uređajem čija je osnovna namjena zaštita web/aplikativnog servera od napada sa mreže. Bez obzira na postojanje mrežnog firewall uređaja u ovakvoj arhitekturi, neophodno je obezbjediti pristup do web aplikacije kako bi ona mogla biti korištena od strane korisnika. Komunikacija između web čitača korisnika i web aplikacije može biti otvorena (kada se aplikaciji pristupa putem HTTP protokola) ili kriptovana (kada se aplikaciji pristupa putem HTTPS protokola).



Slika 1. – *Klasična arhitektura troslojne Web aplikacije sa mrežnim firewall uređajem koji štiti web/aplikativni server.*

Bez obzira na to da li se za pristup web aplikaciji koristi HTTP ili HTTPS protokol, potencijalni napadač neće biti spriječen da izvrši različite vrste napada na web aplikaciju, poput SQLI ili XSS napada.Razlog zbog koga je pomenuto moguće jeste što se napadi na web aplikacije odvijaju na aplikativnom nivou OSI referentnog modela, te što se sigurnost web aplikacije bazira na sigurnom kodu aplikacije, a ne na enkripciji komunikacije, operativnom sistemu na kojem se izvršava web aplikacija i sistem za upravljanje bazom podataka ili postojanju mrežnog*firewall* uređaja. Mnoge programerske greške i propusti prouzrokuju različite vrste ranjivosti web aplikacija, koje se mogu eksploatisati različitim vrstama napada.

3. TIPIČNI NAPADI NA WEB APLIKACIJE

Različiti propusti u projektovanju i implementaciji web aplikacija omogućavaju uspješno izvršavanje brojnih napada na same aplikacije. Kao najznačajniji od ovih propusta mogu se izdvojiti: nepravilna validacija parametara ili nepostojanje validacije, nepravilnaautentikacija, nepravilno upravljanje korisničkim nalozima i sesijama, nepravilno rukovanje izuzecima, loša upotreba kriptografskih tehnika i algoritama, loš način generisanja i čuvanja kriptografskih ključeva, nepostojanje kriptovane komunikacija ilošarestrikcijapristupastranicama web aplikacije.

Kao najznačajniji i najzastupljeniji napadi na web aplikacije mogu se izdvojiti SQLI i XSS. Prema izvještaju Cenzic-a za posljednji kvartal 2009. godine, XSS i SQLI ranjivosti predstavljaju 19% i 17% svih ranjivosti u komercijalnim web aplikacijama, respektivno [1].

3.1 *SQL Injection napad*

Web aplikacija je ranjiva na SQL injection napad ako je napadač u mogućnosti da ubaci SQL naredbu u postojeći SQL upit aplikacije, tj. ako je napadač u mogućnosti da izmijeni semantiku postojećeg SQL upita aplikacije. Ovo se postiže izmjenom vrijednosti parametara koje se koriste za izgradnju SQL upita, najčešće ubacivanjem malicioznih stringova u odgovarajuća polja na web formama.

Za potrebe analize SQL injection napada biće pretpostavljeno da web aplikacija upit dat na slici 2 koristi za autentikaciju korisnika. Ovim upitom se provjerava postojanje korisničkog imena i lozinke koju je korisnik web aplikacije unio na formu za prijavu (slika 3) koja se nalazi na odgovarajućoj stranici za prijavu korisnika. Ako dato korisničko ime i lozinka postoje, upit vraća vrijednosti polja *id*, *first_name* i *last_name*i smatra se da je korisnik uspješno prijavljen.

```
SELECT first_name, last_name, address, last_login FROM students WHERE username =
'marko' AND password = 'markovic';
```

Slika 2. – *Primjer SQL upita koji se koristi za autentikaciju korisnika.*

Ovakvi i slični SQL upiti kojim se provjeravaju korisnički podaci, a posebno upiti čije uspješno izvršavanje omogućava dobijanje odgovarajućih prava i privilegija nad samom web aplikacijom, su veoma čestoprimarne mete napadača.

```
<form name="loginForm" method="P" action="login.jsp">
      <table>
            <tr><td>Username:</td>
                <td><input type="text" name="usernams" value=""></td>
            </tr>
            <tr><td>Password:</td>
                <td><input type="password" name="password" value=""></td>
            </tr>
            <tr><td></td>
                <td><input type="submit" value="Submit"></td>
            </tr>
      </table>
</form>
```

Slika 3. – *HTML forma za prijavu korisnika.*

Kada korisnik *submit*-uje formu, vrijednosti polja forme (*username* i *password*) se koriste za konstrukciju SQL upita kojim se vrši autentikacija korisnika. Na slici 4 dat je segment koda, koji se odnosi na konstrukciju pomenutog SQL upita, metode Java *bean*-a ili Java servleta koja vrši autentikaciju korisnika.

```
String sql = "SELECT first_name, last_name, address, last_login FROM students "
      + " WHERE username = '" + username + "' AND password = '" + password + "'";
```

Slika 4. – *Konstrukcija SQL upita za autentikaciju korisnika.*

Ako aplikacija ne vrši pravilnu validaciju unesenih vrijednosti u polja HTML forme (slika 3), napadač može,putem polja forme, u upit ubaciti takav string koji će izmijeniti semantiku originalnog upita. Za potrebe demonstracije SQL *injection* napada važi pretpostavka da napadač za korisničko ime i lozinku unosi vrijednosti „*' OR 1=1--*“i „abcdefghijkl“, respektivno. Na bazi datih vrijednosti parametara, ranjiva web aplikacija kreira SQL upit za autentikaciju korisnika. Novonastali SQL upit dat je na slici 5.

```
SELECT first_name, last_name, address, last_login FROM students
WHERE username = '' OR 1=1-- ' AND password = 'asdfghjklasdf'
```

Slika 5. – *Novokreirani SQL upit za autentikaciju korisnika.*

Znak „--“ predstavlja komentar u SQL jeziku, tako da je dio SQL upita koji se nalazi iza ovog znaka u potpunosti ignorisan od strane sistema za upravljanje bazama podataka. Korištenjem jednostrukog znaka navoda u vrijednosti parametra *username* string korisničkog imena u SQL upitu sa slike 4 je uspješno zatvoren, a preostali dio vrijednosti parametra *username* (*OR 1=1*) proširuje upit uslovom koji je uvijek istinit. Izvršavanjem ovog upita, sistem za upravljanje bazom podataka će vratiti sve zapise iz tabele *students*, što će aplikacija prepoznati kao uspješnu autentikaciju korisnika. Zbog funkcionalnosti znaka „--“, jasno je da napadač kao lozinku može unijeti proizvoljnu vrijednost.

```
' union select 1,load_file('D:\passwords.txt'),1#
' union select user, 1, 1 from mysql.user#
' union select distinct table_schema, 1, 1 from information_schema.tables limit 0,1#
' union select distinct table_schema, 1, 1 from information_schema.tables limit 1,1#
```

Slika 6. – *Primjeri vrijednosti parametra username na osnovu kojih se mogu konstruisati maliciozni SQL upiti.*

Osim ovog osnovnog SQLI napada, poznatog i kao „OR 1=1“ napad, moguće je izvršiti i brojne druge. Izvršavanjem ovih napada, osim dobijanja podataka iz baze podataka koju koristi ranjiva web aplikacija, moguće je ostvariti i druge ciljeve, kao što su: izmjena podataka u bazi, otkrivanje svih baza podataka na sistemu za upravljanje bazama podataka na kojem se nalazi baza koju koristi ranjiva web aplikacija, otkrivanje strukture izabrane baze podataka (nazivatabela i kolona), kreiranje novih naloga i definisanje njihovih privilegija za pristup bazi podataka, interakcija sa operativnim sistemom na kojem se izvršava sistem za upravljanje bazom podataka (kroz čitanje iz i pisanje u sistemske datoteke na disku ili direktno izvršavanje naredbi), narušavanje strukture baze podataka ili njeno brisanje. Jasno je da je za ostvarivanje navedenih ciljeva neophodno da korisnički nalog pod koji aplikacija koristi za pristupa bazi podataka ima za to odgovarajuće privilegije.Na slici 6 su date različite vrijednosti parametra *username* koje će nakon *submit*-a forme za prijavu korisnika prouzrokovati kreiranje i izvršavanje različitih malicioznih SQL upita na MySQL sistemu za upravljanje bazom podataka.

3.2 *XSS napad*

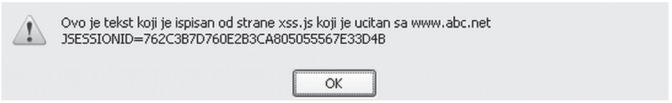
Najzastupljenija vrsta napada na web aplikacije je XSS[1]. XSS se odnosi na različite napade u kojima napadač ubacuje maliciozni JavaScript kod u web aplikaciju, putem vrijednosti GET parametara ili kroz HTML forme[3]. Kada legalan korisnik posjeti „ranjivu“ web stranicu sa malicioznim JavaScript kodom, doći će do njegovog izvršavanja. Na ovaj način zaobilazi se „*Same Origin Policy*“ sigurnosna politika web čitača [4]. U ovakvoj situaciji, bez obzira sa koje stvarne lokacije se maliciozni JavaScript učitava, web čitač će ga smatrati legalnim jer je kod za poziv tog JavaScript-a ugrađen u „ranjivu“ web aplikaciju kojoj korisnik pristupa. Maliciozni JavaScript, kao rezultat ovog napada može doći u posjed različitih sigurnosno osjetljivih informacija, poput kolačića i identifikatora sesije.

XSS napadi su, u opštem slučaju, jednostavni za izvršavanje, ali ih je teško spriječiti i mogu prouzrokovati veliku štetu. Postoje tri vrste XSS napada: reflektovani (nepostojani), uskladišteni (postojani) i DOM-bazirani napadi. Najčešći tip XSS napada je reflektovani XSS napad. Za potrebe analize reflektovanog XSS napada važipretpostavka da korisnik pristupa web aplikaciji na URL adresi www.xyz.com. Sljedeća pretpostavka koja važi jeste da se ne vrši validacija podataka unesenih putem forme za unos korisničkih podataka, sličnojweb formi sa slike 3. Kada maliciozni korisnik u polje za unos imena na ovoj formi unese string oblika <script src=”http://www.abc.net/malicious/xss.js”></script>, i submit-uje podatke u rezultirajućoj HTML strani će biti ugrađen ubačeni kod, tj. poziv malicioznog Java skripta. Na slici 7 dat je jednostavan primjer malicioznog JavaScript koda.

```
document.write ("Ovo je tekst koji je ispisan od strane xss.js koji je ucitan sa
www.abc.net " + document.cookie);
alert ("Ovo je tekst koji je ispisan od strane xss.js koji je ucitan sa www.abc.net
" + document.cookie);
```

Slika 7. – *Primjer JavaScript koda koji je korišten za demonstraciju XSS napada.*

Na slici 8 prikazan je rezultat izvršavanja alert funkcije Java skripta čiji je poziv ugrađen u rezultujuću HTML stranicu, dok je na slici 9 prikazan rezultujući HTML kod u koji je ugrađen poziv malicioznog Java skripta.



Slika 8. – *Rezultat izvršavanja XSS napada.*

```
<td>Korisnik:
        <b>
        <script src="http://www.abc.net/malicious/xss.js"></script>
        <b>
</td>
```

Slika 9. – *HTML kod u koji je ugrađen poziv malicioznog Java skripta.*

Pojavljivanje JavaScript koda u rezultujućoj HTML stranici je indikator ranjivosti web aplikacije na XSS napad. Očigledno je da se ovaj napad, kao i prethodni, može još jednostavnije izvršiti prosljeđivanjem malicioznih vrijednosti parametara putem GET zahtjeva (ako aplikacija to dozvoljava). XSS napadi se najčešće vrše putem stranica za pretraživanje, ali i putem formi za različite vrste unosa.

Osnovni problem napadača kod izvršavanja reflektovanog XSS napada jeste taj što se, na prvi pogled, ubacivanje malicioznog koda odražava na rezultujući HTML kod u web čitaču samog napadača. Sljedeći zadatak napadača je da legitimne korisnike navede da, bez znanja o tom činu, izvrše ovaj napad nad samim sobom. To se može izvršiti različitim tehnikama socijalnog inžinjeringa, na primjer, slanjem linka legitimnom korisniku (putem elektronske pošte) čije će otvaranje u web čitaču rezultirati izvršavanjem reflektovanog XSS napada. U realnoj situaciji maliciozni JavaScript kod bi mogao napadaču poslati kolačić legitimnog korisnika, koji sadrži podatke za autentikaciju ili neke druge sigurnosno osjetljive informacije.

Pored krađe sigurnosno osjetljivih informacija, poput kolačića, postoji i alternativan način eksploatacije XSS ranjivosti web aplikacije. Na primjer,korištenjem odgovarajućeg malicioznog JavaScript koda moguće je izmijeniti lokaciju na koju se podaci sa web forme *submit*-uju. Na ovaj način napadač može, pomoću ovakvog JavaScript koda, preusmjeriti *submit*-ovane podatke na server i aplikaciju kojoj on ima pristup, kako bi došao u posjed ovih podataka.

Druga vrsta XSS napada je uskladišteni XSS napad. Kod ove vrste XSS napada maliciozni kod se smješta na strani servera, u bazi podataka koju web aplikacija koristi. Na ovaj način se maliciozni kod izvršava više puta od strane svih legitimnih korisnika web aplikacije koji posjećuju stranicu u koju se ugrađuje maliciozni kod prethodno smješten u bazu podataka. Kod ove vrste XSS napada ne postoji problem navođenja legitimnih korisnika na izvršavanje napada, kao što je to slučaj sa reflektovanim XSS napadom.

DOM-bazirani napad je XSS napad čije izvršavanje rezultira modifikacijom DOM okruženja u web čitaču korisnika korišćenjem originalnog skripta sa klijentske strane, tako da se klijentski kod izvršava na neočekivan način. Ova vrsta napada poznata je i kao "type-0 XSS".

Najbolja zaštita od XSS napada postiže se kombinacijom white-box i black-box testiranja, kao i provođenjempenetracionog testiranja.

4. TIPOVI TESTOVA

Testovi sigurnosti web aplikacija se mogu grupisati na nekoliko različitih načina, u zavisnosti od osobina samog testa ili načina njegovog izvođenja. S obzirom da je broj postojećih napada na web aplikacije svakim danom sve veći, jedan od načina klasifikacije testova je prema tipu napada za čije otkrivanje je konkretan test prilagođen. Testovi koji su specifični za jednu grupu napada (kakvi su, recimo, SQLI testovi) često uključuju detaljnije procedure za ispitivanje rezultata i kompleksnije metrike za utvrđivanje uspješnosti napada od testova opšte namjene.

Testovi koji dosljedno simuliraju napade na web aplikacije i koji mogu imati destruktivan učinak na testiranu aplikaciju se svrstavaju u grupu invazivnih testova. Iako su ovakvi testovi po svojoj prirodi bliži situacijama sa kojima bi se sistem u upotrebi mogao suočiti, neophodno je obratiti posebnu pažnju na način njihovog izvođenja. Kako bi se izbjeglo uništavanje podataka ili pojedinih dijelova sistema, ovakvi testovi se najčešće izvode u kontrolisanom okruženju, nad podacima koji se ne koriste u produkciji, već predstavljaju kopiju trenutnog stanja i sadržaja informacionog sistema. Na ovaj načinmoguće je ispitati ranjivosti sistema na čitav niz invazivnih testova koje u realnom okruženju nije moguće izvesti.

Neinvazivni testovi simuliraju efekte prethodno opisanih napada na način koji ne može dovesti do gubitka sadržaja, ali istovremeno pokazuje postojanje sigurnosnog propusta koji bi se u tu svrhu mogao iskoristiti. Iz tog razloga, neinvazivni testovi sigurnosti web aplikacija mogu se izvršiti i na produkcionom sistemu.

Najčešći oblik klasifikacije testova sigurnosti web aplikacija odnosi se na stepen integracije procesa testiranja unutar samog sistema. Prema tome, testovi se mogu podijeliti na dvije osnovne grupe, od kojih svaka ima svoje prednosti i nedostatke, te je u skladu sa tim osobinama potrebno i posmatrati dobijene rezultate. Najbolja praksa u realnim situacijama je kombinacija svih navedenih tipova testova, jer se samo tako može u dovoljnoj mjeri osigurati sigurno okruženje za rad testiranog sistema. Osnovne grupe testova koje definiše ova podjela su [5, 6]:

– black-box testovi – predstavljaju testove koji su po načinu izvođenja i stepenu integracije bliži realnim uslovima rada aplikacije. Napadač pristupa sistemu sa pozicije prosječnog korisnika, bez prethodnog detaljnijeg znanja o samom sistemu. Pri izvođenju testa, odnosno simuliranju stvarnog napada, korisnik bazira testiranje na podacima koje je saznao iz prethodnih pokušaja napada, pokušavajući na taj način otkriti informacije o strukturi sistema. U slučaju web aplikacija, dodatne informacije se dobijaju parsiranjem dostupnog izvornog koda (HTML, JavaScript): skup linkova prema ostalim stranama unutar aplikacije, broj i nazivi potrebnih parametara za pozive funkcija, kao i podaci koji su nepažnjom razvojnog tima ostali dostupni krajnjem korisniku (informacije koje se nalaze u komentarima unutar HTML koda). Prednost ovakvog pristupa testiranju je što su dobijeni rezultati bliži realnim uslovima. Napadač koji nema uvid u način funkcionisanja sistema u pozadini web aplikacije neće se moći fokusirati isključivo na određenu grupu testova. Tako je, na primjer, aplikacija koja sa forme za prijavu korisnika dobijene podatke provjerava na LDAP bazi podataka potpuno imuna na SQL injection napade. S druge strane, ovo može predstavljati i nedostatak, jer će sistem za testiranje, iz pomenutog razloga, morati da izvrši mnogo veći broj testova. Čak i za slučaj da je aplikacija potpuno sigurna, vrijeme i resursi potrebni da se obradi veliki broj zahtjeva mogu biti vrlo veliki, što samo po sebi može predstavljati drugi sigurnosni rizik (DoS – *Denial of Service*) i izazvati probleme drugačijeg tipa. Druga bitna negativna osobina black-box testova je nemogućnost revizije sigurnosti kompletnog sistema. U praksi je česta pojava aplikacija kod kojih su sigurnosni mehanizmi ograničeni samo na forme za pristup korisnika, dok su unutrašnji slojevi aplikacije i dalje podložni različitim vrstama napada i eksploataciji sigurnosnih propusta. Korisnik je u tom slučaju u mogućnosti da na dozvoljen način premosti inicijalne sigurnosne barijere, te da nakon toga na dijelu sistema koji nije dovoljno zaštićen izvrši neki od napada kojim će doći u posjed osjetljivh informacija.
– white-box testovi pružaju rješenja na neke od prethodno opisanih problema, ali otvaraju i niz drugih pitanja koja se moraju uzeti u razmatranje prilikom izbora odgovarajućeg testa za informacioni sistem. White-box testovi podrazumijevaju da je korisniku dostupna cjelokupna aplikacija,

te da se testovi izvode na svim njenim segmentima. Aplikacije za testiranje imaju na raspolaganju podatake, kao što su kombinacija korisničkog imena i lozinke ili klijentski digitalni sertifikat, koji će im omogućiti pristup zatvorenim dijelovima aplikacije. Osim toga, informacije o pozadinskoj strukturi sistema će omogućiti odbacivanje onih testova koji ne mogu dati pozitivne rezultate. Na ovaj načinbiće minimizirano procesorsko i korisničko vrijeme potrebno za testiranje i biće omogućeno bolje iskorištenje resursa za testiranje propusta koji su mogući na datoj platformi (ukoliko se utvrdi da se u pozadini nalazi Active Directory, onda nema potrebe za testiranjem SQLI upita). White-box sistemi za testiranje mogu biti implentirani u vidu proxy-ja čiji je zadatak praćenje kompletne komunikacije između web aplikacije i sistema za testiranje. Efekti ovakve implementacije zavise od okruženja u kojem je testirani sistem implementiran (da li je saobraćaj kriptovan, o kakvoj se mrežnoj tehnologiji se radi u konkretnom slučaju, itd.). Osnovni nedostatak ove grupe testova je u tome što se oni po svojoj primjeni u određenoj mjeri udaljavaju od realnih situacija, jer su potpuno otvoreni informacioni sistemi relativno malobrojni, pa se i situacije u kojima je white-box test mjerodavan sreću nešto rjeđe. S druge strane, ovim je eliminisan rizik od greške izazvane slučajnom greškom iskusnijih korisnika u radu sa testiranom aplikacijom (slučajan unos SQLI upita), što je kod prethodne grupe testova bilo označeno kao problem.

Važan segment svih tipova testova, bez obzira na implementaciju predstavlja procesiranje rezultata. Kod black-box testiranja to je nešto lakši zadatak, jer se web aplikacije uglavnom projektuju tako da imaju standardizovanu stranu za poruku o neispravnom korisničkom imenu (ili bilo kojem drugom parametru) koja se može upotrebiti kao referentna strana kod izvršavanja testova. Jednostavnim poređenjem dobijenog rezultata sa sadržajem za koji se zna da je generisan ispravno od strane aplikacije može se doći do zaključka da li je došlo do otkrivanja ranjivosti testirane aplikacije. Izuzetak su inteligentnije aplikacije koje imaju razvijene metode za detekciju automatskih pokušaja prijavljivanja (engl. botovi), kod kojih se nakon određenog broja pogrešnih pokušaja traži podatak koji zahtjeva ručni unos podatka od strane korisnika (prepoznavanje teksta na iskrivljenoj slici itd.). Međutim, čak i ovo predstavlja pozitivan rezultat testiranja, jer je nivo bezbjednosti u aplikaciji sa ovakvim mehaniznom sigurno viši nego u ostalim aplikacijama. Dodatni problem predstavljaju sistemi kod kojih se dinamički ažurira dio sadržaja strane za prijavu (portali, aplikacije sa dinamičkim oglasima itd), bez obzira na akciju korisnika i tačnost unesenih podataka. Kod ovakvih sistema često nije dovoljno poređenje dobijenog odgovora sa referentnim, jer u tom slučaju može doći do lažne detekcije sigrnosnog propusta. Da bi se i ovakvi slučajevi ispravno obradili, u opštem slučaju je potrebno formiranje kompleksnije metrike koja bi pri odlučivanju u obzir uzela i neki drugi faktor osim dužine dobijenog odgovora. Kreiranje ovakve metrike se dodatno komplikuje zbog činjenice da se testovi moraju prilagoditi konkretnoj aplikaciji i da administrator mora izvršiti kalibraciju pravila i filtriranje dobijenih podataka, kako bi rezultati testiranja bili mjerodavni.

Kod white-box aplikacija, situacija je još teža, jer ne postoji strana čija bi svrha bila samo detekcija unosa pogrešnih podataka i obavještavanje korisnika i koja bi se mogla iskoristiti kao referentna strana. Sistem za testiranje bi morao biti upoznat sa sadržajem svake strane koja se testira kako bi se mogla izvršiti procjena dobijenih rezultata. Ovdje je problem sa dinamičkim sadržajima posebno izražen, jer često izgled strane zavisi od trenutnog konteksta, prethodnih akcija korisnika unutar aplikacije i niza drugih relevantnih parametara. Opisani problemi se rješavaju različitim tehnikama. U svakom slučaju, potreban je veći broj referentnih pokušaja, kao i značajno učešće administratora u testiranju. Napredniji sistemi za testiranje mogu implementirati i određenu vrstu ekspertnog sistema, koji bi omogućio korištenje prethodno naučenih rezultata u sljedećim testiranjima. Uspješno učenje takvog sistema umanjuje, ali ne eliminiše potrebu za intervencijama od strane administratora sistema za testiranje.

Konkretnije gledano, prethodno opisani problem nije jednako izražen kod svih testova i aplikacija. Na primjer, XSS testovi se lakše procesiraju od SQL injection testova, dok je uspješnost testa na postojanje validnog digitalnog sertifikata na strani web/aplikativnog servera moguće utvrditi precizno u najvećem broju služajeva. Rezultati dobijeni bilo kakvim testiranjem moraju se posmatrati uzimajući u obzir tip testa, tip sistema koji vrši testiranje, parametri okruženja, kao i da se u opštem slučaju rezultati testa provedeni nad istom aplikacijom od strane različitih sistema ne mogu direktno porediti bez prethodnog prilagođavanja.

Svi prethodno navedeni problemi pokazuju još i da kod testiranja web aplikacija nije moguće postići potpunu automatizaciju, jer uvijek postoji dio sistema koji mora biti pod kontrolom administratora, bilo da se radi o kalibraciji pravila za prepoznavanje rezultata, ili o konfiguraciji samih procedura za testiranje i njihovog prilagođavanja trenutnom okruženju.

5. POSTOJEĆA RJEŠENJA ZA TESTIRANJE SIGURNOSTI WEB APLIKACIJA

Činjenica da je broj postojećih napada na web aplikacije svakim danom sve veći, i da su vrijednosti sistema i cijene korekcije eventualnih problema ili gubitka podataka značajne, dovela je do pojave niza rješenja usmjerenih na poboljšanje određenih segmenata sigurnosti informacionih sistema. Svakih od ovih sistema ima različite karakteristike i fokuse djelovanja.

Jedan način podjelesistema za testiranje je na open-source i komercijalna rješenja, pri čemu je bitno primijetiti da su određena rješenja iz obje grupe značajnije zastupljena u praksi. Pored pomenutog, bitno je napomenuti da postoji i određen broj akademskih rješenja koja nisu značajnije zastupljena u praksi.

Tabela 1. – *Komparativni prikaz postojećih rješenja.*

	ParosProxy	Rational AppScan	WebScarab	Nessus
Platforma	sve (Java)	Windows	sve (Java)	posebne verzije za različite operativne sisteme
Cijena (licenca)	besplatan	preko 6000 $, u zavisnosti od verzije	besplatan	besplatan za ličnu upotrebu
Autor (izdavač)	Chinotec Technologies Company	IBM	OWASP	Tenable Network Security
Proxy način rada	DA	NE	DA	NE
Black box testovi	DA	DA	DA	DA
White box testovi	DA	DA	DA	DA
Simuliranje HTTP GET zahtjeva	DA	DA	DA	DA
Simuliranja HTTP POST zahtjeva	DA	NE	DA	NE
Mogućnost izmjene HTTP zaglavlja	DA	DA	DA	NE
Podržan HTTP i HTTPS protokol	DA	DA	DA	DA
Analiza posebnih elemenata HTML koda	NE	DA	DA	NE
SQLI testovi	DA	NE	DA	DA
XSS testovi	DA	DA	DA	DA
Analiza sadržaja kolačića	DA	NE	DA	NE
Testiranje ostalih segmenata sistema	NE	NE	NE	DA
Mogućnost proširenja baze testova	NE	DA	DA	DA
Mogućnost razvoja sopstvenih modula	NE	NE	DA	NE
Mogućnost distribucije servisa	NE	NE	NE	DA
Sopstveni format za čuvanje rezultata	DA	DA	DA	DA
Sopstveni nivoi ozbiljnosti propusta	DA	DA	DA	DA
Postojanje adekvatne dokumentacije	minimalan opis funkcionalnosti	DA	kratko online uputstvo	DA

Važni kriterijumi koji se moraju uzeti u obzir prilikom vrednovanja aplikacija za testiranje sigurnosti web aplikacija su:cijena,otvorenost programskog koda,tipovi napada koje aplikacija može testirati,način testiranja (black-box ili white-box testiranje, invazivni i neinvazivni testovi),potrebno okruženje za rad aplikacije (operativni sistem, sopstveno grafičko okruženje, zauzeće resursa u toku rada itd), mogućnosti prepoznavanja pozitivnih rezultata i njihova klasifikacija (definisanje sopstvenih metrika ili korištenje postojećih),tip web aplikacija prema kojima je usmjeren sistem za testiranje (mogućnosti testiranja web servisa, ili isključivo „klasičnih“ web aplikacija), način čuvanja rezultata za kasniju analizu,mogućnosti interakcije sa administratorom sistema za testiranje u toku rada (validacija pojedinih rezultata, unos vrijednosti od strane korisnika, kreiranje sekvence za prijavu kosirnika),mogućnosti ažuriranja baza testova i unapređenje unutrašnjih segmenata aplikacije,dostupna podrška od strane razvojnog tima i redovno objavljivanje novih verzija.

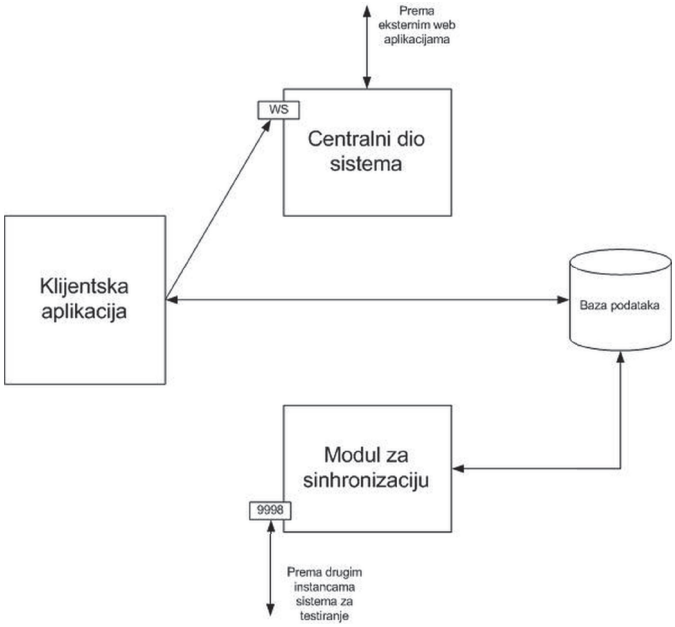
Utabeli 1 dat je prikaz najpoznatijih rješenja koja su trenutno značajnije zastupljena u praksi, sa najvažnijim parametrima za njihovo poređenje [7, 8, 9, 10,11, 12].

6. WASTT

WASTT (Web Application Security Testing Tool) je softversko rješenje za otkrivanje i klasifikaciju sigurnosnih propusta u web aplikacijama. Na bazi prednosti i nedostataka postojećih rješenja formirani su zahtjevi koje WASTTrješenje treba da zadovolji.Pri tome, ovo rješenje objedinjava korisne funkcionalnosti pronađene kod postojećih rješenja i uklanjanja većinu uočenih nedostataka.

Svaki sistem ovakvog tipa treba prvenstveno da omogući modularan i fleksibilan pristup problemu testiranja sigurnosti. Ovo je jedan od najznačajnijih ciljeva svakog sistema slične namjene, jer je samo dovoljno fleksibilnom i modularnom arhitekturom moguće održati korak sa konstantnim porastom broja sigurnosnih propusta u modernim sistemima.

Aplikacija je bazirana na principima otvorenog koda (engl. *open source*), tako da je svaki segment podložan eventualnoj nadogradnji. Osim toga, osnovni modul je realizovan u vidu web servisa, čime je riješen problem distribucije funkcionalnosti prema većem broju korisnika.



Slika 10. – *Osnovni segmentiWASTT sistema.*

Sistem se sastoji od sljedećih segmenata (slika 10): jezgro sistema, klijentski dio sistema i modul za sinhronizaciju.

6.1 *Jezgro sistema*

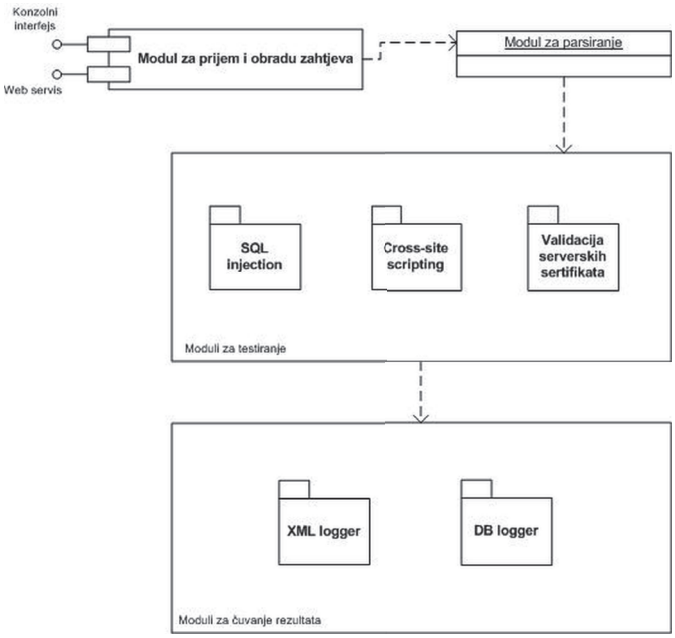
Jezgro sistema (*core* aplikacija) predstavlja dio sistema koji je odgovoran za obradu zahtjeva od strane klijenta i izvršavanje

konkretnih testova nad testiranim sistemima. Sastavljen je od proizvoljnog broja modula, od kojih svaki modul obuhvata procedure potrebne za testiranje određenog aspekta sigurnosnih propusta (odnosno, konkretnog tipa napada), kao i za klasifikaciju eventualnih propusta otkrivenih radom sistema. Arhitektura sistema koji je dio standardne instalacije omogućava vrlo jednostavno dodavanje novih modula i njihovo povezivanje sa ostatkom sistema. S druge strane, elementi koji povezuju module ne postavljaju nikakva ograničenja po pitanju implementacije unutrašnjih procedura koje pojedini moduli koriste. Način slanja podataka prema testiranoj aplikaciji, način njihovog prijema, parsiranje i obrada su u potpunosti prepušteni korisniku, odnosno razvojnom timu. Svaki novi modul obavezan je da poštuje samo propise vezane za povezivanje sa ostatkom sistema, kako bi njihov rad bio uopšte moguć. Osnovna implementacija omogućava tri različita tipa testova:

– SQL injection,

– XSS i

– testove za validaciju digitalnih sertifikata na strani servera.



Slika 11. – *Arhitektura jezgra WASTT sistema.*

Arhitektura prethodno definisanih modula aplikacije se može prikazati dijagramom na slici 11. Kako je većranije navedeno, modul može imati proizvoljnu internu arhitekturu, ali je postojanje sljedećih modula standard za sve postojeće i buduće module za testiranje:modul za parsiranje sadržaja, dio sistema za integraciju modula za testiranje, moduli za izvršavanje testova, moduli za čuvanje rezultata i interfejs prema eksternom sistemu.

6.1.1 Modul za parsiranje sadržaja

Osnovna funkcija modula za parsiranje sadržaja je slanje i prijem HTTP zahtjeva i razlaganje dobijenog odgovora na HTML objekte koji će kasnije biti korišteni, u daljem radu sistema. Prilikom svakog zahtjeva, vrši se ažuriranje skupa linkova koji vode sa jedne strane na drugu u okviru iste matične aplikacije. Postupak se ponavlja rekurzivno dok se skup linkova ne iscrpi, odnosno dok ne ostane niti jedan link čiji sadržaj strane već nije parsiran. Tek tada se sadržaj upisuje u bazu podataka i prelazi se na sljedeći korak. U ovoj aktivnosti učestvuju samo sistem i eksterna (testirana) web aplikacija, i pri tome ne postoji interakcija sa korisnikom sistema. Objekti od značaja za modul za parsiranje su (dati hijerarhijskim redoslijedom): web strane,forme,parametri na formama,parametri preneseni putem GET zahtjeva i linkovi i reference iz formi.

Web strana predstavlja najkrupniji element kod parsiranja web aplikacija. Aplikacija je sastavljena od više web strana, od kojih svaka može sadržavati proizvoljan broj formi, GET parametara i linkova. Da bi strana bila prihvaćena kao sastavni dio aplikacije, mora ispunjavati uslov da potiče sa istog domena kao i URL aplikacije koji je u zahtjevu zadao korisnik. Strane koje ne ispunjavaju ovaj uslov se odbacuju i njihov sadržaj se ne parsira.

Po prijemu HTML sadržaja, vrši se razlaganje elemenata po principu sličnom parsiranju XML dokumenata. Podrazumijeva se da je HTML na stranici formiran validno, ali je aplikacija u stanju parsirati i sadržaje koji ne ispunjavaju u potpunosti kriterijume validnosti. Svaki sadržaj od interesa na nekoj strani se klasifikuje u bazi podataka i u datoteci na sistemu. Elementi nižeg nivoa imaju kaskadne reference prema elementima višeg nivoa, te se zna koja forma pripada kojoj aplikaciji, koji parametar pripada kojoj formi itd. Pri pronalasku forme ili elementa označenog tagom <a> na strani, vrši se ekstrakcija odredišne strane uz primjenu određenih logičkih pravila kojima se osigurava da se testiraju samo one strane koje ulaze u opseg inicijalnog zahtjeva.

Određeni testovi sigurnosti mogu biti djelimično završeni u vrijeme samog parsiranja. To je slučaj kod validacije serverskog digitalnog sertifikata, gdje se već u inicijalnoj komunikaciji između sistema i testirane aplikacije (u cilju parsiranja) određenim programskim tehnikama može pouzdano utvrditi nepostojanje validnog digitalnog sertifikata. Ukoliko je odabran test validnosti sertifikata, ovakav rezultat se čuva u bazi kao sigurnosni propust. U suprotnom, ovaj događaj se zanemaruje.

Kod parsiranja se koristi isključivo GET zahtjev da bi se dobio sadržaj aplikacije, dokkod izvršavanja testova korištena metoda zavisi od samog načina izvođenja testa. Za slanje HTTP zahtjeva odgovoran je eksterni segment modula za parsiranje, čije funkcije koriste svi moduli.

6.1.2 Dio sistema za integraciju modula za testiranje

Osnovna uloga ovog segmenta sistema je uspostavljanje načina integracije svih budućih modula za implementaciju procedura za testiranje. Ne sadrži konkretne procedure, već specificira apstraktne pozive i interfejse koje naknadno razvijeni moduli moraju implementirati. Istovremeno, ovaj segment objavljuje funkcije ostalih segmenata sistema (od kojih su neki i opisani u prethodnojsekciji) sa kojima će moduli za testiranje

morati ostvariti interakciju u cilju komunikacije sa ostatkom sistema. Prethodno opisani modul jedna je od takvih funkcija, jer je parsiranje sadržaja invarijantno u odnosu na tip testa koji se izvodi, a razvojnom timu je na raspolaganju ostavljen izbor potrebnih elemenata od skupa svih elemenata dobijenih parsiranjem. Funkcije koje su zadužene za čuvanje rezultata su još jedan primjer, jer aplikacija koristi jedinstveno skladište podataka kojem svi novimoduli moraju pristupati uniformno.

U cilju interoperabilnosti različitih modula, ovaj integracioni segment sistema ima još jednu vrlo značajnu funkciju. Naime, na ovom mjestu su definisani i nivoi ozbiljnosti sigurnosnih propusta koje aplikacija podržava. Postoji 5 ovakvih nivoa, koji su definisani prema preporukama Američkog instituta za standarde i tehnologiju (NIST). Međutim, s obzirom da u trenutku definisanja spomenutih nivoa nije moguće predvidjeti broj i tipove dodatnih modula uključenih u svaki sistem, klasifikacija otkrivenih propusta u neku od definisanih grupa je prepuštena razvojnom timu, u fazi projektovanja modula. Definisanjem jedinstvenih nivoa za čitav sistem se teži formiranju globalne baze podataka koja će (posebno u akademskim primjenama) pružiti uvid u globalni nivo sigurnosti web aplikacija i informacionih sistema u cjelini. Naravno, i dalje postoji problem interoperabilnosti sa drugim (akademskim ili komercijalnim) sistemima iste namjene koji definišu sopstvene metode klasifikacije rezultata.

6.1.3 Moduli za izvršavanje testova

Kako je već navedeno u nekoliko navrata u prethodnim poglavljima, nad ovim dijelom sistema centralni segmenti imaju najmanju kontrolu. Ono čega se svaki modul mora pridržavati je način komunikacije sa ostatkom sistema, bilo da se radi o preuzimanju rezultata iz prethodnih koraka, slanje podataka o trenutnom toku testa (procenat izvršenih testova) ili skladištenju podataka. Ovo posljednje je od posebnog značaja, jer korisnik koji inicira testiranje ima uvid u dobijene podatke tek nakon završenog testiranja, dok su mu u toku izvršavanja dostupni samo podaci o osnovnom statusu i trajanju testa od njegovog iniciranja. S obzirom na propisanu strukturu skladišta rezultata, od razvojnog tima konkretnih modula u određenoj mjeri zavisi čak i upotrebna vrijednost rezultata (polja predviđena za opis dobijenih rezultata, čiji sadržaj nije striktno definisan itd).

6.1.4 Modul za čuvanje rezultata

Sistem pruža dva načina čuvanja rezultata dobijenih testiranjem, pri čemu svaki od njih ima određene prednosti i nedostatke:

– čuvanje rezultata u okviru XML datoteka.Prednosti korištenja XML datoteka u domenu interoperabilnosti je nedvojbena, pa je to ujedno i najveća prednost ovakvog načina skladištenja rezultata. Ujedno, korištenje jedne ovakve tehnologije je u skladu sa principima kojih se pridržava kompletan sistem. S druge strane, kompleksnost kasnije obrade ovakvih rezultata je nešto što može u određenoj mjeri biti i ograničavajući faktor u daljem radu.

– čuvanje rezultata u bazi podataka.U situacijama sa velikim količinama podataka koje je potrebno efikasno analizirati, mogućnost čuvanja rezultata u bazi podataka pokazuje svoje prave prednosti. Neke od tih prednosti su upravo i iskorištene u klijentskom dijelu sistema za prikaz globalne statistike i praćenje rada kompletnog sistema. S druge strane, izbor sistema za upravljanje bazom podataka (SUBP) donosi određena ograničenja u pogledu okruženja za implementaciju. Kao standardni SUBP je odabran MySQL, kako zbog svoje cijene, tako i zbog najvećeg broja podržanih platformi, čime se umanjuju efekti prethodno pomenutihograničenja.

Sam sistem ne ograničava niti nameće upotrebu jednog ili drugog načina čuvanja rezultata, pa tako većina razvijenih modula koristi oba načina, prepuštajući administratoru izbor skladišta za analizu podataka.

6.1.5 Interfejs prema eksternom sistemu

Sistem za testiranje web aplikacija ima dva osnovna interfejsa za komunikaciju sa eksternim sistemima i krajnjim korisnicima. Jedan interfejs podrazumijeva konzolni pristup sistemu, sa procedurama koje izvršavaju sljedeće korake:

– obrada primljenog zahtjeva,
– pokretanje postupka parsiranja i pripreme aplikacije,
– za svaki od odabranih testova vrši kreiranje paralelne programske niti koja je odgovorna za izvršavanje testa.

Drugi interfejspodrazumijeva komunikaciju SOAP porukama, odnosno pristup sistemu kao web servisu. Kad se radi o načinima pristupa, zbog svoje prirode i mogućnosti distribucije, pristup putem web servisa nudi jednu dodatnu funkciju. Osim pokretanja testova, preko servisa je moguće izvršiti dodjelu određenog testa nekom od korisničkih naloga definisanih na sistemu. Ova funkcionalnost se izvršava u kombinaciji sa klijentskim dijelom sistema, pa se koristi i segment baze podataka odgovoran za administraciju korisničkih naloga – ili druge slične aktivnosti koje originalno nisu podržane od strane centralnog dijela sistema.

6.2 *Klijentski dio sistema*

Zahtjevi korisnika se obrađuju prije aktiviranja potrebnih modula i iniciranja. Sistem predviđa posebnu sintaksu kojom se specificiraju testovi koje je potrebno provesti nad zadatom aplikacijom. Funkcije u jezgru sistema (core dijelu sistema) ne pružaju ograničenja u pogledu odredišnih adresa, postojećih korisničkih naloga na sistemu i privilegija korisničkih naloga, već se za te funkcionalnosti oslanjaju na klijentske dijelove sistema. Core aplikacija obrađuje sve pravilno formirane zahtjeve od strane korisnika. Nakon završetka obrade, korisnik na raspolaganju ima sav sadržaj parsirane odredišne aplikacije, kao i rezultate završenih testova. Ovo je vrlo značajan segment rada sistema, na koji se mora obratiti posebna pažnja. Mogućnosti otkrivanja sigurnosnih propusta bilo koje web aplikacije (ili objavljivanja podataka o otkrivenim propustima) otvara mogući-

nost ugožavanja sigurnosti od strane korisnika koji nemaju potrebno tehničko znanje za izvršavanje ovakvih napada. Iz tog razloga je od velikog značaja postojanje mogućnosti ograničavanja mogućnosti testiranja po tipu testova i po lokaciji/nazivu testirane aplikacije. U protivnom, rad ovakvog sistema bi mogao imati efekat suprotan očekivanom (povećavanje broja potencijalnih napadača, sa alatom koji će im dati uvid u ranjivosti bilo koje aplikacije).

Funkcije koje klijentski dio sistema treba da podrži se mogu podijeliti na dvije osnovne grupe, prema tipu korisnika. Korisniku koji ne pripada administrativnoj grupi su dostupne sljedeće funkcije:

– pregled trenutno aktivnih testova i njihovog statusa – za trenutno aktivne testove se iz baze podataka vrši provjeravanje trenutnog statusa u intervalu od dvije sekunde. S obzirom da se radi o velikom broju upita, za ovu svrhu se koristi Connection Pool u kombinaciji sa AJAX tehnologijom, te se tako omogućava osvježavanje samo onog dijela ekrana koji je potrebno osvježiti. Nakon završetka aktivnog testa, on se briše sa spiska i prebacuje u grupu završenih testova,
– pregled završenih testova – kod pregleda završenih testova nema periodične provjere u bazi podataka, jer više ne može doći do promjene podataka vezanih za ove testove. Grupa završenih testova je obično znatno veća od liste aktivnih testova i zauzima znatno veći dio površine ekrana. Svaki od unosa u tabeli sadrži referencu prema strani na kojoj je moguće detaljnije analizirati rezultate dobijene datim testom,
– pregled statističkih podataka dobijenih iz završenih testova – postoji nekoliko načina na koje je moguće analizirati statističke podatke. Na početnoj strani nalazi se nekoliko grafikona koji analiziraju rezultate u određenim segmentima rada aplikacije. Osim toga, implementiran je i poseban dio aplikacije koji sadrži sumarni pregled rezultata za kompletan sistem. Ovaj drugi statistički prikaz analizira učestalost pojave aplikacija ranjivih na pojedine napade, bez obzira o kojim aplikacijama se u konkretnom slučaju radi,
– iniciranje novih testova – ovaj slučaj upotrebe uključuje komunikaciju sa web servisom putem kojeg su objavljene funkcije centralnog dijela sistema. Kroz klijentsku aplikaciju je isključivo moguće izvršiti dodjelu određenog testa nekom od korisničkih naloga. Direktno pokretanje testova je dostupno samo putem konzolnog interfejsa, kako je već ranije navedeno.
– iniciranje sinhronizacije – web servis za sinhronizaciju je modul koji omogućava proširenje baze rezultata na osnovu podataka dobijenih od strane drugog korisnika. Modul za sinhronizaciju je detaljnije opisan u sekciji 6.3, dok je ovdje potrebno naglasiti da se sinhronizacija može vršiti isključivo uz poštovanje striktno definisanih sigurnosnih ograničenja.

Administratorima su, osim svih funkcija dostupnih običnim korisnicima, dostupne i sljedeće funkcionalnosti:

– pregled rezultata testova drugih korisnika – ova funkcionalnost je implementirana sa namjerom kontrole rada ostalih korisnika od strane administratora. Provjerom pokrenutih testova se stiče uvid u eventualne nedozvoljene zahtjeve od strane korisnika, koji najčešće mogu biti posljedica neispravno konfigurisanih korisničkih naloga,
– administracija korisničkih naloga – korisnički nalozi su definisani sa posebnim osvrtom na sigurnost i kontrolu rada aplikacije od strane administratora. Administrator je u stanju da definiše opseg adresa sa kojih je novom korisniku dozvoljen pristup aplikaciji, opseg adresa na kojem je dozvoljeno testiranje za neki nalog, kao i strane unutar aplikacije kojima je dozvoljen pristup. Svaki pokušaj neispravnog korištenja aplikacije se čuva u bazi podataka.

Ranije su navedena dva načina komunikacije ostalih dijelova sistema sa *core* dijelom. Klijentski dio koristi isključivo pristup putem web servisa, jer je takav način komunikacije znatno jednostavniji i fleksibilniji. Na ovaj način ostavljena je mogućnost distribucije servisa u okruženjima u kojima bi sistem radio sa velikim opterećenjem.

6.3 *Modul za sinhronizaciju resursa*

Modul za sinhronizaciju resursa je najmanje značajan modul za rad kompletnog sistema, te se njegovim izostavljanjem iz implementacije ne gube osnovne funkcije koje sistem nudi. Ipak, postojanje ovakvog modula i njegovo ispravno korištenje doprinosi efikasnijem radu sistema. Modul je dizajniran tako da omogući provjeru baze podataka na drugim, njemu poznatim instancama sistema, te da odgovarajuće rezultate prenese u lokalnu bazu. Pod rezultatima se podrazumijevaju resursi za izvršavanje testova, odnosno sadržaj konfiguracionih datoteka koje se koriste u procesu testiranja. Za ispravno funkcionisanje modula, potrebno je nekoliko preduslova:

– pristup lokalnim podacima.Da bi se izbjeglo ponavljanje sadržaja konfiguracionih datoteka, servis mora imati pristup lokalnoj bazi podataka – ako radi u klijentskom modu onda za filtriranje sadržaja konfiguracionih datoteka, a ako radi u serverskom modu onda za slanje traženih sadržaja prema klijentima.
– pristup klijentskoj/serverskoj strani putem mreže.Ovo je osnovni zahtjev za funkcionisanje ovog dijela sistema, jer bez pristupa mrežnim resursima nema načina za razmjenu podataka između različitih servisa, što je njegova jedina funkcija.
– pristup bazi podataka.Pristup bazi je bitan zbog logova. U slučaju da proces sinhronizacije ne funkcioniše ispravno moguće je vidjeti razlog ovakvog ponašanja aplikacije, kao i sadržaje dobijene od ostalih učesnika u komunikaciji, tj. od udaljenih sistema.

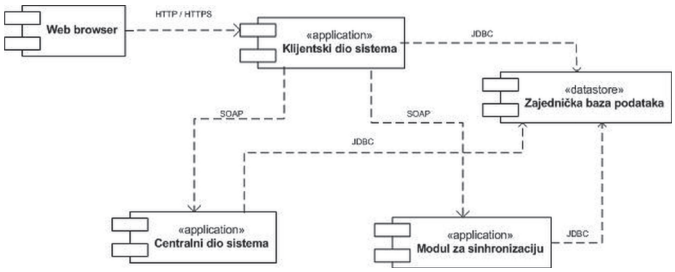
Servisu na raspolaganju stoje dva načina rada. Jedan je serverski, u kojem servis odgovara na svaki zahtjev upućen od strane klijentskih instanci.Drugi je klijentski, u kojem servis radi isključivo na ažuriranju sopstvene baze putem serverskih instanci koji su mu na raspolaganju. Dva moda rada nisu međusobno isključivi, i omogućen je istovremeni rad modula u oba moda.

Parametri za funkcionisanje se navode u bazi podataka i uključuju sljedeće konfiguracije:

- definisanje načina rada,
- definisanje servera u klijentskom načinu rada,
- definisanje klijenata sa kojima je dozvoljena komunikacija,
- definisanje tipa resursa koji će se ažurirati,
- definisanje lokacija konfiguracionih datoteka za pojedine module za testiranje.

Modul za sinhronizaciju komunicira sa svim ostalim modulima sistema. Sa klijentskim dijelom se komunikacija odvija indirektno – putem baze podataka, jer se kroz klijentsku aplikaciju vrši iniciranje procesa sinhronizacije i konfiguracija parametara za rad ovog modula. Sa centralnim dijelom sistema nema direktne komunikacije, ali se vrši sinhronizovanje resursa koji su direktno odgovorni za rad i funkcionalnost centralnog dijela sistema.

Komunikacija između servisa za sinhronizaciju i klijentske aplikacije se odvija putem TCP poruka, posebno definisanim protokolom aplikativnog nivoa. U prvom planu je jednostavnost komunikacije između modula za sinhronizaciju i ostatka sistema, te se ona zasniva na porukama sa malim zaglavljem, čime se nastojala postići veća efikasnost komunikacije. Modul za sinhronizaciju ne pruža posebne potvrde klijentskoj aplikaciji o uspješno ili neuspješno završenoj sinhronizaciji, već se podaci o radu čuvaju u za to posebno podređenim tabelama unutar baze podataka.



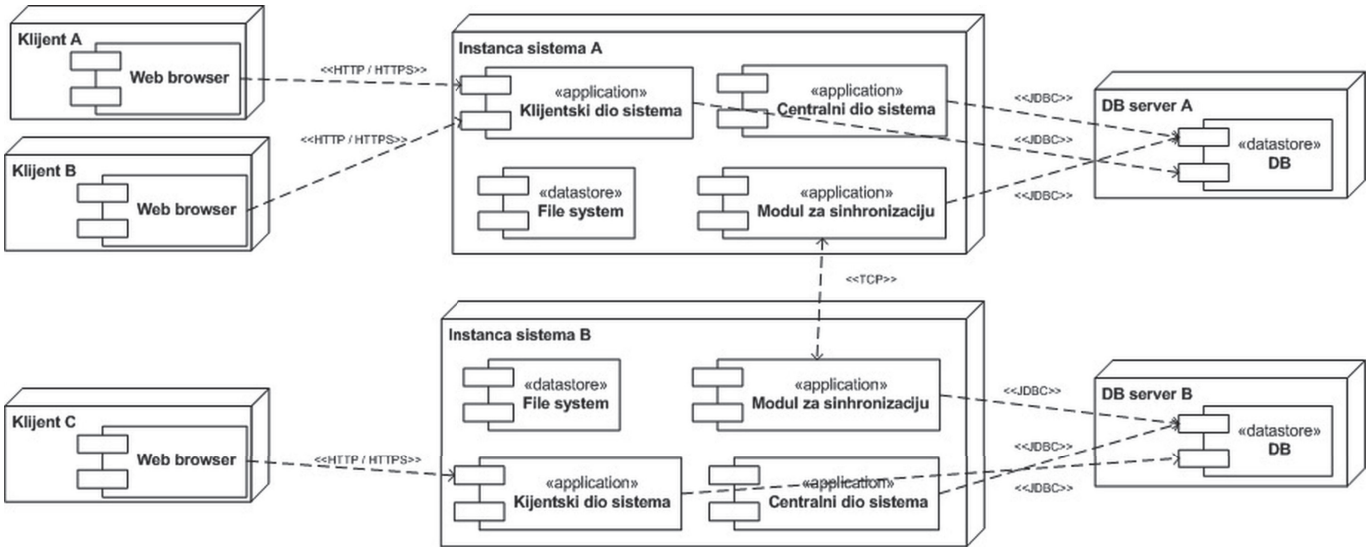
Slika 12. – *Dijagram komponenti.*

Razvijeni sistem, u domenu konačne implementacije, ima veoma jednostavnu arhitekturu (slika 12). Kako je već rečeno u poglavlju o specifikaciji softverskih zahtjeva, hardverski zahtjevi su minimalni. Naravno, sa značajnijim porastom broja korisnika moguće su izmjene u planiranim resursima, ali je to stavka koja se ne može precizno isplanirati u inicijalnoj implementaciji, već zavisi od niza drugih faktora (slika 13).

7. ZAKLJUČAK

Web aplikacije su postale veoma atraktivni ciljevi različitih vrsta napada, jer rezultati narušavanja sigurnosti web aplikacija mogu donijeti veliku finansijsku korist za napadača. Iz tog razloga sigurnost web aplikacija je postala veoma važan segment pri njihovom projektovanju i implementaciji.

U radu je analiziran problem sigurnosti web aplikacija, opisana je arhitektura web aplikacija i ukazano je na specifičnost problema sigurnosti web aplikacija i potrebu njihove zaštite na aplikativnom nivou. Detaljnije su opisani najzastupljeniji napadi na web aplikacije, SQL injection i XSS. U radu je dat i pregled tipova testova sigurnosti web aplikacija, kao i pregled postojećih rješenja. U ovom radu predstavljen je WASTT (Web Application Security Testing Tool) – softversko rješenje za otkrivanje i klasifikaciju sigurnosnih propusta u web aplikaci-



Slika 13. – *Dijagram raspoređenosti.*

jama.Zahtjevi koje WASTT rješenje treba da zadovolji kreirani su na bazi prednosti i nedostataka postojećih rješenja, kako besplatnih, tako i komercijalnih.

REFERENCE

[1] http://www.cenzic.com/downloads/Cenzic_AppsecTrends_Q3-Q4-2009.pdf, 2010, posljednja posjeta 08.05.2010.
[2] Session tracking on the web, V. Raghvendra, Internetworking, 3(1), March 2000.
[3] Automatic Creation of SQL Injection and Cross-Site Scripting Attacks, Adam Kiezun, Philip J. Guo, KarthickJayaraman, Michael D. Ernst, Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, May 2009,
[4] Dynamic pharming attacks and locked same-origin policies for web browsers, Chris Karlof, Umesh Shankar, J. D. Tygar, David Wagner, CCS '07: Proceedings of the 14th ACM conference on Computer and communications security, October 2007,
[5] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli.Fundamentals of Software Engineering.Prentice-Hall International, 1994.
[6] Paolo Tonella and FilippoRicca.A 2-Layer Model for the White-Box Testing of Web Applications.In IEEE International Workshop on Web Site Evolution (WSE), 2004.
[7] http://www.parosproxy.org, posljednja posjeta 08.05.2010.
[8] http://www.nist.gov/index.html,posljednja posjeta 08.05.2010.
[9] http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project, posljednja posjeta 08.05.2010.
[10] http://yehg.net/lab/pr0js/training/webscarab.php, posljednja posjeta 08.05.2010.
[11] http://www-01.ibm.com/software/awdtools/appscan/, posljednja posjeta 08.05.2010.
[12] http://www.nessus.org/documentation/, posljednja posjeta 08.05.2010.

Ognjen Joldžić, dipl.ing., Elektrotehnički fakultet Banjaluka, RS, BiH
Kontakt: ognjen.joldzic (at) gmail.com
Oblasti interesovanja: sigurnost, kriptografija, PKI, objektno-orijentisano programiranje i modelovanje, Internet programiranje, XML-bazirana međuoperativnost, Web servisi, računarske mreže

doc. dr Zoran Đurić, Elektrotehnički fakultet Banjaluka, RS, BiH
Kontakt: zoran.djuric (at) etfbl.net
Oblasti interesovanja: sigurnost, kriptografija, PKI, platni sistemi i protokoli, formalna verifikacija, objektno-orijentisano programiranje i modelovanje, Internet programiranje