STASEC – ALAT ZA OTKRIVANJE SIGURNOSNIH PROPUSTA STATIČKOM ANALIZOM JAVA IZVORNOG KODA
STASEC – TOOL FOR DETECTION OF SECURITY VULNERABILITIES USING STATIC ANALYSIS OF JAVA SOURCE CODE

Zoran Đurić *Elektrotehnički fakultetBanja Luka*
Dijana Vuković, *Univerzitetski računarski centar Banja Luka*

Sadržaj - *U ovom radu se analizira problem sigurnosti web aplikacija i ukazuje se na različite vrste napada na web aplikacije. Kao najznačajnije vrste napada na web aplikacije, detaljno su analizirani SQL (Structured Query Language) Injection i XSS (Cross-site scripting). Opisana je i tehnika detekcije ranjivosti web aplikacije statičkom analizom njenog izvornog koda. U radu je predstavljen STASEC - alat za otkrivanje sigurnosnih propusta statičkom analizom Java izvornog koda.*

Abstract – *This paper presents the problem of Web application security and points out the different types of attacks on Web applications. As the most important types of attacks on Web applications, the SQL (Structured Query Language) Injection and XSS (Cross-site scripting) attacks are analyzed in detail. Also, the technique for vulnerability detection in Web applications using source code static analysis is described. This paper gives an overview of STASEC - a tool for detecting security vulnerabilities using static analysis of Java source code.*

1. UVOD

Ranjivost web aplikacija na potencijalne sigurnosne napade, u vrijeme kada se web bazirani informacioni sistemi sve više koriste kao podrška poslovanju, predstavlja veliki problem. Kao jedan od primarnih faktora u dizajnu i implementaciji samih aplikacija javlja se sigurnost web aplikacija.

Sigurnosni propusti u aplikacijama mogu se detektovati na dva načina: statičkom analizom izvornog koda i dinamičkom analizom. Pod statičkom analizom izvornog koda podrazumijeva se analiziranje izvornog koda aplikacije prije njenog puštanja u rad. Nasuprot statičkoj analizi izvornog koda, dinamička analiza se obavlja nakon puštanja same aplikacije u rad. Često se, radi detektovanja sigurnosnih propusta sa što većim stepenom tačnosti, koristi kombinacija obe vrste analize, pri čemu rezultati statičke analize izvornog koda pomažu prilikom dinamičke analize.

Web aplikacije obrađuju klijentske zahtjeve, koji obično dolaze putem web čitača. Često obrada zahtjeva podrazumijeva generisanje odgovarajućih upita i njihovo slanje ka serveru baze podataka u cilju izvršavanja implementirane poslovne logike. Shodno tome, web aplikacije su najranjivije na ulazne podatke koji se prosljeđuju aplikaciji putem HTML (*Hyper Text Markup Language*) formi ili direktno kroz URL (*Uniform Resource Locator*) adresu. Neadekvatna validacija ulaznih podataka može dovesti do gubitka ili izmjene povjerljivih informacija.

Statičkom analizom izvornog koda moguće je otkriti potencijalne sigurnosne propuste prije samog puštanja aplikacije u rad i otkloniti ih pri samom procesu implementacije. Najčešći napadi na web aplikacije prema OWASP-u [1] su SQL Injection i Cross-site Scripting (XSS) i relativno ih je jednostavno detektovati statičkom analizom izvornog koda korišćenjem OWASP-ovih smjernica za reviziju koda [2].

U poglavlju 2 opisana je tehnika detekcije ranjivosti web aplikacije statičkom analizom njenog izvornog koda. Poglavlje 3 opisuje napade na web aplikacije sa posebnim osvrtom na dva najčešće korišćena napada – SQL Injection i XSS, i daje smjernice za njihovu detekciju korišćenjem statičke analize izvornog koda. U poglavlju 4 predstavljen je STASEC - alat za otkrivanje sigurnosnih propusta statičkom analizom Java izvornog koda. Na kraju rada dat je zaključak.

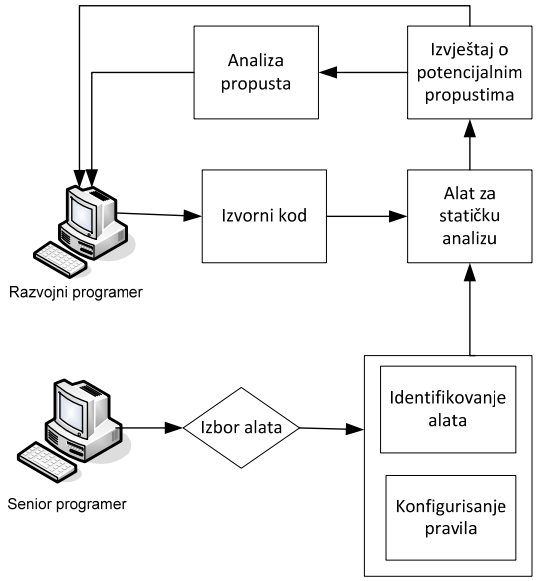
2. STATIČKA ANALIZA IZVORNOG KODA

Pod statičkom analizom izvornog koda podrazumijeva se testiranje aplikacija na različite vrste ranjivosti, analizom izvornog koda. Najčešće korišćene metode statičke analize su: sintaksna analiza (*syntactic analysis*) i analiza toka podataka (*data-flow analysis)*. Sintaksna analiza se sastoji od poređenja ulaznog koda sa predefinisanim uzorcima i za njeno korišćenje potrebno je poznavati sintaksu samog programskog jezika čiji se kod testira. Analiza toka podataka podrazumijeva praćenje vrijednosti podataka u određenom izvršnom segmentu programa ili metode.

Proces razvoja aplikacije uz statičku analizu izvornog koda odvija se u nekoliko koraka – razvojni programer piše kod, nakon završetka prosljeđuje ga alatu za statičku analizu koji je prethodno izabran kao najpogodniji i konfigurisan da provjerava odgovarajuća pravila. Nakon izvršene statičke analize, generiše se izvještaj o potencijalnim propustima. Razvojni programer učestvuje u analiziranju propusta, otklanja pronađene propuste i, nakon toga, proces se ponavlja slanjem korigovanog izvornog koda alatu za statičku analizu (slika 1).

Jedna od dobrih stra na statičke analize leži u činjenici da su unaprijed poznate instrukcije koje softver može da izvrši, te ne postoji potreba za nagađanjem ili interpretacijom ponašanja pojedinih instrukcija. Poznavanje programskog jezika u kojem je pisan programski kod olakšava analizu svih mogućih ishoda izvršavanja instrukcija, kao i pronalaženje potencijalnih sigurnosnih propusta sintaksnom analizom. Pored toga, statička analiza smanjuje vrijeme potrebno za otklanjanje propusta, jer se može tačno odrediti na kojem mjestu u kodu postoji sigurnosni propust, i direktno utiče na poboljšanje kvaliteta i pouzdanosti razvijenog softvera.

Kao glavni nedostatak statičke analize može se istaći potreba za pristupom izvornom kodu same aplikacije. Kao nedostatak može se istaći i to što se statičkom analizom izvornog koda aplikacije ne pronalaze problemi vezani za samo okruženje u kojem se aplikacija izvršava.



Slika 1. Proces razvoja aplikacije uz korištenje statičke analize

3. NAPADI NA WEB APLIKACIJE

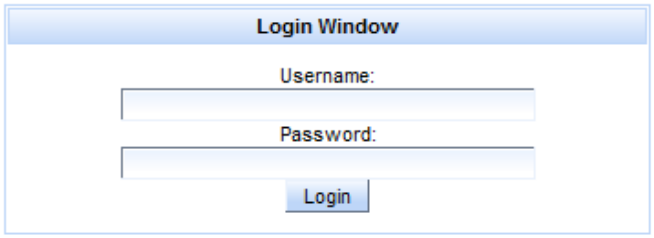
Web aplikacije su obično softverski proizvodi razvijeni za specifične klijente i mogu da sadrže određene propuste, koji se javljaju kao posljedica neadekvatnog razvoja softvera. Kao takve, ranjive su na različite vrste napada. Potencijalno slabe tačke okruženja u kojem se izvršavaju web aplikacije, a koje napadači koriste pri napadu su: web klijenti (izvršavanje malicioznog koda, ranjivost web čitača), transportni nivo (prisluškivanje HTTP (*Hyper Text Transfer Protocol*) komunikacije, SSL (*Secure Socket Layer*) redirekcija, *Man in the Middle* napadi), web/aplikativni serveri, web aplikacije i baza podataka (neautorizovano izvršavanje komandi kroz upite koji se šalju bazi, manipulacija upitima za neautorizovan pristup podacima, interakcija s operativnim sistemom). Najveću prijetnju web aplikacijama predstavljaju propusti koji se pojavljuju u samom procesu razvoja koda. Korišćenjem tehnika statičke analize izvornog koda, potencijane propuste moguće je otkloniti prije puštanja u rad same aplikacije i time smanjiti mogućnost izvršavanja potencijalnih napada.

Većina napada na Web aplikacije prouzrokovana je neadekvatnom provjerom ulaznih podataka. Prema OWASP-ovoj listi [1] u 10 najvećih prijetnji Web aplikacijama spadaju: *Injection*, XSS, propusti u upravljanju autentikacijom i sesijama (*Broken Authentication and Session Management*), nesigurno direktno referenciranje objekata (*Insecure Direct Object References*), CSRF (*Cross Site Request Forgery*), sigurnosni propusti u konfiguracijama (*Security Misconfiguration*), propusti u restrikciji URL pristupa *(Failure to Restrict URL Access*), redirekcija i prosljeđivanje bez validacije (*Unvalidated Redirects and Forwards*), nesigurna skladišta podataka sa stanovišta kriptografije (*Insecure Cryptographic Storage*) i nedovoljna zaštita podataka na transportnom nivou (*Insufficient Transport Layer Protection*).

3.1. SQL INJECTION

SQL Injection je vrsta napad u kojem napadač unosi SQL (*Structured Query Language*) kod putem web formi u cilju izmjene semantike originalnog SQL upita. Rezultati izvršavanja SQL Injection napada mogu biti različiti, od zaobilaska autentikacije, do izmjene podataka u bazi podataka koju aplikacija koristi, ili drugim bazama podataka koje se nalaze na istom sistemu za upravljanje bazama podataka.

Nakon uspješno izvršenog SQL Injection napada, napadač može da očita osjetljive podatke iz baze, izmijeni podatke u bazi korišćenjem osnovnih komandi za unos, modifikiovanje i brisanje – INSERT, UPDATE i DELETE, izvrši administratorske poslove (kao što su gašenje servera baze podataka, brisanje pojedinih tabela i sl.) ili čak izvrši određene komande nad operativnim sistemom na kojem radi server baze podataka (npr. preuzimanje *passwd* datoteke u kojoj su smješteni podaci o korisnicima koji imaju mogućnost prijave na sistem, na Linux operativnim sistemima). Jasno je da je za uspješno izvršavanje pojedinih malicioznih radnji neophodno da korisnik pod kojim aplikacija pristupa bazi podataka ima za to odgovarajuće privilegije.



Slika 2. Forma za prijavu korisnika

Nakon što korisnik unese pristupne podatke u formu na slici 2, i inicira akciju klikom na dugme Login, prosljeđeni podaci ka serveru za autentikaciju provjeravaju se u bazi podataka validnih korisnika sistema. Ako se u ovu svrhu koristi neparametrizovani upit “SELECT * FROM users WHERE username=’“+request.getParameter(“username”)+“’ AND password=’“+ request.getParameter(“password”)+“’“, gdje se korisničko ime i lozinka očitani iz zahtjeva pomoću getParameter metode dodaju u upit jednostavnom konkatenacijom stringova.

Unošenjem sekvence ’ OR 1=1# (# predstavlja znak za početak komentara u MySQL-u) u odgovarajuće polje za unos korisničkog imena na formi za prijavu korisnika, i njenim prosljeđivanja na serversku stranu, SQL upit će dobiti oblik ’’SELECT * FROM users WHERE username=’’ OR 1=1#’ AND password=’’“. Izvršavanje ovog SQL upita, napadaču će omogućiti autentikaciju bez poznavanja korisničkog imena i lozinke (lozinka se neće ni provjeravati, jer je taj dio upita sada dio komentara), a u zavisnosti od poslovne logike same aplikacije, i eventualni uvid u informacije o drugim korisnicima.

Smjernice kojih bi se trebalo pridržavati pri implementaciji aplikacija koje nisu ranjive na SQL Injection su sljedeće:

- korišćenje parametrizovanih upita i parametrizovanih uskladištenih procedura;
- kreiranje korisničkog naloga za pristup bazi podataka sa minimalnim pravima i privilegijama;
- kreiranje i korišćenje adekvatnih validatora ulaznih podataka;
- adekvatna obrada izuzetaka.

Statičkom analizom izvornog koda korišćenjem tehnike sintaksne analize jednostavno se mogu otkriti sigurnosni propusti vezani za korišćenje neparametrizovanih upita (npr. provjerom da li se za kreiranje i izvršavanje SQL naredbi koristi *Statement* umjesto *PreparedStatement* interfejsa, u programskom jeziku Java) i neadekvatne obrade izuzetaka (npr. korišćenjem metode printStackTrace() u programskom jeziku Java za ispis generisanog izuzetka mogu se prikazati informacije kao što su naziv baze podataka ili naziv tabele u bazi, vrijednosti pojedinih parametara i sl.).

3.2. XSS

Druga najzastupljenija vrsta napada na web aplikacije je XSS. XSS se odnosi na različite napade u kojima napadač ubacuje maliciozni JavaScript kod u web aplikaciju [7] putem GET parametara ili web formi. 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 [8]. U ovakvoj situaciji, bez obzira sa koje stvarne lokacije se učitava maliciozni JavaScript, web čitač će ga smatrati legalnim jer je kod za poziv tog JavaScript-a ugrađen u „ranjivu“ web aplikaciju kojoj korisnik pristupa.

Uspješan XSS napad može da se ostvari u web aplikacijama koje koriste ulaz od strane korisnika kao dio izlaza koji generišu, bez prethodne validacije. Najčešće se XSS napadi obavljaju kroz stranice za pretraživanje, ali i putem forme za različite vrste unosa.

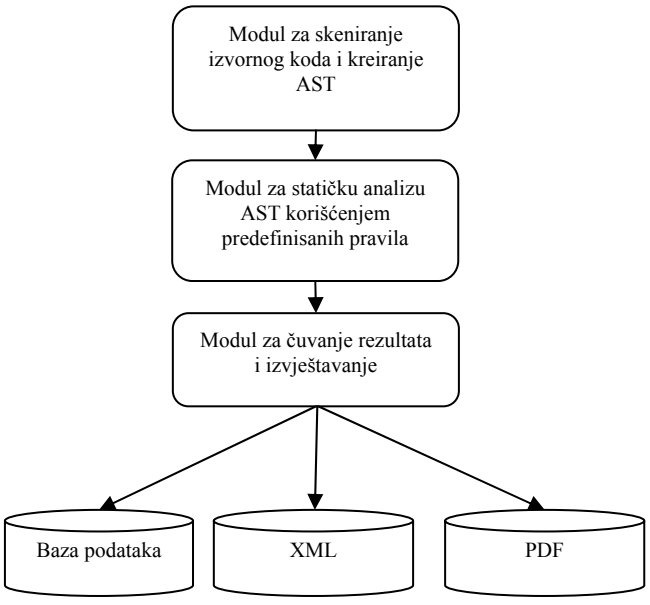
Neka za unesenu riječi *knjiga* putem forme za pretraživanje, web aplikacija generiše odgovor, ukoliko tražena riječ ne postoji – “Riječ *knjiga* nije pronađena”. Ukoliko se umjesto *knjiga* unese JavaScript kod *<script>alert(‘Pozdrav!’);</script>*, prilikom ispisa odgovora poslatog sa serverske strane prvo će se pojaviti prozor upozorenja na kojem će biti ispisano *Pozdrav!*. Umjesto jednostavnog skripta za prikaz prozora upozorenja, napadač može da izvrši XSS napad unošenjem *<script src=http://hacker.com/malicious.js></script>*, čijim izvršavanjem se može doći u posjed kolačića, identifikatora sesije i drugih osjetljivih informacija a u zavisnosti od funkcije JavaScript datoteke *malicous.js*.

Postoje tri tipa XSS napada: uskladišteni (*stored*), reflektovani (*reflected*) i DOM-bazirani XSS. Uskladišteni XSS napadi su oni kod kojih je ubačeni kod smješten na serverskoj strani, na primjer u bazi podataka. Web čitač svakog korisnika prima maliciozni skript od servera u trenutku kad zahtijeva uskladištene informacije. Reflektivni XSS napadi su napadi kod kojih je ubačeni kod (kao dio zahtjeva) reflektovan od strane Web servera (kao dio odgovora). DOM bazirani XSS (poznat i kao “type-0 XSS”) je XSS napad čije izvršavanje rezultuje modifikovanjem DOM okruženja u web čitaču korisnika korišćenjem originalnog skripta sa klijentske strane, tako da se klijentski kod izvršava na neočekivan način.

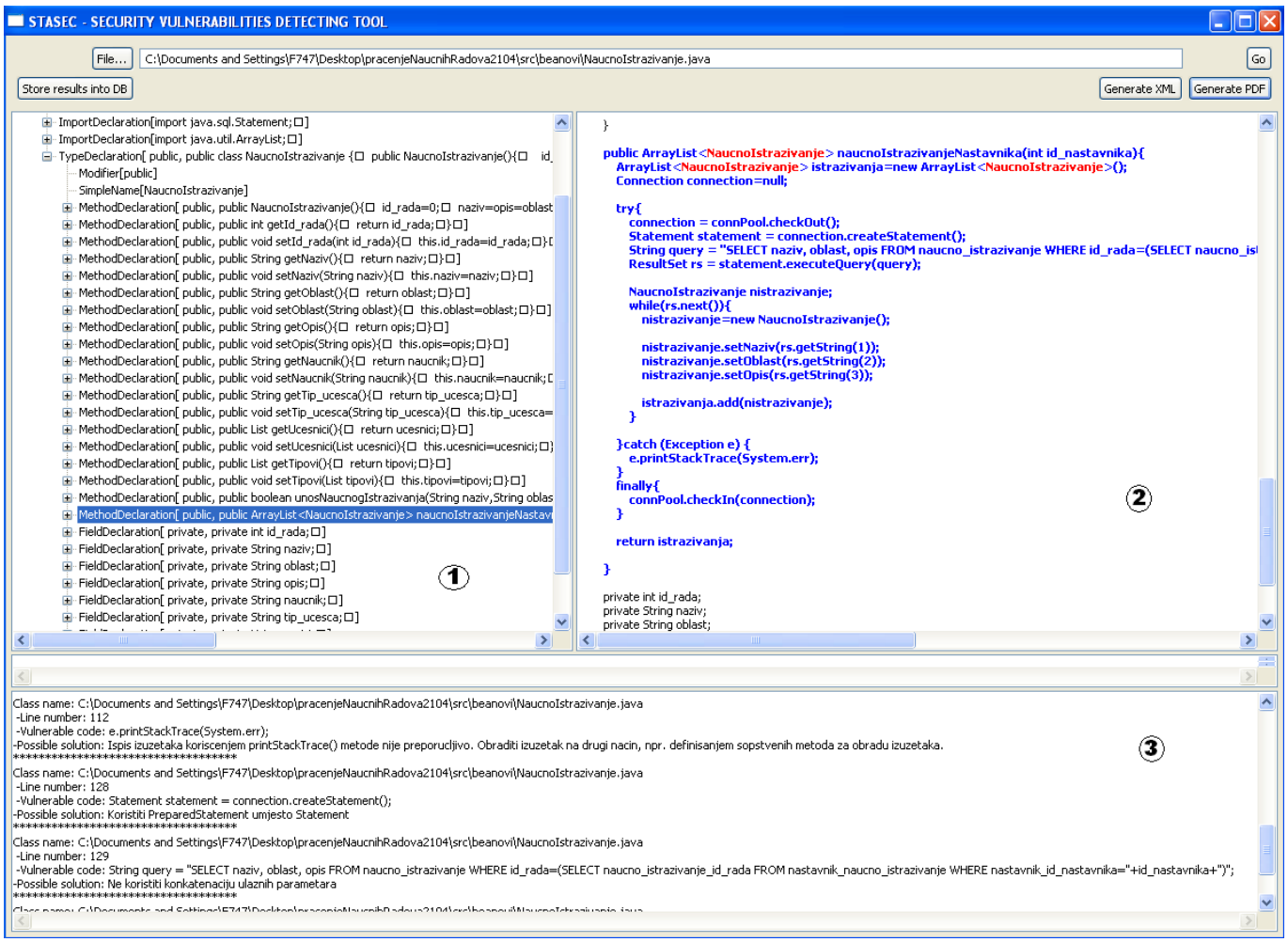
Osnovni problem zbog kog su XSS napadi uspješni na većini aplikacija je neadekvatna validacija ulaznih podataka. Najbolja zaštita od XSS napada postiže se kombinovanjem statičke analize izvornog koda, dinamičke analize i *penetration* testiranja.

4. STASEC – ALAT ZA STATIČKU ANALIZU JAVA IZVORNOG KODA

STASEC – alat za statičku analizu Java izvornog koda koristi metod sintaksne analize za skeniranje izvornog koda aplikacija pisanih u programskom jeziku Java u cilju detektovanja potencijalnih sigurnosnih propusta. Alat je zasnovan na prijedlogu alata [3] za otkrivanje sigurnosnih propusta web aplikacija statičkom analizom izvornog koda prikazanim na slici 3, sa dvije izmjene – STASEC ne posjeduje modul za provjeru kompleksnosti i pored čuvanja rezultata analize u bazi podataka i XML formatu, rezultate analize moguće je sačuvati i kao PDF dokument.



Slika 3. Prijedlog rješenja alata za statičku analizu izvornog koda



Slika 4. STASEC - alat za statičku analizu Java izvornog koda

Java programski jezik prema TIOBE-ovom indeksu [4] za januar 2010. godine predstavlja vodeći jezik pri razvoju web aplikacija. Java je objektno-orijentisani programski jezik opšte namjene pogodan za razvoj web aplikacija korišćenjem odgovarajućih tehnologija za razvoj web aplikacija kao što su JavaServer Pages (JSP) i JavaServer Faces (JSF). Kako je za statičku analizu izvornog koda korišćenjem metode sintakse analize potrebno poznavati sintaksu programskog jezika čiji izvorni kod se analizira, tako je i pri razvoju modula za statičku analizu u STASEC-u korišćena specifikacija programskog jezika Java – Java Language Specification (JLS).

STASEC (slika 4) je desktop aplikacija razvijena u programskom jeziku Java, i sastoji se od sljedećih modula – modul za skeniranje izvornog koda i kreiranje AST, modula za statičku analizu AST korišćenjem predefinisanih pravila i modula za čuvanje rezultata i izvještavanje.

Modul za skeniranje izvornog koda i kreiranje AST koristi Java klasu *ASTParser*, iz Eclipse JDT (*Java Development Tools*) projekta, za kreiranje apstraktnog sintaksnog stabla Java klase čiji se izvorni kod analizira. Statičku analizu samog izvornog koda lakše je vršiti kada se sam kod predstavi u obliku apstraktnog sintaksnog stabla. Predstavljanjem izvornog koda u obliku apstraktnog sintaksnog stabla korišćenjem klase *ASTParser* može se obezbijediti i mogućnost direktnog modifikovanja koda u kojem su detektovani sigurnosni propusti. Pri implementaciji prvog modula korišćena je aplikacija za kreiranje AST-a opisana u [5]. AST skenirane aplikacije prikazano je na slici 4, oznaka 1.

Drugi modul, modul za statičku analizu AST korišćenjem predefinisanih pravila pruža mogućnost statičke analize izvornog koda u cilju pronalaženja potencijalnih sigurnosnih propusta u web aplikaciji vezanih za SQL Injection napad. Modul koristi Java API [9] klase *Pattern* i *Matcher* kojima se, na osnovu OWASP-ovih smjernica za skeniranje Java izvornog koda [2], kreiraju odgovarajući regularni izrazi i koriste pri pretraživanju izvornog koda u potrazi za sigurnosnim propustima vezanim za SQL Injection napade. Alat analizira izvorni kod i otkriva ranjivosti na SQL Injection napade traženjem neparametrizovanih upita koji se prosljeđuju bazi podataka i neadekvatno obrađenih *SQLException* izuzetaka. Na slici 4, oznakom 2 predstavljen je izvorni kod skenirane aplikacije sa označenom metodom u kojoj su otkriveni sigurnosni propusti, dok su u dijelu prozora označenog oznakom 3 prikazani pronađeni propusti u formatu – linija koda u kojoj je sigunosni propust otkriven, ranjivi kod i moguće rješenje problema.

Treći modul, modul za čuvanje rezultata i izvještavanje pruža korisniku mogućnost smještanja rezultata analize, zajedno sa prijedlogom rješenja pojedinih propusta, u MySQL bazu podataka (slika 4, dugme *Store results into DB*), XML fajl (slika 4, dugme *Generate XML*) i pruža mogućnost eksportovanja rezultata u PDF dokument (slika 4,

dugme *Generate PDF*), korišćenjem iText biblioteke za kreiranje PDF dokumenata [6]. Pored izvještavanja, u ovom modulu korisniku je omogućeno da unese prijedloge rješenja pojedinih propusta, koja postaju dostupna za dalje analize.

Pri testiranju samog alata kreirana je pilot aplikacija *Praćenje naučnih radova* korišćenjem Java tehnologije za razvoj web aplikacija – JSP, čije su klase implementirane u cilju komunikacije sa bazom podataka ranjive na SQL Injection napade. Skeniranjem izvornog koda aplikacije koji se koristi pri komunikaciji sa bazom podataka (17 Java klasa) korišćenjem STASEC alata otkriveno je 80 sigurnosnih propusta vezanih za neparametrizovane upite i 51 sigurnosni propust vezan za neadekvatno obrađene izuzetke.

5. ZAKLJUČAK

Napadi na Web aplikacije mogu da prouzrokuju veliku štetu, kao što je finansijski gubitak ili gubitak osjetljivih informacija. Iz ovog razloga same aplikacije treba da budu dizajnirane i implementirane uzimajući u obzir različite sigurnosne prijetnje. Web aplikacije bi trebalo da budu otporne na različite vrste napada, poput *SQL Injection* i XSS napada. Ranjivost aplikacija na pojedine napade relativno jednostavno može biti otkrivena statičkom analizom izvornog koda, prije puštanja aplikacije u produkciju, primjenjujući odgovarajući alat za analizu.

Java programski jezik prema TIOBE-ovom indeksu za januar 2010. godine predstavlja vodeći jezik pri razvoju web aplikacija. STASEC pruža mogućnost statičke analiza Web aplikacija pisanih na programskom jeziku Java. Ova aplikacija posjeduje modul za testiranje aplikacije na ranjivost na SQL Injection napade.

Ovakav alat olakšava analizu aplikacija na sigurnosne propuste i pojednostavljuje njihovo otklanjanje, jer sam alat generiše prijedlog potencijalnih rješenja za otkrivene propuste. Alat pruža i mogućnost izmjene izvornog koda nakon izvršene analize, sa naznačenim mjestima na kojima su detektovani sigurnosni propusti.

LITERATURA

[1] Williams J., Wichers D., *The OWASP Top 10 -2010 Release Candidate*, The OWASP Foundation, November 2009.

[2] Van der Stock A., Lowery D., Rook D., Cruz D., Keary E., Williams J., Chapman J., Morana M. M., Prego P., *OWASP Code Review Guide V1.1*, The OWASP Foundation, November 2008.

[3] Vuković D., Đurić Z., *Otkrivanje sigurnosnih propusta web aplikacija statičkom analizom izvornog koda - prijedlog alata, YU INFO 2010,* rad prihvaćen za objavljivanje, januar 2010.

[4] TIOBE Programming Community Index for January 2010, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, posljednja posjeta februar 2010.

[5] Marques M., *Exploring Eclipse's ASTParser*, IBM, april 2005.

[6] iText, http://itextpdf.com/, posljednja posjeta februar 2010.

[7] Kiezun A., Guo J. P., Jayaraman K., Ernst D. M., *Automatic Creation of SQL Injection and Cross-Site Scripting Attacks,* Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, May 2009.

[8] Karlof C., Shankar U., Tygar J. D., Wagner D., *Dynamic pharming attacks and locked same-origin policies for web browsers*, CCS '07: Proceedings of the 14th ACM conferenceon Computer and communications security, October 2007.

[9] Java API, http://java.sun.com/javase/6/docs/api/, posljednja posjeta februar 2010.