SQLiTrainer - sistem za učenje o SQLi sigurnosnim propustima u aplikacijama

Đorđe Madić, Žarko Stanisavljević

***Apstrakt* — Sigurnosni propusti u aplikacijama koji nastaju prilikom njihovog razvoja i ostaju nedetektovani u produkcionom okruženju mogu dovesti do narušavanja integriteta, poverljivosti i dostupnosti takvih aplikacija. *SQLiTrainer* predstavlja skup ranjivih aplikacija kojima se mogu demonstrirati različite vrste *SQLi (eng. SQL injection)* ranjivosti. U radu je opisan način implementacije *SQLiTrainer* sistema i dati su primeri na koji način se sistem može iskoristiti za praktičnu obuku programera. Sistem je uspešno korišćen za izvođenje laboratorijskih vežbi na predmetu Zaštita računarskih sistema i mreža na Elektrotehničkom fakultetu u Beogradu.**

***Ključne reči* — SQLi, sigurnosni propusti, razvoj bezbednog softvera.**

I. Uvod

Razvoj bezbednog softvera podrazumeva postojanje svesti kod programera o potencijalnim problemima, a zatim i primenu čitavog seta dobrih praksi, kao i automatizovanih alata tokom procesa razvoja softvera. Aplikacije kod kojih postoje sigurnosne ranjivosti koje se mogu zloupotrebiti mogu dovesti do štete kako za korisnike takvih aplikacija, tako i za njihove autore.

Jedan od sigurnosnih propusta koji je često zastupljen u veb aplikacijama je *SQL injection* [1], kod koga se na različite načine na nepredviđen način mogu umetnuti naredbe koje mogu narušiti integritet, poverljivost i dostupnost baza podataka. Ovaj propust se već duži niz godina nalazi na top listama najčešćih sigurnosnih propusta u aplikacijama koje objavljuju organizacije kao što je *OWASP (Open Web Application Security Project)* [2].

U opštem slučaju nije jednostavno omogućiti programerima da kroz praktičan rad unaprede svoje znanje o ovakvim problemima, jer to podrazumeva izučavanje različitih mehanizama kojima se narušava informaciona bezbednost aplikacija na kojima se primenjuju. Primena ovih mehanizama kada se izvršavaju prema sistemima fizičkih i pravnih lica koja nisu upoznata i saglasna sa aktivnostima na proveri ranjivosti i testiranju upada u njihove sisteme je kažnjiva svuda u svetu (npr. u Srbiji prema Krivičnom zakoniku Republike Srbije (Članovi 298 do 304a)). Iz tog razloga postoje različiti sistemi koji omogućavaju svojim korisnicima da u zatvorenom okruženju na praktičan način obave obuku, a da ne prekrše zakon [3-5]. U ovom radu prikazan je jedan novi sistem za učenje o *SQLi* sigurnosnim propustima u aplikacijama.

Đorđe Madić radi u kompaniji Zuehlke Engineering, Bul. Milutina Milankovića 1i, 11070 Novi Beograd, Srbija (e-mail: djordje.madic@zuehlke.com).

U trenutku rada na ovom istraživanju Đorđe Madić je bio student master studija na Elektrotehničkom fakultetu, Univerziteta u Beogradu.

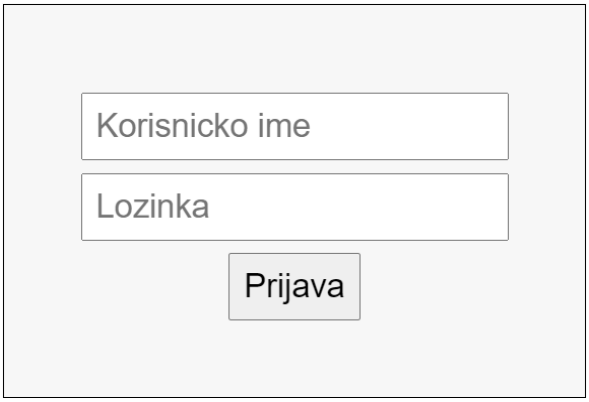
Žarko Stanisavljević radi na Elektrotehničkom fakultetu, Univerziteta u Beogradu, Bul. kralja Aleksandra 73, 11120 Beograd, Srbija (telefon: +381-11-3218-484; e-mail: zarko.stanisavljevic@etf.bg.ac.rs).

U drugom poglavlju su na primeru opisani uzrok i koraci kod izvođenja *SQLi* napada. U trećem poglavlju prikazan je način korišćenja realizovanog sistema na primeru jedne laboratorijske vežbe. U četvrtom poglavlju prikazana je implementacija laboratorijskih vežbi. U petom poglavlju dat je zaključak.

II. *SQL Injection*

OWASP definiše *injection* kao „slanje nepouzdanih podataka interpreteru kao deo komande ili zahteva“. *SQL injection* je tip napada koji se koristi za neovlašćeni pristup *SQL* bazi podataka koju aplikacija koristi. Napadač može čitati osetljive podatke, menjati njihovu strukturu, a u nekim slučajevima i izvršavati komande nad operativnim sistemom baze. Uzrok postojanja propusta je što aplikacija dozvoljava da korisnikov unos učestvuje u kreiranju *SQL* upita, omogućavajući mu da modifikuje originalni upit u svoju korist. U nastavku je na primeru *SQLi Login Bypass* napada objašnjen postupak izvršavanja napada.

SQLi Login Bypass počinje na stranici za prijavu korisnika, kao na Sl. 1. Cilj napada je, u bukvalnom prevodu sa engleskog, „zaobići prijavu“, odnosno prijaviti se ne znajući ni jedno korisničko ime ni lozinku.



Sl. 1. Stranica za prijavu korisnika

Prvi korak napada je analiza aplikacije. Ideja je pronaći ulazne podatke aplikacije koji se potencijalno koriste za kreiranje *SQL* upita. Ti podaci su kandidati da „nose“ napadački upit. U primeru prijave korisnika postoje dva ulazna podatka, korisničko ime i lozinka.

U narednom koraku potrebno je pretpostaviti kako izgleda *SQL* upit i odabrati ulazne podatke za napad. Na primer, upit za prijavu može biti sledeći:

```
SELECT * FROM korisnik
WHERE korisnicko_ime='$korisnicko_ime'
AND lozinka='$lozinka'
```

Delovi upita *$korisnicko_ime* i *$lozinka* su ulazni podaci sa korisničkog interfejsa, a napad se, na primer, može izvršiti kroz *$korisnicko_ime*.

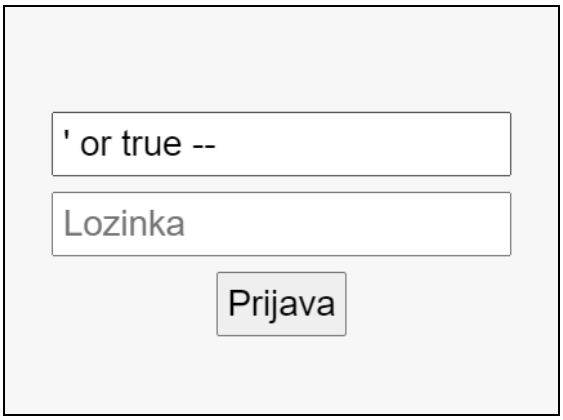
Napad se nastavlja na korisničkom interfejsu. U najčešćem slučaju tekstualni unos u polje za korisničko ime biće tretiran kao podatak, odnosno neće biti interpretiran kao komanda od strane baze podataka. Kako je korisničko ime tekstualnog tipa, u upitu se koristi apostrof za označavanje početka i kraja tekstualnog podatka. U slučaju da se u polje za korisničko ime unese apostrof, karakteri koji prethode tretiraće se kao podatak, dok će se oni koji slede tretirati kao komanda.

Koristeći ovu činjenicu napadač ima priliku da modifikuje originalni upit. Pod pretpostavkom da je prijava korisnika uspešna ukoliko upit vrati bar jedan rezultat, napad se može izvršiti kao na Sl. 2. što rezultuje sledećim upitom:

```
SELECT * FROM korisnik
WHERE korisnicko_ime='' or true --' and
password=''
```

Rezultat upita su svi redovi tabele „korisnik". Simbol „--" predstavlja oznaku za komentar čime se ignoriše deo upita nakon korisničkog imena.

Uslov „*or true*" čini da svaki red bude deo rezultata, ignorišući korisničko ime.



Sl. 2. *SQLi* napad na slučaju korišćenja prijave korisnika

Osnovu prevencije *SQLi* sigurnosnog propusta čine parametrizovani upiti (eng. *Parametrized Statement*, *Prepared Statement*). Oni omogućavaju da se bazi prvo prosledi šablon upita koji će se koristiti, a zatim za svako izvršavanje šablona i konkretni podaci. Korišćenje parametrizovanih upita garantuje da konkretni podaci neće biti interpretirani, čime se eliminiše *SQLi* sigurnosni propust. U nastavku je primer korišćenja parametrizovanog upita za pronalazak korisnika sa određenim korisničkim imenom u programskom jeziku *Java*:

```
String query = "SELECT * FROM korisnik WHERE
korisnicko_ime = ?";
PreparedStatement preparedStatement =
connection.prepareStatement(query);
preparedStatement.setString(1, "petar");
ResultSet results =
preparedStatement.executeQuery();
```

III. Primer korišćenja *SQLiTrainer* sistema

Primer korišćenja realizovanog sistema biće dat kroz prikaz jedne od laboratorijskih vežbi, dok će način upotrebe sistema u nastavi biti prikazan na primeru predmeta Zaštita računarskih sistema i mreža (ZRM) [6] na Elektrotehničkom fakultetu u Beogradu (ETF).

A. *Laboratorijska vežba Megatron*

Laboratorijska vežba *Megatron* zasniva se na aplikaciji koja predstavlja veb prodavnicu kompjuterske opreme. Vežba počinje na stranici za pretragu proizvoda. Pretraga se vrši po nazivu, gde je moguće uneti i samo deo naziva proizvoda. Rezultat pretrage prikazuje se u vidu tabele, sa kolonama za naziv i cenu proizvoda, kao na Sl. 3.

Sl. 3 Pretraga prozvoda

Cilj laboratorijske vežbe je pronaći korisničko ime i lozinku svih korisnika aplikacije. Boduju se i sledeće informacije:

- Naziv baze koju koristi aplikacija
- Verzija baze
- Nazivi tabela
- Nazivi kolona

Napad se izvršava kroz polje za pretragu proizvoda, dok se podaci koji su rezultat napada prikazuju u tabeli rezultata pretrage. Na primer, unosom sledećeg teksta u polje za pretragu dolazi se do naziva baze:

```
' and false union select 1, database() --
```

Sledećim unosom dolazi se do verzije baze:

```
' and false union select 1, h2version() --
```

Kako bi se došlo do korisničkog imena i lozinke svih korisnika aplikacije, prvo je potrebno pronaći naziv tabele koja sadrži korisnike. Sledećim unosom dolazi se do naziva svih tabela u bazi:

```
' and false union select 1, table_name from
information_schema.tables where
table_schema=database() --
```

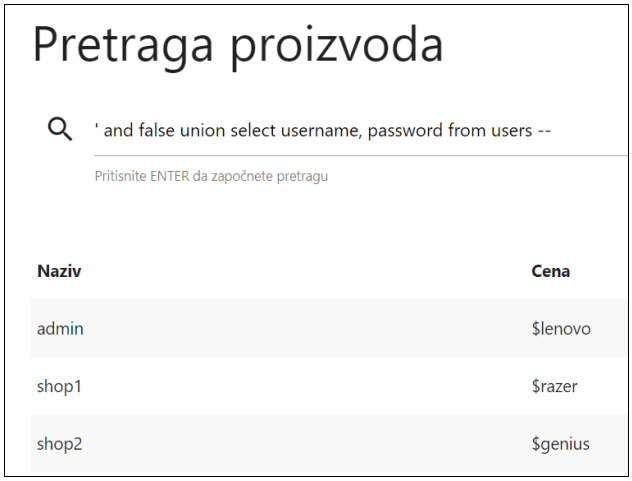
Iz prethodnog koraka saznaje se da je tabela sa korisnicima sačuvana pod imenom *users*. Sledećim unosom dolazi se do naziva svih kolona ove tabele:

```
' and false union select 1, column_name from
information_schema.columns where
table_name='users' --
```

Poslednji korak je definisanje upita koji prikazuje korisničko ime i lozinku svih korisnika:

```
' and false union select username, password
from users --
```

Tabela sa rezultatima sadržaće korisničko ime i lozinku svih korisnika, kao na Sl. 4.



Sl. 4 Korisničko ime i lozinka svih korisnika aplikacije

B. *Način upotrebe u nastavi*

ZRM je predmet master studija na Modulu za računarsku tehniku i informatiku ETF-a, koji je razvijen u okviru Erasmus+ KA2 projekta pod nazivom *Information Security Services Education in Serbia (ISSES)* [7]. Uzimajući u obzir probleme kod praktičnog izučavanja tema koje se obrađuju na predmetu, a koji su pomenuti u poglavlju I, za studente je napravljeno zatvoreno virtuelno laboratorijsko okruženje u okviru Laboratorije za informacionu bezbednost, koja je uspostavljena i opremljena u okviru istog (*ISSES*) projekta.

Svaki student ima sopstveno virtuelno laboratorijsko okruženje kome pristupa korišćenjem *VPN* veze. Za različite teme koje se obrađuju na predmetu koriste se različite konfiguracije virtuelnih laboratorijskih okruženja. U slučaju laboratorijskih vežbi u kojima se koristi *SQLiTrainer* sistem laboratorijsko okruženje se sastoji od jedne virtuelne mašine na kojoj je pokrenut *Ubuntu Linux* i na kojoj je pokrenut *SQLiTrainer* sistem. Studenti mogu da pristupe aplikacijama *SQLiTrainer* sistema iz svojih pretraživača korišćenjem *VPN* veze.

SQLiTrainer sistem se koristi za izvođenje laboratorijiskih vežbi, ali i kao deo finalnog praktičnog ispita na predmetu. Zahvaljujući načinu implementacije sistema, uz minimalne izmene u kodu, moguće je jednostavno izmeniti svaku od aplikacija tako da se dobiju drugačiji problemi sa istom tematikom, čime je omogućeno da se isti sistem iskoristi i prilikom obučavanja studenata, ali i prilikom provere njihovog znanja.

Još jedan važan aspekt, kada je u pitanju upotreba sistema, jeste i jednostavnost instalacije i konfiguracije. Prilikom konfigurisanja laboratorijskog okruženja za izvođenje laboratorijskih vežbi potrebno je pokrenuti više aplikacija istovremeno. Način implementacije *SQLiTrainer* sistema omogućava da se svaka aplikacija može pokrenuti na različitom portu, čime se prethodno postiže na jednostavan način. Kada je u pitanju instalacija, jedno rešenje je da se aplikacije iskopiraju na svaku virtuelnu mašinu i pokrenu odgovarajućim komandama. Ovo rešenje je vremenski zahtevno i nepraktično kada postoji veliki broj studenata na predmetu, a samim tim i veliki broj laboratorijskih okruženja koje je potrebno pripremiti. Način implementacije *SQLiTrainer* sistema dozvoljava da se iskoristi neki od alata za automatizaciju, kao što je na primer *Ansible* [8], čime se prethodni problem efikasno rešava na taj način što se napišu odgovarajuće skripte za ovaj alat kojima se prethodno manuelni posao kopiranja i pokretanja aplikacija u potpunosti automatizuje.

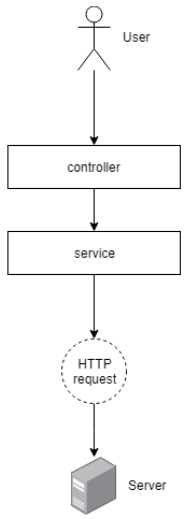
Opisani sistem je korišćen u nastavi u dve uzastopne školske godine 2019/2020 i 2020/2021. Prve školske godine finalni praktični ispit uspešno je savladalo 67% studenata (22/33), dok je u drugoj školskoj godini uspešno bilo 65% studenata (53/81).

IV. Implementacija *SQLiTrainer* sistema

Sistem čine četiri aplikacije identične strukture. Dve demonstriraju *Union-based SQLi*, pravolinijski i jednostavan napad gde se u kratkim iteracijama otkriva sve više podataka iz baze. Sledeća demonstrira *Blind SQLi* za koju je specifično da se podaci iz baze nikada ne prikazuju napadaču, i spada u teže napade za manuelno izvršavanje. Poslednja demonstrira *SQLi Login Bypass*, gde je cilj napadača da uspešno izvrši korisničku prijavu bez prethodnog poznavanja bilo kog korisničkog imena ili lozinke. U ovoj aplikaciji student se upoznaje sa upotrebom *HTTP Proxy* server, kao alata u izvršavanju *SQLi* napada.

Sistem je implementiran kao skup *Java* veb aplikacija koristeći *Spring Boot* [9] i *h2* [10] *in-memory* bazu podataka. Korisnički interfejs aplikacija kreiran je koristeći *HTML*, *CSS* i *JavaScript*. Za kreiranje lepšeg korisničkog interfejsa korišćen je *MaterializeCSS* [11], dok *AngularJS* [12] pojednostavljuje pisanje koda za interakciju sa korisnikom i *HTTP* (eng. *Hypertext Transfer Protocol*) komunikaciju sa serverom.

Korisnički interfejs i serverska aplikacija mogu se posmatrati kao dve odvojene aplikacije koje komuniciraju preko *HTTP*-a.



Sl. 5 Životni ciklus zahteva u aplikaciji korisničkog interfejsa

Korisnički interfejs realizovan je kao *Single Page Application* (*SPA*). Srž aplikacije čine *HTML* stranica *index.html* i *JavaScript* kod *app.js*. Resursi aplikacije organizovani su u posebne direktorijume:

- *css* – sadrži *CSS* biblioteke i definicije stilova specifičnih za aplikaciju,
- *images* – sadrži fotografije korišćene na korisničkom interfejsu i
- *js* – sadrži *JavaScript* biblioteke i kod aplikacije.

Organizacija koda je slojevita i definiše dva sloja:

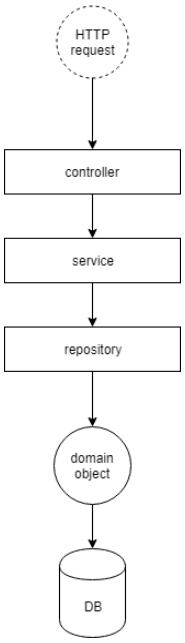
- *controller* – kod koji obrađuje korisničke akcije i
- *service* – kod za komunikaciju sa serverskom aplikacijom.

Sloj *controller* je viši sloj i zavisan od sloja *service*, a korisnički zahtevi prolaze kroz oba sloja aplikacije. Na Sl. 5 prikazano je kako korisnički zahtev putuje kroz aplikaciju korisničkog interfejsa i do serverske aplikacije.

Organizcija serverskog koda je takođe slojevita, gde su slojevi aplikacije predstavljeni sledećim *Java* paketima:

- *controller* – klase za razmenu podataka sa aplikacijom korisničkog interfejsa,
- *service* – klase koje izvršavaju poslovnu logiku aplikacije i
- *repository* – klase za čitanje i upis domenskih objekata u bazu podataka.

Pored navedenih paketa postoji i paket *domain* koji sadrži klase koje predstavljaju domenske entitete.



Sl. 6 Životni ciklus zahteva u serverskoj aplikaciji

Zahtevi koji dolaze sa korisničkog interfejsa prolaze kroz sve slojeve aplikacije. Način razmene podataka između slojeva prikazan je na Sl. 6 na primeru zahteva koji na kraju rezultuje upisom u bazu podataka. Svaki sloj aplikacije zavisan je od sledećeg (nižeg) sloja, a često su svi paketi aplikacije zavisni od domenskog. U nekim implementacijama izostavljen je servisni sloj, jer ne bi sadržao nikakvu logiku, već bi samo prosleđivao podatke sledećem sloju.

Svaka instanca aplikacije poseduje svoju instancu baze, koja se pokreće zajedno sa aplikacijom i čuva podatke u memoriji. Kod serverske aplikacije prate *SQL* skripte koje se izvšavaju nad bazom prilikom pokretanja aplikacije, kako bi pri svakom pokretanju aplikacije stanje baze bilo identično. Skripte se nalaze u sledećim fajlovima:

- *schema.sql* – izvršava se prva i njena uloga je da kreira relacionu šemu baze i
- *data.sql* – popunjava bazu podacima.

Prilikom zaustavljanja aplikacije zaustavlja se i baza podataka, a podaci iz nje trajno nestaju.

Varijacije problema laboratorijskih vežbi mogu se kreirati na više načina. Najjednostavniji je izmeniti *SQL* skripte čime se menja početno stanje baze podataka. Kompleksnije izmene, poput izmene imena kolona i tabela, zahtevaju i manje izmene u kodu aplikacije. Navedene skripte se mogu pokrenuti i nad nekom drugom bazom podaka, na primer *PostgreSQL* ili *MySQL*, za šta je dovoljno u konfiguracionom fajlu aplikacije navesti parametre za konekciju. Korišćenje različitih baza podataka povećava kompleksnost zadatka jer koriste različite dijalekte *SQL* jezika.

V. Zaključak

U ovom radu predstavljen je jedan novi sistem za učenje o *SQLi* sigurnosnim propustima u aplikacijama. *SQLiTrainer* služi za praktičnu obuku programera u oblasti razvoja bezbednog softvera. Realizovan je kao skup ranjivih aplikacija kojima se mogu demonstrirati različiti tipovi *SQLi* propusta koji se mogu javiti u aplikacijama. Omogućava proveru postojanja propusta u sigurnom okruženju, kao i učenje tehnika kojima se mogu otkloniti uočeni propusti. U radu je prikazan opis *SQLiTrainer* sistema i način njegovog korišćenja. U budućnosti se planira dodavanje skupa vežbi u kojima će studenti biti u prilici da isprave propuste koji postoje u aplikacijama.

Zahvalnica

Autori žele da se zahvale master inž. Adrianu Milakoviću i prof. dr Pavlu Vuletiću na pomoći prilikom uvođenja sistema na laboratorijske vežbe na predmetu Zaštita računarskih sistema i mreža.

Literatura

[1] „*SQL injection*,“ Dostupno na: https://owasp.org/www-community/attacks/SQL_Injection, poslednji put pristupano: maj 2021.

[2] „*OWASP*,“ Dostupno na: https://owasp.org/, poslednji put pristupano: maj 2021.

[3] „*Avatao*,“ Dostupno na: https://avatao.com/, poslednji put pristupano: maj 2021.

[4] „*Juice Shop*,“ Dostupno na: https://owasp.org/www-project-juice-shop/, poslednji put pristupano: maj 2021.

[5] „*Security Idiots*,“ Dostupno na: http://www.securityidiots.com/, poslednji put pristupano: maj 2021.

[6] „Zaštita računarskih sistema i mreža,“ Dostupno na: https://www.etf.bg.ac.rs/fis/karton_predmeta/13M111ZRM-2019, poslednji put pristupano: maj 2021.

[7] „*Information Security Services Education in Serbia*,“ Dostupno na: https://isses.etf.bg.ac.rs/, poslednji put pristupano: maj 2021.

[8] „*Ansible*,“ Dostupno na: https://www.ansible.com/, poslednji put pristupano: maj 2021.

[9] „*Spring Boot*,“ Dostupno na: https://spring.io/projects/spring-boot, poslednji put pristupano: maj 2021.

[10] „*H2 database*,“ Dostupno na: https://www.h2database.com /html/main.html, poslednji put pristupano: maj 2021.

[11] „*Materialize CSS*,“ Dostupno na: https://materializecss.com/, poslednji put pristupano: maj 2021.

[12] „*AngularJS*,“ Dostupno na: https://angularjs.org/, poslednji put pristupano: maj 2021.

Abstract

During the application development process security

vulnerabilities can occur and remain in application in production environment. These vulnerabilities can cause confidentiality, integrity and availability breaches. SQLiTrainer represents a set of vulnerable applications that can be used to demonstrate different types of SQLi vulnerabilities. Implementation of the SQLiTrainer system is given in the paper and the examples on how to use the system for programmer practical training is proposed. The system was successfully used for laboratory exercises at the Advanced System and Network Security course at the University of Belgrade, School of Electrical Engineering.

SQLITRAINER - SYSTEM FOR LEARNING ABOUT SQLI VULNERABILITY IN APPLICATIONS

Djordje Madic, Zarko Stanisavljevic