Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Útoky SQL Injection

Jan Zamazal

© 2015 ČZU v Praze

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Katedra informačního inženýrství

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jan Zamazal

Informatika

Název práce

Útoky SQL Injection

Název anglicky

SQL injection attacks

Cíle práce

Bakalářská práce je tematicky zaměřená na problematiku zabezpečení www databází před Injection útoky. Cílem práce je:
- vymezit teoretické principy problematiky relačních databází a jejich zabezpečení proti zneužití dat prostřednictvím Injection útoků,
- zmapovat současnou situaci v této problematice, identifikovat její relevantnost a požadavky s ní spojené,
- navrhnout řešení těchto identifikovaných požadavků,
- navržené záležitosti ověřit formou praktického řešení,
- ověřené záležitosti zobecnit pro další možná použití.

Metodika

Použitá metodika zadané bakalářské práce bude založena na studiu a analýze dostupných informačních zdrojů a existujících řešení v dané oblasti. Stěžejní pro vypracování této závěrečné práce budou metody a techniky relačně databázové technologie v kontextu se zabezpečením takto evidovaných dat. Navrhované řešení bude zohledňovat identifikované požadavky a očekávání spojená s řešenou záležitostí. Na podkladě syntézy teoretických poznatků a dosažených výsledků budou formulovány závěry této bakalářské práce a následně zobecněny pro další možná použití.

Doporučený rozsah práce

40-50 stran

Klíčová slova

relační databázová technologie, SQL, zabezpečení databázových evidencí, zneužití dat, Injection útok

---

Doporučené zdroje informací

BEGG, C., CONOLLY, T., HOLOWCZAK, R.: Mistrovství databáze, profesionální průvodce tvorbou efektivních databází. Computer Press. Brno 2009. ISSBN 978-80-251-2328-7

BRYLA, B., LONEY, K.: Mistrovství v Oracle Database 10g. Computer Press Brno 2006. EAN 978802512779

HERMANDEZ, M.: Návrh databází, GRADA 2005. ISBN 80-247-0900-7.

LACKO, L: ORACLE. Správa, programování a použití databázového systému. Computer Press Brno 2007. EAN 97880251149002.

MOLINARO, A.: SQL Kuchařka programátora. Computer Press. Brno 2009. ISBN 978-80-251-2617-2

POKORNÝ, J.: Databázové systémy. ČVUT Praha 2013. ISBN 978-80-01-05212-9

Předběžný termín obhajoby

2015/06 (červen)

Vedoucí práce

doc. Dr. Ing. Václav Vostrovský, Ph.D.

Elektronicky schváleno dne 10. 11. 2014

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 10. 11. 2014

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 16. 03. 2015

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Útoky SQL Injection" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 16.3.2015 _____

Poděkování

Rád bych touto cestou poděkoval panu doc. Ing. Václavu Vostrovskému, Ph.D. za vedení práce a připomínky, které mi poskytl při jejím vypracováním.

Útoky SQL Injection

SQL injection attacks

Souhrn

Bakalářská práce se zabývá tématem útoků typu SQL Injection na webové aplikace napojené na databázový systém a obranou proti nim. V teoretické části předkládá stručnou historii jazyka SQL a základy práce s SQL databázemi. Dále obsahuje přehled možných typů napadení webové aplikace způsobem SQL Injection. Rozebírá a porovnává způsoby ochrany proti tomuto typu útoku. V praktické části testuje reálnou hrozbu užitím popsaných postupů na nezabezpečené webové aplikaci. Je zde také navržena nejvhodnější ochrana WWW aplikace, která je implementována a testováním je ověřena její funkčnost v praktickém prostředí.

Summary

The bachelor's thesis deals with the issue of SQL Injection attacking web applications connected to the database system. After a brief history of SQL language and basics of SQL queries, the work goes on with the overview of possibilities to attack web applications by SQL Injection. On examples the thesis analyzes and compares the ways of the protection against this type of attack. The practical part of the work presents results of testing the unsecured web application. In the end the thesis proposes the most appropriate security of WWW application, which is also implemented. Its functionality is verified by testing in its natural environment.

Klíčová slova: SQL Injection, webová aplikace, útok na databáze, databázový systém, WWW aplikace, SQL, zabezpečení databáze, zneužití dat

Keywords: SQL Injection, web application, database attack, database system, WWW application, SQL, database security, data abuse

Obsah

Seznam tabulek

1 Úvod

Přestože od zveřejnění možné zranitelnosti typu SQL Injection uplynula již dlouhá doba[1], (datum spuštění dej do poznámky pod čarou) stále je spuštěné a veřejně dostupné velké množství webových aplikací napojených na databázi, které nejsou proti tomuto typu útoků chráněny. SQL Injection útok je termín, označující zneužití neošetřeného vstupu aplikace pro neoprávněný přístup k datům uložených v databázi SQL, jejich změnu či další nakládání. Jsou při tom zneužity uživatelské vstupy aplikace v podobě například HTML formuláře nebo parametrů v adresním řádku. Potenciální útočník do nich může zadat údaje, které aplikace neočekává. Tímto je možné data z databáze neoprávněně zcizit, změnit je nebo vymazat. Aplikace nezabezpečené proti tomuto typu útoku přidávají tyto uživatelské vstupy přímo do dotazu pro SQL databázi. Nezabezpečené uživatelské vstupy vystavují aplikaci vysokému riziku možnosti uživatele upravit tento dotaz podle libosti a zobrazit si tak data, ke kterým nemá oprávnění přistupovat. Útočník může tímto způsobem do databáze nová data vložit nebo uložená data pozměnit podle toho, co s napadenou aplikací zamýšlí. I přes vysoká rizika spojená s touto zranitelností je na světě stále vysoké množství programátorů a administrátorů zejména webových aplikací, kteří buď o tomto nebezpečí nevědí, nebo ho ignorují v domnění, že jejich aplikace není pro útok lukrativní. Opak je však pravdou. Jakákoliv databáze, shromažďující data o svých uživatelích, obsahuje cenné osobní údaje, které se při nedbalosti ze strany správce mohou dostat do nepovolaných rukou a v horším případě mohou být zneužity.

První část této práce se zabývá teoretickými principy provedení SQL Injection útoku a klade si za cíl obecně zhodnotit stávající stav dané problematiky. Ve druhé kapitole se dotýká historie jazyka SQL a vysvětluje jeho základy pro pochopení principů možného napadení Injection útokem. Kapitola dále pokračuje popisem teorie průběhu celého útoku. Na příkladech jsou zde prezentovány praktiky při něm používané. Na konci kapitoly jsou rozebrány možnosti ochrany a jejich zhodnocení po teoretické stránce.

Druhá část práce je cílena na návrh řešení problému zabezpečení proti SQL Injection útoku a ověřit účinnost navrhovaného řešení. Ve třetí kapitole je proto prezentován výchozí stav

[1] NT Web Technology Vulnerabilities. *Phrack*. 1998, vol. 8, issue 54. Dostupné z: http://phrack.org/issues/54/8.html

– tedy aplikace bez zabezpečení, která je testována na rezistenci proti útoku. Jsou zde navrhnuta dvě řešení zabezpečení.

Čtvrtá kapitola obsahuje testy jak výchozí nezabezpečené aplikace, tak navrhovaných řešení. Cílem je zjistit, která z nich jsou pro použití v praxi vhodnější a lépe zabraňují napadení. V závěru kapitoly jsou předloženy okomentované výsledky praktických testů původního i nově navržených řešení.

Poslední kapitola shrnuje výsledky testů a hodnotí přínos práce.

2 Cíle a metodika

2.1 Cíle práce

Tématem bakalářské práce je zabezpečení databází napojených na WWW aplikace proti SQL Injection útokům. Cílem teoretické části je stručně seznámit s historií jazyka SQL a na příkladech ukázat základy práce s SQL databázemi. Toto je důležité pro pochopení teoretických principů problematiky SQL Injection útoků. Část teoretické práce má za cíl osvětlit obecné možnosti vniknutí útočníka do databáze a na příkladech popsat teorii útoku na aplikaci bez jakéhokoliv zabezpečení. Zároveň si klade za úkol zmapovat současnou situaci v této problematice, kdy stále velké množství webových aplikací nemá žádnou ochranu implementovánu.

Cílem praktické části této práce je navržení řešení problému nezabezpečení webových aplikací proti SQL Injection útokům, implementování řešení do testovací aplikace a jeho otestování praktickými zkouškami a testy v reálném prostředí. Pro další použití v praxi jsou výsledky reálných testů interpretovány a zobecněny.

2.2 Metodika práce

Metodika je založena na studiu a analýze dostupných zdrojů pojednávajících o dané tématice. Takto získané metody a techniky jsou využity při návrhu řešení. Syntézou teoretických poznatků a dosažených výsledků v praktické části bude formulován a zobecněn závěr této práce.

3 Teoretické principy řešené problematiky

3.1 Základy SQL

Pro pochopení principu SQL Injection útoku je nutné znát alespoň základní principy práce s SQL databázemi, následující kapitola se proto bude věnovat vysvětlení běžných přístupů k datům a jejich správu pomocí tohoto dotazovacího jazyka. Okrajově je zmíněna historie SQL a jeho standardizace.

3.1.1 Historie jazyka SQL

Jazyk SQL je od začátku spojený s vývojem relačních databází. Vývojář společnosti IBM E. F. Codd začátkem sedmdesátých let publikoval článek *Relační model dat pro velké sdílené datové banky*, kde matematicky popisuje, jak by v budoucnu mohla vypadat manipulace a ukládání dat za pomoci tabulkové struktury.[2] Relační databáze i samotný jazyk SQL v podstatě vycházejí z tohoto článku.

Na základě Coddovi práce IBM nastartovalo vývoj projektu *Systém/R*, v rámci jehož první fáze byl vytvořen funkční prototyp relačního databázového systému. Současně s tímto systémem byl vyvíjen v projektu jazyk pro dotazování databází. Ten, v té době nazývaný *SEQUEL (Structured English Query Language)*, byl později ve spojení s distribucí přepracovaného prototypu do několika zákaznických míst k ohodnocení, přejmenován na *SQL (Structured Query Language)*.[3]

3.1.2 Standardy SQL

První oficiální standart pro jazyk SQL vydal v roce 1986 *American National Standard Institute (*Dále jen *ANSI) – ANSI X3.135*[4]. Až na několik drobných odlišností je založený na systému *DB2 SQL*. *International Standard Organization* (Dále jen *ISO*) ho přijala v roce 1987.

[2] CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM*. 1970, vol. 13, issue 6, s. 377-387. DOI: 10.1145/362384.362685.
[3] GROFF, James R a Paul N WEINBERG. *SQL: kompletní průvodce*. Vyd. 1. Brno: CP Books, 2005, 936 s. ISBN 80-251-0369-2.
[4] ANSI X3.135:1986. Database Language SQL. Washington: ANSI, 1986.

Standart vydaný *ISO*, byl následně roku 1989 schválen Americkou vládou jako *Federal Information Processing Standard* (dale jen *FIPS*), která jej zároveň pozměnila a rozšířila. Tato upravená verze je obvykle nazývána *SQL-89* nebo *SLQ1*.[5]

3.1.3 Základní konstrukce jazyka SQL

Práce v databázích SQL je rozdělená do tří základních typů, podle způsobu nakládání s informacemi. První z nich je Definice dat: *DDL – Data Definition Language („jazyk pro definici dat")*. Skládání příkazů definující nové struktury.

`CREATE, ALTER, DROP`

Druhý - Manipulace s daty: *DML – Data Manipulation Language („jazyk pro manipulaci s daty")*. Dotazování nad množinou dat vkládající nové údaje, odstranění či změny již zapsaných záznamů.

`SELECT, INSERT, UPDATE, DELETE`

A třetí - Řízení přístupu k datům: *DCL – Data Control Language („jazyk pro ovládání dat")*. Určování přístupových práv uživatelů nebo například řízení transakcí.

`GRANT, REVOKE`

3.1.4 Příklady základního použití

3.1.4.1 Vytvoření tabulky

Pro ilustraci bude vytvořena tabulka `users` obsahující jména, příjmení a telefonní čísla, všechny sloupce budou datového typu `VARCHAR`. Zde se hodí použití konstrukce `CREATE`, kterou se v SQL vytváří nové objekty, v tomto případě se bude jednat o tabulku.

```
CREATE TABLE users
(name VARCHAR(25),
surname VARCHAR(25),
phone VARCHAR(9)
);
```

Vytvoření tabulky je možné ověřit příkazem `ALTER`:

[5] GROFF, James R a Paul N WEINBERG. *SQL: kompletní průvodce*. Vyd. 1. Brno: CP Books, 2005, 936 s. ISBN 80-251-0369-2.

```
ALTER TABLE users;
```

3.1.4.2 Vložení dat

Klíčovým slovem `INSERT` jsou vloženy do nově vytvořené struktury vzorová data. `INTO` určuje název tabulky, do které se budou zapisovat. V množině `VALUES` jsou potom hodnoty určené k vložení.

Tímto příkazem se zapisují údaje *Petr*, *Novotný* a *728425981* do tabulky `users`, do sloupců `name`, `surname` a `phone`. V druhém příkladu jsou opět do tabulky `users` vkládána do stejných sloupců data *Marie*, *Mašinková*, *723571597*.

```
INSERT INTO users
(name, Surname, Phone)
VALUES
('Petr','Novotný','728435981');

INSERT INTO users
(name, Surname, Phone)
VALUES
('Marie','Mašinková','723571597');
```

Tabulka s daty tedy vypadá následovně

Name	**Surname**	**Phone**
Petr	Novotný	728425981
Marie	Mašinková	723571597

Tabulka 1 - tabulka z ukázkové databáze

3.1.4.3 Výpis dat

Záznamy se vypisují pomocí příkazu `SELECT`, který zároveň vybírá, z jakých sloupců budou informace vybrány. Nedílnou součástí je klíčové slovo `FROM`, určující tabulku obsahující požadovaná data. Klauzule `WHERE` zajišťuje výběr řádků, které obsahují pouze záznamy vyhovující podmínce, která následuje za tímto klíčovým slovem.

Následujícím příkazem jsou tedy vybrány řádky z tabulky `users`, kde sloupec `surname` obsahuje hodnotu *Novotný*.

```
SELECT * FROM users WHERE surname='Novotný';
```

3.1.4.4 Úprava dat

Aktualizaci některého z řádku tabulky je možné provézt příkazem `UPDATE`, obsahující klíčové slovo `SET` určující sloupec, ve kterém se informace přepíše. `WHERE` zde jako v případě výpisu dat filtruje řádek (řádky), ve kterých se má změna provést.

Názorně je změněno telefonní číslo u druhého uživatele s příjmením *Mašinková* na *728912375*.

```
UPDATE users
SET phone='728912375'
WHERE surname='Mašinková'
```

3.1.4.5 Odstranění záznamu

Libovolné řádky z tabulky se odstraňují příkazem `DELETE`. Opět je zde použita klauzule `FROM` s názvem zpracovávané tabulky a podmínku `WHERE`, filtrující ovlivněné záznamy.

Následujícím příkazem bude odstraněn záznam z tabulky `users` s hodnotou *Novotný* ve sloupci `surname`.

```
DELETE FROM users
WHERE surname='Novotný';
```

3.1.4.6 Odstranění tabulky

Pro zahození celé tabulky uložené v databázi je používáno sousloví `DROP TABLE`, doplněné o název tabulky, se kterou je manipulováno.

Tabulka `users` bude z databáze odstraněna:

```
DROP TABLE users;
```

Toto byly základní funkcionality jazyka SQL, které jsou potřeba pro pochopení principů prezentovaných v mé práci.

3.2 SQL Injection útok

Tento typ útoku na SQL databáze využívá neošetřených vstupů aplikace. Při zpracování dat, zadávaných typicky uživatelem, v některých případech však i aplikací samotnou, se spustí podstrčený kód z některého ze vstupů. Tento kód změní výsledky příkazů používaných v aplikaci pro komunikaci s bází dat.[6]

3.2.1 Možnosti útočníka

Aplikace, ve kterých není implementována jakákoliv obrana proti útoku typu SQL Injection, se vystavují vysokému riziku proniknutí do databáze a neoprávněnému nakládání s daty v ní uložených. Při správně vedeném útoku je možné data nejen číst, ale i je upravovat či odstraňovat. Zkušený útočník může pomocí tohoto typu útoku ovládnout celý stroj, na kterém aplikace s databázovým systémem běží.

3.2.2 Vedení útoku

Níže je pro názornost uveden jednoduchý příklad, jak takový útok může vypadat. Použit je typický formulář pro přihlášení do webové aplikace, napsaný v jazyce HTML a zpracovávaný skriptovacím jazykem PHP, který dnes využívá velké množství aplikací.

```
<form name="login" action="login.php" method="post">
  Username: <input type="text" name="userName" />
  Password: <input type="password" name="password" />
  <input type="submit" value="Přihlásit" />
</form>
```

Ve standardním případě, který aplikace očekává, uživatel do tohoto formuláře zadá své uživatelské jméno a heslo. Zadané vstupy aplikace předloží databázovému serveru pro kontrolu existujícího řádku s těmito údaji. Pokud záznam nebude nalezen, odpoví databáze aplikaci negativním hlášením. Aplikace jej následně interpretuje uživateli.

Soubor `login.php`, ve kterém se v tomto příkladu zpracovávají data z formuláře, typicky obsahuje podobnou kontrukci pro kontrolu uživatelského jména a hesla:

[6] CLARKE, Justin. SQL injection attacks and defense. Waltham, MA: Elsevier, c2012, xvviii, 547 p. ISBN 9781597499637.

```
$query = 'SELECT userName FROM users WHERE
userName=\''.$userName.'\' AND password=\''.$password.'\'';
```

V běžném případě, kdy uživatel zadá do vstupu uživatelské jméno řetězec `admin` a jako heslo `xxwuwjs04`, příkaz pro SQL Server bude vypadat následovně:

```
SELECT userName FROM users WHERE userName='admin' AND
password='xxwuwjs04';
```

Když aplikace dostane od databáze odpověď potvrzující existenci záznamu, umožní přihlášení uživatele. V opačné situaci zobrazí chybové hlášení informující o špatně zadaných přihlašovacích údajích.

Pokud nepůjde o běžného uživatele, ale útočníka, který se bude snažit podstrčit řetězec způsobující změnu chování aplikace a zadá jako uživatelské jméno řetězec `admin` a heslo `jakekolivheslo' OR '1'='1`, bude dotaz pro databázi pozměněný:

```
SELECT userName FROM users WHERE userName='admin' AND
password='jakekolivheslo' OR '1'='1'
```

Po ukončení kontroly existence záznamu databázový stroj vykoná porovnání 1=1, které vždy končí pravdou a aplikaci odpoví také vrácením pravdy. Skript tedy uživatele přihlásí i bez správně zadaných údajů.

3.2.3 Fáze útoku

Vedení útoku SQL Injection na libovolnou webovou aplikaci by se dalo rozdělit do těchto částí:

1. identifikace vstupů potenciálně použitelných pro vniknutí do systému,
2. nalezení vhodného vstupu, který útočník použije,
3. prověření možnosti proniknutí do systému pomocí cíleného „pokus-omyl“ postupu, využívajíc chybových hlášení daného databázového stroje,
4. vedení samotného útoku kombinací podstrčených řetězců do vstupů v různých částech aplikace.

Aby útočník mohl pokračovat v útoku, každá předcházející část musí být úspěšná.[7]

3.2.3.1 Identifikace vstupů

Vzhledem k typickému uspořádání webové aplikace klient/server je zřejmé, že program bude mít dvě části. Pomocí webového prohlížeče uživatel obsluhuje klientskou část, která v tomto případě pomocí protokolu HTTP vytváří požadavky odesílané serveru. Konkrétní metody tohoto protokolu pro předávání dat mezi klientem a serverem jsou GET a POST.

3.2.3.2 Metoda GET

Metoda GET využívá pro sdílení hodnot a parametrů URL adresu webové aplikace. Při zobrazení webové stránky vznikne na server požadavek pro spuštění skriptu či části aplikace s parametry uvedenými právě v URL. Například při odeslání požadavku adresy

```
index.php?parametr1=hodnota1&parametr2=hodnota2
```

bude předán serverové části aplikace dotaz na spuštění skriptu index.php s parametry parametr1 a parametr2, nabývajících hodnot hodnota1 a hodnota2. Tato metoda je používána především v těch částech aplikace, kde je již odkaz vygenerován a uživatel na něj pouze klikne. Typicky je to fulltextové vyhledávání, zobrazení konkrétního produktu nebo například volba kategorie.[8]

3.2.3.3 Metoda POST

Obsah parametrů metody POST je předán části aplikace nebo skriptu, který je za tímto účelem deklarován přímo v daném formuláři. Stejně jako metoda GET využívá metoda POST pro předání parametrů požadavek na webový server. Na rozdíl od metody GET však metoda POST odesílané parametry nezařazuje do URL a jsou tedy skryty uživateli. Metoda POST se typicky uplatňuje při vyplňování jakýchkoliv interaktivních formulářů s potvrzením odeslání ke zpracování – např. registrační či přihlašovací formuláře webových aplikací.[9]

---

[7] CLARKE, Justin. SQL injection attacks and defense. Waltham, MA: Elsevier, c2012, xvviii, 547 p. ISBN 9781597499637.

[8] HTTP Methods: GET vs. POST. *W3Schools.com* [online]. [cit. 2015-8-28]. Dostupné z: http://www.w3schools.com/tags/ref_httpmethods.asp

[9] Tamtéž.

3.2.4 Nalezení vhodného vstupu

Útočník hledá uživatelské vstupy, které by mohly být potenciálně využitelné ke vniknutí do databáze. Zejména jsou to formuláře `<form>` s uživatelskými vstupy `<input>`, při předpokladu použití POST metody. Pro metodu GET jsou potenciálně zranitelné URL adresy obsahující parametry s hodnotami. Jejich analýzou útočník zjišťuje, ze kterých budou surová vstupní data předávána SQL serveru a jsou tedy vhodné k vedení útoku na bázi dat.

3.2.5 Chyby vypisované SQL Serverem

Protože se tato práce zabývá konkrétním databázovým systémem *MySQL*, je zde uvedena ukázková chyba vypisovaná tímto SQL serverem. Chybová hlášení vypisována na uživatelský výstup jsou velkou potenciální hrozbou pro celou běžící aplikaci, jelikož z nich lze odvodit spoustu informací o stroji, na kterém aplikace běží i o samotné aplikaci komunikující s databází.

Příklad chybového hlášení:

```
Warning: mysql_fetch_array(): supplied argument is not a
valid MySQL result
Resource in /var/www/whatever.com/showcategory.php on line 8
```

V případě, že server odpovídá touto chybou na vložení apostrofu do řetězce hodnoty parametru v URL adrese, vyplývá z toho možnost zranitelnosti Injection v tomto vstupu aplikace. Hodnota parametru je totiž aplikací bez jakékoliv úpravy doplňována do dotazu pro SQL server.

Například takto:

```
$category = $_GET["category"];
mysql_query("SELECT * FROM products WHERE
category='$category'");
while($row = mysql_fetch_array($result, MYSQL_NUM) {
     printf("ID: %s Name: %s, $row[0], $row[1]);
}
```

Dotaz obyčejného uživatele, respektive adresa URL, na kterou odkazuje link aplikaci, by vypadala následovně:

```
showcategory.php?category=motherboards
```

Skript obstarávající zobrazování kategorií by převzal hodnotu parametru `motherboards` a použil ho v příkazu pro databázi. Výsledný dotaz prováděný SQL serverem by byl:

```
SELECT * FROM products WHERE category='motherboards'
```

Databázový systém vybere záznamy obsahující `motherboards` ve sloupci `category` a řádky vrátí aplikaci, která je zobrazí.

Pokud však útočník podstrčí do URL adresy místo řetězce

```
showcategory.php?category=motherboards
```

řetězec upravený

```
showcategory.php?category=motherboards'
```

dojde k chybě a SQL server vrátí hlášení, usnadňující útočníkovi proniknout do systému.[10]

3.2.6 Testování zranitelnosti

Při zjištění, u jakých vstupů je možné předávání vložených dat SQL stroji, útočník pokračuje testováním jednotlivých technik a pro zjištění nejvhodnějšího typu útoku vkládá pozměněné řetězce ovlivňující běh aplikace.

3.2.7 Samotný útok

Po zjištění základních informací o napadané aplikaci, které útočník provedl v minulých bodech, je možné začít samotný útok. Jeho postup se liší podle toho, zda útočník chce data odcizit, změnit je, odstranit či ovládnout celý databázový stroj. Vytvoří tedy řetězec měnící výsledky chování aplikace podle svých potřeb a podstrčí ho do uživatelského vstupu.

[10] Clarke, J.: SQL Injection attacks and defense, Elsevier, Inc., 2012, 978-1-59749-963-7

3.2.8 Způsoby použití zranitelnosti SQL Injection

3.2.8.1 Vložení uvozovek

Mnoho aplikací není ošetřených ani proti jednoduchému přidání apostrofů do vstupu. Tímto způsobem lze ukončit dotaz předčasně a vrátit například jen jeden řádek. Vynechá se tím totiž z původního dotazu klauzule `WHERE`, která bude odstraněna ze zpracovávaného dotazu pomocí komentáře na konci zaměněného řetězce:

```
user ‘
OR ‘1’=’1
```

3.2.8.2 Komentáře

Jazyk SQL nebere v úvahu žádné znaky od znaku začínající komentář. Tato funkcionalita je výhodou pro útočníka, který může zaměnit řetězec doplňující zbytek dotazu, který nahradí původní, zamýšlený tvůrcem aplikace. Existuje mnoho způsobů, jakým lze zaměnit zbytek dotazu.[11]

```
user’--
admin’ #
1--
1 OR 1=1--
‘ OR ‘1’=‘1’--
-1 and 1=2--
‘and ‘1’=’2’--
1/*comment*/
```

3.2.8.3 Vložené komentáře

Pokud je aplikace chráněná filtrováním klíčových slov ve vstupech, je možné tuto ochranu obejít používáním vložených komentářů, které databázový systém nebude brát v potaz. Je tedy možné zamaskovat klíčové slovo vložené do dotazu za účelem změny chování aplikace, například celý SQL příkaz:

```
DE/**/LETE FRO/**/M users
```

[11] CLARKE, Justin. SQL injection attacks and defense. Waltham, MA: Elsevier, c2012, xvviii, 547 p. ISBN 9781597499637.

Tento řetězec podstrčený na některém vstupu aplikace způsobí odstranění všech záznamů z tabulky `users`.

3.2.8.4 Spojování tabulek

Pro spojení více výsledků příkazů `SELECT` je v SQL používána konstrukce `UNION`, spojující všechny výsledky do jedné tabulky. Při podstrčení správného řetězce je konstrukci možné použít pro získání dat i z jiných tabulek, které aplikace v dané části nepoužívá:

```
user' union all
select accountNumber from accounts --
```

Odpovědí na tento dotaz budou nejen záznamy struktury, ze které požaduje data samotná aplikace, ale také všechny hodnoty sloupce `accountNumber` tabulky `accounts`.

3.2.8.5 Zjišťování počtu sloupců

Jednoduchým navyšováním hodnoty parametru příkazu `ORDER BY` je možné zjistit počet sloupců dané tabulky. V běžném případě se klíčové spojení `ORDER BY` používá pro vyžádání seřazení vracených dat, kdy hodnotou parametru je zpravidla číslo sloupce, v němž jsou hodnoty, podle kterých se výsledný soubor dat má řadit.

Pokud tedy útočník použije příkaz uvedený níže, vrácený soubor dat bude seřazen podle hodnot prvního sloupce:

```
user' ORDER BY 1 --
```

Kontinuálním navyšováním hodnoty dosáhne odpovědi chybovým hlášením ve chvíli, kdy hodnota o jednotku překročí počet sloupců obsažených v tabulce.

3.2.8.6 Zjišťování názvů sloupců

Pro zjištění názvů sloupců v tabulce je možné použít příkaz `GROUP BY`, který za normálních okolností slouží ke kompletaci dat podle klíčového sloupce. Pokud je do dotazu vložen náhodný název sloupce, SQL server vrací chybové hlášení obsahující správný název:

```
user‘ GROUP BY whatever --
```

3.2.8.7 Zjišťování datových typů sloupců

Zde najde využití opět funkce `UNION`, pomocí které je možné zjistit či alespoň odhadnout datový typ sloupců. Pokud je znám jejich počet, není problém v dotazu požadovat vypsání záznamu hodnoty typu INTEGER[12], tzn. bez uvozovek. Pokud SQL server vrátí chybové hlášení, jedná se nejspíše o datový typ VARCHAR:[13][14]

```
user‘ union select 1 from accounts –
```

3.2.8.8 Information_schema

Information_schema je množinou pohledů, vracející informace o struktuře celého databázového systému. Pomocí tohoto nástroje je možné zjistit v podstatě cokoliv o jakémkoliv objektu, který se v systému nachází, ať už je to tabulka, sloupec, rutina nebo pohled. Popisuje tedy v podstatě celý databázový systém.

Information_schema používá pohledy i pro popis sebe sama. Tento základní návrh popisu byl specifikován v roce 1992 ve standardu *ISO/IEC 9075:1992* (*Database Language SQL*). Díky standardizaci obsahuje tento diagram každý databázový systém a vždy má shodnou strukturu. Z toho důvodu je vysoce náchylný ke zneužití za účelem získání informací o struktuře napadané databáze, které je možné využít pro další způsoby útoku. Mezi nejčastější údaje, které útočník bude chtít získat, patří názvy tabulek, které je možné vyžádat příkazem:

```
UNION SELECT GROUP_CONCAT(table_name) FROM
information_schema.tables WHERE version=10
```

Názvy sloupců je možné získat zadáním tohoto dotazu:

```
UNION SELECT GROUP_CONCAT(column_name) FROM
information_schema.columns WHERE table_name = 'tablename'
```

[12] Datový typ celých čísel
[13] Datový typ řetězců neurčité délky
[14] CLARKE, Justin. SQL injection attacks and defense. Waltham, MA: Elsevier, c2012, xvviii, 547 p. ISBN 9781597499637.

3.2.9 Ochrana v novějších verzích MySQL

Výše popsané základní postupy pro napadání MySQL (SQL) databází mají ukázat a vysvětlit obecné principy a možností útočníka. Některé z těchto typů útoků již v aktuálních verzích databázových systémů nefungují vůbec či v pozměněné nebo omezené formě. Zranitelnost tohoto typu je známá již přes deset let a vývojáři si konečně začínají uvědomovat, jaká hrozba se v ní skrývá. Proto do svých produktů implementují více či méně účinné ochrany proti tomuto napadení. Proti jakým typům útoků je aktuální verze systému *MySQL* rezistentní, bude ověřeno v praktické části této práce, konkrétně při testování nezabezpečeného prostředí a skriptu, který nemá implementované žádné ochrany proti SQL Injection.

3.2.10 Způsoby ochrany proti SQL Injection

3.2.10.1 Escapování znaků (zastaralé a nebezpečné)

Escapování znaků[15] jinou odpovídající sekvencí při doplnění uživatelských nebo dalších dynamicky doplňovaných částí SQL dotazu je v případě skriptovacího jazyka PHP a databáze MySQL realizováno pomocí této funkce:

```
mysql_real_escape_string($unescapedString)
```

V tomto konkrétním případě, tedy použití kombinace PHP a MySQL, escapovací funkce vkládá před tyto speciální znaky oddělné čárkou zpětná lomítka a databázový server je potom nečte jako znaky speciální, nýbrž jako součást textového řetězce.

```
\x00, \n, \r, \, ', ", \x1a
```

Tato ochrana je však nedostatečná a při podstrčení řetězce, který žádné speciální znaky neobsahuje, není účinná. Jako příklad lze uvést následující řetězec, který neobsahuje žádné speciální znaky, a přesto je pomocí něj možné změnit chování aplikace.

```
1 OR 1=1
```

Tento řetězec by byl účinný např. při pokusu zneužít metodu GET, změnou parametrů v adresním řádku. Další problémy přináší nutnost ošetřovat touto funkcí jakoukoliv

[15] Nahrazení znaků majících v daném kontextu speciální význam.

proměnnou, která se může ocitnout v dotazu pro databázi. Je zde tedy obrovský prostor pro vytvoření bezpečnostní chyby při vývoji aplikace.

3.2.10.2 Svazování proměnných v PHP Data Objects

Pro skriptovací jazyk PHP existuje objektové rozhraní, jehož pomocí lze pracovat s rozličnými databázovými systémy. Nazývá se *PHP Data Objects* (Dále jen *PDO*) a umožňuje komunikaci mezi databázemi. Svým uživatelům přináší výhody v podobě robustnosti a přenosnosti na jiné databázové systémy pouhým přenastavením parametrů konstruktoru.

Dalším přínosem je bezesporu i účinná ochrana proti útokům typu SQL Injection v podobě svazování proměnných. V zásadě jde o využití předpřipravených dotazů, o jejichž správnou konstrukci se za vývojáře stará sama knihovna *PDO*. Usnadňuje práci programátorovi a rovněž pomáhá předcházet chybám, které by mohl zapříčinit lidský faktor při ručním zadávání dotazů. Tím zároveň předchází narušení bezpečnosti dané aplikace.[16]

Svazování proměnných v reálné aplikaci může vypadat podobně, jako v tomto příkladu:

```
$dbh = new
PDO('mysql:host=localhost;dbname=lidicoznam;charset=utf8',
$login, $heslo); //připojení k db a vybrání tabulky
$jmeno = $_POST['jmeno'];
$prijmeni = $_POST['prijmeni'];

$sth = $dbh->prepare('insert into uzivatele (jmeno, prijmeni)
values (:jmeno, :prijmeni)');
$sth->bindValue(':jmeno', $jmeno, PDO::PARAM_STR);
$sth->bindValue(':prijmeni', $prijmeni, PDO::PARAM_STR);
$sth->execute(); //vykoná příkaz a vrátí počet ovlivněných
řádků
```

Nově vytvářená třída má parametry konstruktoru, obsahující proměnné použitého databázového systému, databáze v něm uložené, kódování a uživatelské jméno a heslo, pod kterým aplikace do databáze přistupuje:

[16] VESELÝ, Ondřej. Ochrana proti SQL Injection v PHP. *Birknet* [online]. 2013 [cit. 2015-02-15]. Dostupné z: http://www.birknet.eu/node/ochrana-proti-sql-injection-v-php/

```
$dbh = new
PDO('mysql:host=localhost;dbname=lidicoznam;charset=utf8',
$login, $heslo);
```

Skript uvedený níže zapisuje do tabulky `uzivatele` nové záznamy, do sloupců `jmeno` a `prijmeni`. Může být použit pro zpracování dat z odeslaného HTML formuláře v libovolné části webové aplikace. Tomu je uzpůsobeno i načtení obsahu proměnných `jmeno` a `prijmeni` z odeslaných dat pomocí metody POST:

```
$jmeno = $_POST['jmeno'];
$prijmeni = $_POST['prijmeni'];
```

Metoda `prepare` knihovny PDO vytvoří připravený dotaz v databázi s hodnotami `:jmeno` a `:prijmeni`:

```
$sth = $dbh->prepare('insert into uzivatele (jmeno, prijmeni)
values (:jmeno, :prijmeni)');
```

K těmto hodnotám v připraveném příkazu „svážeme“ pomocí metody `bindValue` proměnné poskytnuté webovou aplikací. Konstanta `PDO::PARAM_STR` určuje, že se data mají interpretovat jako datový typ CHAR nebo VARCHAR.

```
$sth->bindValue(':jmeno', $jmeno, PDO::PARAM_STR);
$sth->bindValue(':prijmeni', $prijmeni, PDO::PARAM_STR);
```

Metoda `execute` vykoná celý připravený příkaz a její návratová hodnota se rovná počtu ovlivněných řádků.

```
$sth->execute();
```

4 Navržené řešení

4.1 Koncepce

Praktická část této práce obsahuje reálné testy výše popsaných typů útoků na nezabezpečenou část aplikace a komentuje jejich výsledky. Cílem testování je ověřit či vyvrátit poznatky prezentované v teoretické části práce a zhodnotit reálnou hrozbu proniknutí útočníka do nezabezpečené aplikace. Dále zde jsou prezentována řešení pro ochranu webových aplikací proti útokům typu SQL Injection a implementace jejich nejdůležitějších funkcí. Práce pokračuje otestováním účinnosti těchto řešení na praktických příkladech, kde budou zkonfrontovány s postupy popsanými v teoretické části práce. Pro porovnání je stejným způsobem otestováno původní, nezabezpečené řešení. Veškeré zdrojové kódy a ukázky použití jsou psané v jazyce PHP verze 5 a jsou spouštěné knihovou webového serveru *Apache2* - PHP verze 5.4.36-0+deb7u3. Předpokládají použití SQL Databázového systému *MySQL* ve verzi 5.6.23.

4.2 Použitý formulář

Pro ukázky jednoduché napadnutelnosti a zobrazení dat uložených v databázi bez znalosti správných přihlašovacích údajů – a to jak do webové aplikace, tak do databázového systému – je použit formulář pro přihlašování uživatelů do aplikace. Formulář obsahuje dva vstupy „uživatelské jméno“ typu text a „heslo“ typu password. Pro demonstraci účinnosti implementovaných ochran poslouží identický přihlašovací formulář.

```
<form method="POST" action="login.php">
    <label for="login">uživatelské jméno:</label>
    <input type="text" name="userName" value="" />
    <label for="password">heslo:</label>
    <input type="password" name="password" value="" />
    <input type="submit" name="submit" value="Přihlásit">
</form>
```

4.3 Zpracující skript bez zabezpečení

Tento formulář zpracovává skript, který používá vstupy z formuláře v dotazu odesílaném na databázový server pro zjištění identity přihlašovaného uživatele. Na základě porovnání uživatelského jména a hesla potvrdí či zamítne přihlášení. Skript není žádným způsobem chráněn proti napadnutí, do databáze se připojuje pod uživatelským jménem root. Řádky zobrazující vložené uživatelské jméno a heslo jsou zde jen pro rychlou orientaci v konstrukci aktuálního dotazu. Stejnému účelu slouží i zobrazení právě odeslaného dotazu a případné zobrazení chyby, kterou vrátí databáze.

```
if(isset($_POST["submit"])){

    $hostname = "localhost";
    $username = "root";
    $password = "toor";
    $database = "test";

    $databaseLink = mysql_pconnect($hostname, $username,
$password);

    if(!$databaseLink){
        echo 'Could not connect to server: ' .
mysql_error($databaseLink);
    }
    else{
        if(!mysql_select_db($database, $databaseLink)){
          echo 'Cannot select database: ' .
mysql_error($databaseLink);
        }
    }

    $userName = $_POST["userName"];
    $password = $_POST["password"];

    echo "vstup uziv. jmeno:<br />".$userName."<br /><br />";
    echo "heslo:<br />".$password."<br /><br />";
```

```
    $query = "SELECT users_username FROM users WHERE
users_username='".$userName."' AND
users_password='".$password."';";
    echo "dotaz:<br />".$query."<br /><br />";

    if(!$result = mysql_query($query,$databaseLink)){
        echo "chyba:<br />";
        echo mysql_error($databaseLink)."<br /><br />";
    }
    else{
        if(mysql_num_rows($result) == 0){
            echo "Chybné přihlašovací údaje";
        }
        else{
            while($row = mysql_fetch_array($result)) {
            echo "přihlášen!<br /><br />";
            echo "přihlášeno pod uživatelským jménem:";
            echo $row['users_username']." <br /><br />";
            }
        }
    }
}
```

4.4 Zpracující skript se zastaralým zabezpečením

Lehká úprava původního nezabezpečeného skriptu spočívá v přidání funkce `mysql_real_escape_string`, která zpracovává data přijatá metodou POST a ukládá je do proměnných v aplikaci. Toto řešení ochrany nelze doporučit, jelikož nezajišťuje skutečnou ochranu proti SQL Injection. Další problém nastává v datových typech proměnné zpracovávané touto funkcí. V případě, že pro zpracování použijeme proměnou datového typu string, jež obsahuje číselnou hodnotu, databáze bude mít problém se zpracováním.

```
if(isset($_POST["submit"])){

    $hostname = "localhost";
    $username = "test_user";
    $password = "test";
    $database = "test";
```

```
    $databaseLink = mysql_pconnect($hostname, $username,
$password);

    if(!$databaseLink){
        echo 'Could not connect to server: ' .
mysql_error($databaseLink);
    }
    else{
        if(!mysql_select_db($database, $databaseLink)){
          echo 'Cannot select database: ' .
mysql_error($databaseLink);
        }
    }

    $userName = mysql_real_escape_string($_POST["userName"]);
    $password = mysql_real_escape_string($_POST["password"]);

    echo "vstup uziv. jmeno:<br />".$userName."<br /><br />";
    echo "heslo:<br />".$password."<br /><br />";

    $query = "SELECT users_username FROM users WHERE
users_username='".$userName."' AND
users_password='".$password."';";
    echo "dotaz:<br />".$query."<br /><br />";

    if(!$result = mysql_query($query,$databaseLink)){
        echo "chyba:<br />";
        echo mysql_error($databaseLink)."<br /><br />";
    }
    else{
        if(mysql_num_rows($result) == 0){
            echo "Chybné přihlašovací údaje";
        }
        else{
            while($row = mysql_fetch_array($result)) {
            echo "přihlášen!<br /><br />";
            echo "přihlášeno pod uživatelským jménem:";
            echo $row['users_username']." <br /><br />";
            }
        }
```

```
    }
}
```

4.5 Zpracující skript s doporučeným zabezpečením

Při použití svazování proměnných pomocí *PDO* knihovny se nejdříve vytvoří nová instance třídy *PDO*, která je zapsána do proměnné `$dbh`. Parametry konstruktoru této třídy jsou adresa databázového stroje, jméno databáze, uživatelské jméno a heslo. Zbytek skriptu pracuje na stejném principu jako předešlé skripty.

```
if(isset($_POST["submit"])){

    $db_hostname="localhost";
    $db_dbname="test";
    $db_username="test_user";
    $db_password="test";

    try {
        $dbh = new
PDO('mysql:host='.$db_hostname.';dbname='.$db_dbname.'',
$db_username, $db_password);
    } catch (PDOException $e) {
        print "Error!: " . $e->getMessage() . "<br/>";
        die();
    }

    $username = $_POST["username"];
    $password = $_POST["password"];

    echo "vstup uziv. jmeno:<br />".$username."<br /><br />";
    echo "heslo:<br />".$password."<br /><br />";


    $stmt = $dbh->prepare('SELECT users_username FROM users
WHERE users_username = :username AND users_password =
:password');
    $dbh->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
    $stmt->bindValue(':username', $username, PDO::PARAM_STR);
    $stmt->bindValue(':password', $password, PDO::PARAM_STR);
    $stmt->execute();
```

```
    $query = $stmt->queryString;
    echo "dotaz:<br />";
    echo $query."<br /><br />";

    if($stmt->rowCount() == 0){
        echo "Chybné přihlašovací údaje";
    }
    else{
        while($row = $stmt->fetchAll()){
            echo "přihlášen!<br /><br />";
            echo "přihlášeno pod uživatelským jménem:<br />";
            $logged_username = $row[0]["users_username"];
            echo $logged_username." <br /><br />";
        }
    }

}
```

4.6 Testovací prostředí

Pro vytvoření testovací databáze byl nainstalován čistý operační systém *Debian 7.2 Wheezy*, a doinstalovány byly balíky takzvaného *LAMP*[17] serveru. Balíky obsahují webový server *Apache 2*, databázový systém *MySQL* a PHP, coby jazyk pro psaní skriptů, zpracovávaných webovým serverem. Jedná se o tyto balíky:

```
apache2 apache2-doc mysql-server mysql-client  php5 php5-
mysql libapache2-mod-php5
```

4.6.1 Nezabezpečené prostředí

Vše bylo ponecháno ve výchozím nastavení pro simulaci naprosto nezabezpečeného systému. Do databáze se aplikace připojovala pod uživatelským jménem root, což představuje velký nedostatek z pohledu bezpečnosti. I v dnešní době si však mnoho vývojářů či administrátorů neuvědomuje nebo nepřiznává vysoký stupeň nebezpečí, kterého se přihlašováním pod uživatelským jménem root dopouští. Toto uživatelské jméno je při testování záměrně použito jako demonstrace toho, jakým typům útoku lze zabránit

[17] Prostředí tvořené komponentami: Linux**,** Apache**,** MySQL a PHP

pouhým vytvořením nového uživatele. Po instalaci má knihovna PHP povolené zobrazování chyb přímo na výstup stránky, proto se veškeré chyby ve skriptech či dotazech pro databázi zobrazí i běžným uživatelům. Z těchto informací lze v některých případech vyčíst verze operačního systému, která je na serveru instalována, případně jaké verze softwaru server používá. Tyto informace také mohou pomoci útočníkovi získat kontrolu nad databázovým systémem nebo i celým strojem, na kterém systém běží. Toto prostředí bylo použito i pro testy částečně zabezpečené aplikace pomocí metody `mysql_real_escape_string()`. Pokud je aplikace zabezpečená pomocí této zastaralé metody, lze předpokládat, že se tvůrce nezabýval zabezpečením ze strany prostředí, na kterém je aplikace provozována. V opačném případě by bylo nejspíše použito zabezpečení pomocí *PDO* knihovny a svazování proměnných.

4.6.2 Zabezpečené prostředí

Testování aplikace chráněné metodou svazování proměnných *PDO* knihovnou je spouštěno ve stejném prostředí jako předchozí nechráněné ukázky. Pro připojení do databáze je však použito vlastní, pro tento účel vytvořené, uživatelské jméno a heslo. Zobrazování chyb v nastavení PHP je vypnuto.

4.7 Testovací databáze

Databáze vytvořená za účelem testování obsahuje pouze dvě tabulky. První tabulka zahrnuje uživatelská jména a hesla pro ověření identity při přihlašování.[18] Druhá tabulka obsahuje emailové adresy uživatelů a čísla jejich bankovních účtů. V praxi by se použití dvou tabulek pro skladování údajů spárovaných pomocí jediného klíče (uživatele), u takto malé databáze, nahradilo jedinou tabulkou, pro demonstraci některých typů útoků a možností útočníka jsou však použity tabulky dvě.

[18] Hesla jsou pro jednodušší pochopení prováděných testů uložena v prostém textu, na účinnost útoků to nemá žádný vliv.

users_id	users_username	users_password
2	ondrejstary	skola,098
5	admin	pojliskw1.2
6	Mariefenyklova	uhodnout23,+

Tabulka 2 - tabulka users

emails_id	emails_users_id	emails_email	emails_account
1	1	novak@gmail.com	8763579461351/0800
2	2	ondra@email.com	87321849848/0300
3	5	admin@website.com	546413184864/0100
4	6	fenyklova.marie@company.com	764131311111/3500

Tabulka 3 - tabulka emails

4.8 Postup testování

U každého typu útoku je nejdříve vyzkoušen daný postup na nechráněný kód zpracující data odeslaná z přihlašovacího formuláře. Následně je proveden test stejného postupu na variantě obsahující zastaralou ochranu pomocí escapování znaků. Nakonec jsou stejné metody útoku testovány na upravené verzi skriptu, zabezpečené pomocí svazování proměnných v PDO.

4.9 Jednotlivé vstupy

- uživatelské jméno – řetězec vyplněný ve formuláři do vstupu username
- heslo – řetězec vyplněný ve formuláři do vstupu password
- dotaz – finální dotaz složený z částí zahrnujících odeslané řetězce z formuláře
- výsledek – Řetězec zobrazovaný aplikací po odeslání formuláře – tento výstup se v běžných případech zobrazuje uživateli

5 Reálné testy

5.1 Nezabezpečená aplikace

5.1.1 Předpokládaný scénář běžného uživatele – správné heslo

Uživatelské jméno	`petrnovak`
Heslo	`heslo1234*`
Dotaz	`SELECT users_username FROM users WHERE users_username='petrnovak' AND users_password='heslo1234*';`
Výsledek	`přihlášeno pod uživatelským jménem:petrnovak`

Tabulka 4 - nezabezpečená aplikace - předpokládaný scénář běžného uživatele - správné heslo

Jako uživatelské jméno byl použit řetězec obsahující jméno skutečně uložené v databázi, heslo je také správné a koresponduje s heslem uloženým v databázi ve stejném řádku jako uživatelské jméno.

5.1.2 Předpokládaný scénář běžného uživatele – špatné heslo

Uživatelské jméno	`petrnovak`
Heslo	`sesasdj165151*`
Dotaz	`SELECT users_username FROM users WHERE users_username='petrnovak' AND users_password='sesasdj165151*';`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 5 - nezabezpečená aplikace - předpokládaný scénář běžného uživatele - špatné heslo

Uživatelské jméno bylo zadáno správné, ale heslo bylo chybné.

5.1.3 Vložení uvozovek

Uživatelské jméno	`admin' OR '1'='1`
Heslo	`jakekolivheslo`
Dotaz	`SELECT users_username FROM users WHERE users_username='admin' OR '1'='1' AND users_password='jakekolivheslo';`

Výsledek	`přihlášeno pod uživatelským jménem:admin`

Tabulka 6 - nezabezpečená aplikace - vložení uvozovek

Vstup uživatelského jména byl využit k pokusu o proniknutí do aplikace pomocí pozměněného řetězce využívající vložených uvozovek, do vstupu hesla byl vyplněn náhodný řetězec. Přihlášení proběhlo úspěšně.

5.1.4 Komentáře

Uživatelské jméno	`admin' --`
Heslo	`opetajkekolivheslo`
Dotaz	`SELECT users_username FROM users WHERE users_username='admin' -- ' AND users_password='opetajkekolivheslo';`
Výsledek	`přihlášeno pod uživatelským jménem:admin`

Tabulka 7 nezabezpečená aplikace - komentáře

Vstup uživatelského jména byl využit k pokusu o proniknutí do aplikace pomocí znaků pro ignorování zbytku dotazu (komentář), do vstupu hesla byl vyplněn náhodný řetězec. Jelikož ho databázový systém díky komentáři na konci vstupu uživatelského jména nepovažuje za součást dotazu, pokus o proniknutí do aplikace byl úspěšný.

5.1.5 Spojování tabulek

Uživatelské jméno	`admin' UNION ALL SELECT emails_account FROM emails --`
Heslo	`jakekoliv`
Dotaz	`SELECT users_username FROM users WHERE users_username='admin' UNION ALL SELECT emails_account FROM emails -- ' AND users_password='jakekoliv';`

Výsledek	`přihlášeno pod uživatelským jménem:admin`
	`přihlášeno pod uživatelským jménem:8763579461351/0800`
	`přihlášeno pod uživatelským jménem:87321849848/0300`
	`přihlášeno pod uživatelským jménem:546413184864/0100`

	přihlášeno pod uživatelským jménem:764131311111/3500

Tabulka 8 - nezabezpečená aplikace - spojování tabulek

Uživatelské jméno je vyplněno částí dotazu pokoušejícího se zobrazit obsah tabulky emails. Na vstupu pro heslo ani v tomto případě nezáleží, protože na konci řetězce uživatelského jména je opět znak pro vyřazení zbytku řádku z dotazu. V tomto případě došlo k vyzrazení čísel bankovních účtů uživateli, který k jejich zobrazení nemá oprávnění. Tato čísla účtu byla uložena v jiné tabulce než uživatelská jména a hesla.

5.1.6 Zjišťování počtu sloupců

Uživatelské jméno	admin‘ ORDER BY 1 --
Heslo	heslo
Dotaz	SELECT users_username FROM users WHERE users_username='admin‘ ORDER BY 1 -- ' AND users_password='heslo';
Výsledek	Chybné přihlašovací údaje

Tabulka 9 - nezabezpečená aplikace - zjišťování počtu sloupců 1

Vstup uživatelského jména je vyplněn řetězcem pokoušejícím zjistit počet sloupců v tabulce. Heslo je opět bezpředmětné, neboť je z dotazu vynecháno. Nebyla zobrazena žádná chyba, pouze zpráva o chybně zadaných přihlašovacích údajích. Proti tomuto typu útoku je již aktuální verze *MySQL* zabezpečena.

Uživatelské jméno	admin‘ ORDER BY 6 --
Heslo	heslo
Dotaz	SELECT users_username FROM users WHERE users_username='admin‘ ORDER BY 6 -- ' AND users_password='heslo';
Výsledek	Chybné přihlašovací údaje

Tabulka 10 - nezabezpečená aplikace - zjišťování počtu sloupců 2

Vstup uživatelského jména je vyplněn řetězcem pokoušejícím se zjistit počet sloupců v tabulce. Heslo je opět bezpředmětné, neboť je z dotazu vynecháno. Nebyla zobrazena žádná chyba, pouze zpráva o chybně zadaných přihlašovacích údajích. Proti tomuto typu útoku je již aktuální verze *MySQL* zabezpečena.

5.1.7 Zjišťování názvů sloupců

Uživatelské jméno	`user‘ GROUP BY whatever --`
Heslo	`heslo`
Dotaz	`SELECT users_username FROM users WHERE users_username='user‘ GROUP BY whatever -- ' AND users_password='heslo';`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 11 - nezabezpečená aplikace - zjišťování názvů sloupců

Uživatelské jméno je řetězcem pokoušejícím se zjistit název sloupce v tabulce. Vstup hesla je opět z dotazu vynechán. Nebyla zobrazena žádná chyba, pouze zpráva o chybně zadaných přihlašovacích údajích. Proti tomuto typu útoku je již aktuální verze *MySQL* zabezpečena.

5.1.8 Zjišťování datových typů sloupců

Uživatelské jméno	`user‘ union select 1 from accounts --`
Heslo	`heslo`
Dotaz	`SELECT users_username FROM users WHERE users_username='user‘ union select 1 from accounts --' AND users_password='heslo';`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 12 - nezabezpečená aplikace - zjišťování datových typů sloupců

Vstup uživatelského jména je využit k pokusu o zjištění datového typu sloupců z tabulky. Heslo je z dotazu vynecháno. Nebyla zobrazena žádná chyba, pouze zpráva o chybně zadaných přihlašovacích údajích. Proti tomuto typu útoku je již aktuální verze *MySQL* zabezpečena.

5.1.9 Information_schema

Uživatelské jméno	`admin' UNION SELECT GROUP_CONCAT(table_name) FROM information_schema.tables --`
Heslo	`heslo`

Dotaz	`SELECT users_username FROM users WHERE users_username='admin' UNION SELECT GROUP_CONCAT(table_name) FROM information_schema.tables -- ' AND users_password='heslo';`
Výsledek	`Illegal mix of collations for operation 'UNION'`

Tabulka 13 - nezabezpečená aplikace - information_schema

Uživatelské jméno je použito pro pokus o zjištění názvů tabulek ze struktury information_schema. Heslo je z dotazu vynecháno. K zobrazení názvů tabulek uložených ve struktuře information_schema nedošlo pouze z důvodu jiného datového typu sloupce `users_username`. Díky tomu, že je aplikace přihlášena k databázi pomocí uživatelského jména root, má však útočník plný přístup k této struktuře.

5.2 Aplikace zabezpečená escapováním znaků

5.2.1 Předpokládaný scénář běžného uživatele – správné heslo

Uživatelské jméno	`petrnovak`
Heslo	`heslo1234*`
Dotaz	`SELECT users_username FROM users WHERE users_username='petrnovak' AND users_password='heslo1234*';`
Výsledek	`přihlášeno pod uživatelským jménem:petrnovak`

Tabulka 14 - aplikace zabezpečená escapováním znaků - předpokládaný scénář běžného uživatele - správné heslo

Jako uživatelské jméno byl použit řetězec obsahující jméno skutečně uložené v databázi, heslo je také správné a koresponduje s heslem uloženým v databázi, ve stejném řádku jako uživatelské jméno.

5.2.2 Předpokládaný scénář běžného uživatele – špatné heslo

Uživatelské jméno	`petrnovak`
Heslo	`sesasdj165151*`
Dotaz	`SELECT users_username FROM users WHERE users_username='petrnovak' AND`

	`users_password='sesasdj165151*';`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 15 - aplikace zabezpečená escapováním znaků - předpokládaný scénář běžného uživatele - špatné heslo

Uživatelské jméno bylo zadáno správné, ale heslo bylo náhodné.

5.2.3 Vložení uvozovek

Uživatelské jméno	`admin' OR '1'='1`
Heslo	`jakekolivheslo`
Dotaz	`SELECT users_username FROM users WHERE users_username='admin\' OR \'1\'=\'1' AND users_password='jakekolivheslo';`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 16 - aplikace zabezpečená escapováním znaků - vložení uvozovek

Vstup uživatelského jména byl využit k pokusu o proniknutí do aplikace pomocí pozměněného řetězce využívající vložených uvozovek, do vstupu hesla byl vyplněn náhodný řetězec. Provedené testy aplikace zabezpečené escapováním řetězců zadávaných uživatelů dokazují účinnost této ochrany v tomto konkrétním případě. Funkce byla použita na všechny vstupy zadávané uživatelem. Díky konstrukci tohoto konkrétního dotazu není prostor pro použití řetězce neobsahující speciální znaky. Neznamená to však, že je toto zabezpečení dostatečné vždy a rozhodně nelze doporučit jeho použití. Všechny testy aplikace zabezpečené tímto způsobem skončili informací o chybně zadaných přihlašovacích údajích a útočník nezískal žádná data, ke kterým nemá oprávnění.

5.2.4 Komentáře

Uživatelské jméno	`admin' --`
Heslo	`opetajkekolivheslo`
Dotaz	`SELECT users_username FROM users WHERE users_username='admin\' -- ' AND users_password='opetajkekolivheslo';`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 17 - aplikace zabezpečená escapováním znaků - komentáře

Vstup uživatelského jména byl využit k pokusu o proniknutí do aplikace pomocí znaků pro ignorování zbytku dotazu (komentář), do vstupu hesla byl vyplněn náhodný řetězec neboť ho díky komentáři na konci vstupu uživatelského jména databázový systém nepovažuje za součást dotazu.

5.2.5 Spojování tabulek

Uživatelské jméno	`admin' UNION ALL SELECT emails_account FROM emails --`
Heslo	`jakekoliv`
Dotaz	`SELECT users_username FROM users WHERE users_username='admin\' UNION ALL SELECT emails_account FROM emails -- ' AND users_password='jakekoliv';`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 18 - aplikace zabezpečená escapováním znaků - spojování tabulek

Uživatelské jméno je vyplněno částí dotazu pokoušejícího se zobrazit obsah tabulky emails. Na vstupu pro heslo ani v tomto případě nezáleží, protože na konci řetězce uživatelského jména je opět znak pro vyřazení zbytku řádku z dotazu.

5.2.6 Zjišťování počtu sloupců

Uživatelské jméno	`admin' ORDER BY 1 --`
Heslo	`heslo`
Dotaz	`SELECT users_username FROM users WHERE users_username='admin\' ORDER BY 1 -- ' AND users_password='heslo';`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 19 - aplikace zabezpečená escapováním znaků - zjišťování počtu sloupců 1

Vstup uživatelského jména je vyplněn řetězcem pokoušejícím zjistit počet sloupců v tabulce. Heslo je opět bezpředmětné, neboť je z dotazu vynecháno.

Uživatelské jméno	`admin' ORDER BY 6 --`
Heslo	`heslo`

Dotaz	`SELECT users_username FROM users WHERE users_username='admin\' ORDER BY 6 -- ' AND users_password='heslo';`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 20 - aplikace zabezpečená escapováním znaků - zjišťování počtu sloupců 2

5.2.7 Zjišťování názvů sloupců

Uživatelské jméno	`user' GROUP BY whatever --`
Heslo	`heslo`
Dotaz	`SELECT users_username FROM users WHERE users_username='user\' GROUP BY whatever -- ' AND users_password='heslo';`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 21 - aplikace zabezpečená escapováním znaků - zjišťování názvů sloupců

Uživatelské jméno je řetězcem pokoušejícím se zjistit název sloupce v tabulce. Vstup hesla je opět z dotazu vynechán.

5.2.8 Zjišťování datových typů sloupců

Uživatelské jméno	`user' union select 1 from accounts --`
Heslo	`heslo`
Dotaz	`SELECT users_username FROM users WHERE users_username='user\' union select 1 from accounts -- ' AND users_password='heslo';`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 22 - aplikace zabezpečená escapováním znaků - zjišťování datových typů sloupců

Vstup uživatelského jména je využit k pokusu o zjištění datového typu sloupců z tabulky. Heslo je z dotazu vynecháno.

5.2.9 Information_schema

Uživatelské jméno	`admin' UNION SELECT GROUP_CONCAT(table_name) FROM information_schema.tables --`

Heslo	`heslo`
Dotaz	`SELECT users_username FROM users WHERE users_username='admin\' UNION SELECT GROUP_CONCAT(table_name) FROM information_schema.tables -- ' AND users_password='heslo';`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 23 - aplikace zabezpečená escapováním znaků - information_schema

Uživatelské jméno je použito pro pokus o zjištění názvů tabulek ze struktury information_schema. Heslo je z dotazu vynecháno.

5.3 Aplikace zabezpečená svazováním proměnných v PDO

5.3.1 Předpokládaný scénář běžného uživatele

Uživatelské jméno	`petrnovak`
Heslo	`heslo1234*`
Dotaz	`SELECT users_username FROM users WHERE users_username = :username AND users_password = :password`
Výsledek	`přihlášen jako: petrnovak`

Tabulka 24 - aplikace zabezpečená svazováním proměnných v PDO - předpokládaný scénář běžného uživatele - správné heslo

Jako uživatelské jméno byl použit řetězec obsahující jméno skutečně uložené v databázi, heslo je také správné a koresponduje s heslem uloženým v databázi, ve stejném řádku jako uživatelské jméno.

5.3.2 Předpokládaný scénář běžného uživatele – špatné heslo

Uživatelské jméno	`petrnovak`
Heslo	`sesasdj165151*`
Dotaz	`SELECT users_username FROM users WHERE users_username = :username AND users_password = :password`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 25 - aplikace zabezpečená svazováním proměnných v PDO - předpokládaný scénář běžného uživatele - špatné heslo

Uživatelské jméno bylo zadáno správné, ale heslo bylo náhodné.

5.3.3 Vložení uvozovek

Uživatelské jméno	`admin' OR '1'='1`
Heslo	`jakekolivheslo`
Dotaz	`SELECT users_username FROM users WHERE users_username = :username AND users_password = :password`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 26 - aplikace zabezpečená svazováním proměnných v PDO - vložení uvozovek

Vstup uživatelského jména byl využit k pokusu o proniknutí do aplikace pomocí pozměněného řetězce využívající vložených uvozovek, do vstupu hesla byl vyplněn náhodný řetězec.

5.3.4 Komentáře

Uživatelské jméno	`admin' --`
Heslo	`opetajkekolivheslo`
Dotaz	`SELECT users_username FROM users WHERE users_username = :username AND users_password = :password`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 27 - aplikace zabezpečená svazováním proměnných v PDO - komentáře

Vstup uživatelského jména byl využit k pokusu o proniknutí do aplikace pomocí znaků pro ignorování zbytku dotazu (komentář). Do vstupu hesla byl vyplněn náhodný řetězec, jelikož ho díky komentáři na konci vstupu uživatelského jména databázový systém nepovažuje za součást dotazu.

5.3.5 Spojování tabulek

Uživatelské jméno	`admin' UNION ALL SELECT emails_account FROM emails --`
Heslo	`jakekoliv`

Dotaz	`SELECT users_username FROM users WHERE users_username = :username AND users_password = :password`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 28 - aplikace zabezpečená svazováním proměnných v PDO - spojování tabulek

Uživatelské jméno je vyplněno částí dotazu pokoušejícího se zobrazit obsah tabulky emails. Na vstupu pro heslo ani v tomto případě nezáleží, protože na konci řetězce uživatelského jména je opět znak pro vyřazení zbytku řádku z dotazu.

5.3.6 Zjišťování počtu sloupců

Uživatelské jméno	`admin' ORDER BY 1 --`
Heslo	`heslo`
Dotaz	`SELECT users_username FROM users WHERE users_username = :username AND users_password = :password`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 29 - aplikace zabezpečená svazováním proměnných v PDO - zjišťování počtu sloupců 1

Vstup uživatelského jména je vyplněn řetězcem pokoušejícím se zjistit počet sloupců v tabulce. Heslo je opět bezpředmětné, neboť je z dotazu vynecháno.

Uživatelské jméno	`admin' ORDER BY 6 --`
Heslo	`heslo`
Dotaz	`SELECT users_username FROM users WHERE users_username = :username AND users_password = :password`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 30 - aplikace zabezpečená svazováním proměnných v PDO - zjišťování počtu sloupců 2

5.3.7 Zjišťování názvů sloupců

Uživatelské jméno	`user' GROUP BY whatever --`
Heslo	`heslo`

Dotaz	`SELECT users_username FROM users WHERE users_username = :username AND users_password = :password`
Výsledek	`(žádný výsledek)`

Tabulka 31 - aplikace zabezpečená svazováním proměnných v PDO - zjišťování názvů sloupců

Uživatelské jméno je řetězcem pokoušejícím se zjistit název sloupce v tabulce. Vstup hesla je opět z dotazu vynechán.

5.3.8 Zjišťování datových typů sloupců

Uživatelské jméno	`user‘ union select 1 from accounts –`
Heslo	`heslo`
Dotaz	`SELECT users_username FROM users WHERE users_username = :username AND users_password = :password`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 32 - aplikace zabezpečená svazováním proměnných v PDO - zjišťování datových typů sloupců

Vstup uživatelského jména je využit k pokusu o zjištění datového typu sloupců z tabulky. Heslo je z dotazu vynecháno.

5.3.9 Information_schema

Uživatelské jméno	`admin' UNION SELECT GROUP_CONCAT(table_name) FROM information_schema.tables --`
Heslo	`heslo`
Dotaz	`SELECT users_username FROM users WHERE users_username = :username AND users_password = :password`
Výsledek	`Chybné přihlašovací údaje`

Tabulka 33 - aplikace zabezpečená svazováním proměnných v PDO - information_schema

Uživatelské jméno je použito pro pokus o zjištění názvů tabulek ze struktury information_schema. Heslo je z dotazu vynecháno.

5.4 Hodnocení výsledků

Výsledky testů výchozího a navrhovaných řešení odpovídají s minimálními odchylkami předpokladům prezentovaným v teoretické části práce. Skript, který nebyl zabezpečen žádným způsobem, umožňoval přihlásit se do aplikace bez znalosti správného uživatelského jména a hesla. Při podstrčení správného řetězce bylo rovněž možné číst data i z dalších tabulek uložených v databázi. Nebylo však možné zjistit popsanými metodami počet sloupců v tabulce, stejně jako jejich název a datový typ. Aktuální verze MySQL nezobrazuje v chybových hlášeních informace o správném názvu sloupce, ale pouze oznámí o neexistenci sloupce použitého v dotazu. Při pokusu o zneužití struktury information_schema nastal problém s datovými typy sloupců, ovšem databázový systém by řádky s názvy tabulek umožnil číst, pokud by byl útok veden pomocí uživatelského vstupu jiného datového typu, shodného s information_schema.

Aplikace zabezpečená pomocí escapování speciálních znaků je dle testů ochráněna před těmito základními typy útoků pouze za předpokladu, že je funkce mysql_real_escape_string použita u všech uživatelských vstupů, což však může být snadno porušeno nepozorností vývojáře. Dokladuje to zjevné použití jakéhokoliv vstupu, který uživatel vloží v pozměněné podobě SQL dotazu.

V případě použití ochrany v podobě svazování proměnných knihovnou PDO, dotaz vždy zůstává stejný – předpřipravený, což je vidět na všech výsledcích testů provedených na skriptu ošetřeném touto metodou. Uživatelské vstupy jsou pouze doplňovány do proměnných v tomto předpřipraveném dotazu a nemohou nijak ovlivnit podobu dotazu. Jedná se navíc přímo o použitou technologii k vývoji aplikace, nemůže se tedy stát, že by programátor opomněl některý ze vstupů ošetřit. Ochrana je tak implementována již od samého vývoje aplikace bez dalších pomocných funkcí. Toto je také největší výhodou při vytváření aplikací za pomoci této knihovny.

6 Závěr

Téma SQL Injection útoku je stále aktuální i přes velkou časovou prodlevu od jeho první prezentace. I přes neustálé upozorňování na nebezpečí spojené s ignorováním možností tohoto typu útoku, řada webových aplikací stále nemá implementováno žádné zabezpečení proti zcizení a zneužití dat tímto způsobem. Cílem této práce bylo seznámit s historií jazyka SQL a na příkladech ukázat základy práce s tímto jazykem. Především základy práce s SQL databázemi byly důležité pro pochopení dalších teoretických principů uvedených v práci. Za další cíl si práce kladla prezentovat teoretické poznatky z dané problematiky, osvětlit obecné možnosti útočníka útočícího na databázi a na příkladech popsat teorii útoku na aplikaci, která není zabezpečena proti SQL Injection útokům.

Cílem praktické části práce bylo navrhnout řešení zabezpečení WWW aplikace proti úspěšnému zkonfrontování tímto typem útoku a ověřit rezistenci v praxi. Na základě reálných testů porovnávajících stejnou aplikaci před implementací ochrany a po začlenění zabezpečení způsobem svazování proměnných pomocí *PDO* je možné konstatovat, že pouhou změnou způsobu vývoje dané aplikace je možné zabránit vysokému nebezpečí krádeže dat. Toto řešení s sebou navíc nese výhodu univerzálnosti použití aplikace s širokým portfoliem databázových systémů bez nutnosti přepisovat metody a funkce v aplikaci obsažené. Zvyšuje tedy zároveň robustnost i bezpečnost celé aplikace.

Primární cíl práce - navrhnout účinné řešení zabezpečení proti útokům SQL Injection - byl splněn.

Metodu svazování proměnných pomocí *PDO* v kombinaci s předpřipravenými dotazy, které využívají tutéž technologii, lze pro komunikaci webových aplikací doporučit jako nejvhodnější. V případě již spuštěných a nezabezpečených aplikacích je třeba v nejkratším možném termínu přeprogramovat metody a funkce obsluhující databázi do vyhovující podoby.

7 Použitá literatura

1. CLARKE, Justin. SQL injection attacks and defense. Waltham, MA: Elsevier, c2012, xvviii, 547 p. ISBN 9781597499637.
2. GROFF, James R a Paul N WEINBERG. *SQL: kompletní průvodce*. Vyd. 1. Brno: CP Books, 2005, 936 s. ISBN 80-251-0369-2.
3. ANSI X3.135:1986. Database Language SQL. Washington: ANSI, 1986
4. CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM*. 1970, vol. 13, issue 6, s. 377-387. DOI: 10.1145/362384.362685.
5. NT Web Technology Vulnerabilities. *Phrack*. 1998, vol. 8, issue 54. Dostupné z: http://phrack.org/issues/54/8.html
6. HTTP Methods: GET vs. POST. *W3Schools.com* [online]. [cit. 2015-8-28]. Dostupné z: http://www.w3schools.com/tags/ref_httpmethods.asp
7. VESELÝ, Ondřej. Ochrana proti SQL Injection v PHP. *Birknet* [online]. 2013 [cit. 2015-02-15]. Dostupné z: http://www.birknet.eu/node/ochrana-proti-sql-injection-v-php/