VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

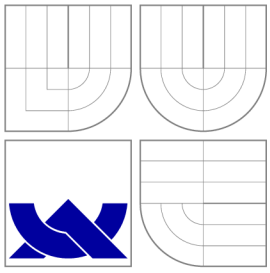
NÁSTROJ PRO DETEKCI ZRANITELNOSTI SQL INJECTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

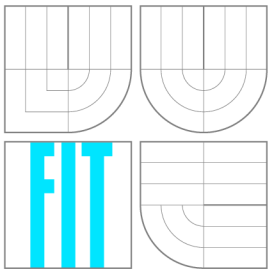
AUTOR PRÁCE
AUTHOR

MATOUŠ KUTYPA

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO DETEKCI ZRANITELNOSTI SQL INJECTION

TOOL FOR SQL INJECTION VULNERABILITY DETECTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MATOUŠ KUTYPA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MAROŠ BARABAS
SUPERVISOR

BRNO 2013

Abstrakt

Bakalářská práce je zaměřena na problematiku bezpečnostní chyby SQL injection. V práci jsou popsány běžně používané postupy při útocích na informační systémy a jsou také probrány možnosti obrany včetně uvedení způsobů správné validace vstupů aplikace. Teoretická část práce obsahuje nezbytný základ, jaký by měl penetrační tester znát, aby byl schopen prověřit vstupy aplikace na odolnost proti útokům typu SQL injection. Součástí práce je analýza, návrh a implementace nástroje specializovaného na detekci obtížně zjistitelných zranitelností webové aplikace. Implementovaný nástroj byl otestován a porovnán s jinými běžně dostupnými nástroji. V rámci práce byla také vytvořena webová aplikace pro demonstraci různých variant zranitelných vstupů SQL injection.

Abstract

The Bachelor thesis is focused on the issue of SQL injection vulnerabilities. The thesis presents commonly used procedures in the attacks against information systems and are also discussed possibilities of defense including the correct ways of input validation. The theoretical part contains the essential foundation of what should the penetration tester know, to be able to examine the inputs of application for SQL injection vulnerability. The thesis also describes analysis, design and implementation of specialized tool for Web application vulnerability detection. The implemented tool was tested and compared with other existing tools. Within the thesis has been also implemented a Web application, which demonstrates many different variants of SQL injection vulnerable inputs.

Klíčová slova

SQL injection, bezpečnost aplikací, informační systém, databázový systém, zranitelnost, útok, obrana, automatizované testování, škodlivý kód, injekce, zabezpečení vstupu.

Keywords

SQL injection, security of application, information system, database system, vulnerability, attack, defense, automatic testing, malicious code, injection, safety of input.

Citace

Matouš Kutypa: Nástroj pro detekci zranitelnosti SQL Injection, bakalářská práce, Brno, FIT VUT v Brně, 2013

Nástroj pro detekci zranitelnosti SQL Injection

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Maroše Barabase.

.....
Matouš Kutypa
14. května 2013

Poděkování

Děkuji mému vedoucímu bakalářské práce Ing. Marošovi Barabasovi za příležitost vypracovat práci na téma SQL injection a za profesionální vedení. Velmi si vážím trpělivosti a osobního přístupu při řešení problémů. Odborné rady mi významně pomohly ke splnění zadání práce.

Obsah

Kapitola 1

Úvod

Tato práce se věnuje problematice bezpečnostní chyby typu SQL injection. Chyba může být do informačního systému zanesena při špatném postupu vývojáře a následky mohou být pro provozovatele napadeného informačního systému fatální.

Slabiny webové aplikace umožňující úspěšný útok typu SQL injection patří podle metriky OWASP TOP 10 - 2013 k nejvíce kritickým slabinám webových aplikací a jsou zařazeny do nejrizikovější kategorie A1 [16]. Zranitelné však nejsou pouze webové aplikace, zranitelná je každá aplikace, která nesprávně validuje své vstupy a tyto vstupy dále využije pro sestavení SQL dotazu.

Slabina SQL injection může zpřístupnit útočníkovi data napadeného informačního systému, bez znalosti jakýchkoliv autentizačních údajů. Uložená data pak může útočník zcizit, modifikovat nebo smazat danou databázi a tím v podstatě zastavit poskytovanou službu.

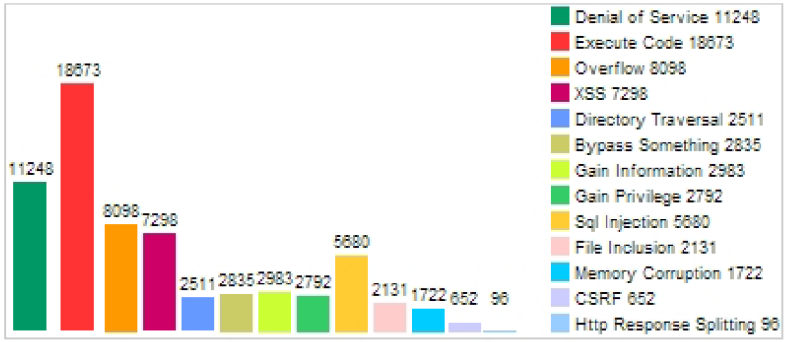
1.1 Motivace

Požadavky na správu a ukládání dat zahrnujících citlivé informace, jako jsou například osobní údaje uživatelů nebo zákazníků, se neustále zvyšují. Bezpečnost dat ukládaných do informačních systémů závisí na zabezpečení daného systému, a právě slabina SQL injection může být pro útočníka klíčem k získání citlivých dat uživatelů, protože vrstva, která má uložená data chránit svou úlohu nesplní.

Mnoho tutoriálů a praktických příkladů, které slouží pro výuku implementace spolupráce webové aplikace s databázovým systémem, neobsahuje správnou validaci a ošetření vstupů uživatele a možná i z tohoto důvodu, je v dnešní době zranitelnost SQL injection tolik rozšířená.

Nemálo automatizovaných nástrojů pro detekci a následné zneužití slabiny typu SQL injection je dostupných veřejnosti na internetu. Snadná dostupnost a čím dál intuitivnější ovládání nástrojů výrazně snižují nároky na znalost útočníka. Detekovat a zneužít slabinu tak může i běžný uživatel bez hlubších znalostí o fungování informačních systémů.

Útoky SQL injection jsou v praxi časté a občas bývají úspěšné. Jako příklad lze uvést úspěšný útok z roku 2011 na vývojářské fórum společnosti Nokia [13]. Během tohoto incidentu byly odcizeny údaje registrovaných uživatelů, mezi kterými byly například data narození nebo kontakty uživatelů. Další v praxi nalezené slabiny SQL injection je možné vyhledat na serveru www.cvedetails.com, kde jsou k dispozici přehledné seznamy aktuálně evidovaných bezpečnostních chyb různých používaných technologií.



Obrázek 1.1: Počet nálezů SQL injection v porovnání s počty ostatních nálezů, zdroj: [15]

Na výše uvedeném grafu jsou znázorněny počty nálezů bezpečnostních chyb rozčleněné do různých kategorií. Jedná se především o chyby nalezené v běžně používaných technologiích, jako jsou například databázové servery. Z grafu vyplývá, že bezpečnostní chyby typu SQL injection jsou poměrně časté, v uvedeném případě zabírají 10% ze všech nalezených chyb.

Slabina SQL injection je velmi závažnou chybou aplikace a její případné zneužití nemusí být odhaleno, protože v mnoha případech je útočník schopný po sobě smazat i záznamy o činnosti. Veškeré role uživatelů nebo omezování výpisu dat z databáze na úrovni aplikace ztrácí význam. Slabina může být zneužita nejen k prolomení autentizace a překročení přidělených práv, ale také i k napadení a převzetí kontroly nad celým serverem hostujícím danou aplikaci.

Problematika útoků SQL injection je aktuální nejen pro stále více populární webové aplikace, ale i pro jakoukoliv architekturu typu klient-server. V praxi se může jednat například o napadení komunikace mezi mobilní aplikací a serverem, což může stejně jako u webové aplikace vést ke kompromitaci poskytovatele dané služby.

1.2 Cíl práce

Cílem této práce je popsat útoky typu SQL injection (v literatuře označované zkratkou SQLIA) a zranitelnosti aplikací, které tyto útoky umožňují. Budou vysvětleny nejčastější příčiny zavedení slabiny SQL injection do aplikace a postupy popisující, jakým způsobem je možné se těmto útokům úspěšně a efektivně bránit. Dále pak budou popsány techniky detekce zranitelností SQL injection využitelné při penetračním testování a možnosti jejich automatizace s dostupnými volně šiřitelnými nástroji. Budou uvedeny výhody a úskalí automatizované detekce slabin SQL injection a obecně popsána problematika automatizovaného testování. Také budou nastíněny různé v praxi používané postupy zneužití odhalené slabiny SQL injection.

Cílem praktické části práce je navržení, implementace a ověření funkčnosti nástroje pro automatizovanou detekci výskytu slabin typu SQL injection ve webové aplikaci. Nástroj bude navržen takovým způsobem, aby byla umožněna dodatečná implementace dalších částí souvisejících s detekcí SQL injection, jako je například crawler, pro možné budoucí využití potenciálu implementované detekce slabiny SQL injection v rámci komplexnějšího automatizovaného testování bezpečnosti webové aplikace. Samotná detekce zranitelného vstupu při útoku SQL injection bude navržena a implementována jako samostatný zásuvný modul a bude umožněno jeho snadné rozšiřování o další databázové systémy. Také bude

možné snadné rozšiřování o další funkcionality jako je práce s cookies nebo možnost testovat aplikaci dostupnou pouze přes protokol HTTPS a to bez zásahu do samotné implementace detekce slabiny SQL injection.

Součástí praktické části je také implementace webové aplikace napojenou na databázi demonstrující některé varianty SQL injection slabin. Aplikace bude obsahovat přehled základních i složitějších případů zranitelných vstupů, které mohou být bez pokročilé znalosti problematiky SQL injection obtížně zjistitelné. Aplikace bude také obsahovat možnost vracení náhodně velké odpovědi pro ztížení automatizované detekce SQL injection a demonstrovat správně ošetřený vstup, který netrpí slabinou SQL injection.

1.3 Konvence

V práci jsou uvedeny ukázky zdrojových kódů v jazyce PHP. Jedná se o populární skriptovací jazyk hojně používaný u webových aplikací. Některé ukázky jsou převzaty přímo z praktické části této práce. Místy jsou používány spojení cílová aplikace, cílová databáze a podobně, čímž je označován systém nebo jeho prvek, na který je veden útok.

Většina uvedených demonstračních SQL dotazů a ukázek injekcí je určena pro databázový systém MySQL. Mnoho z uvedených ukázek určených pro databázový systém MySQL je možné použít i pro jiné databázové systémy, protože se jedná o základní konstrukce jazyka SQL. Některé SQL dotazy jsou přímo určeny pro jiné konkrétní databázové systémy, protože využívají jejich specifické konstrukce.

Následuje ukázka, jak jsou v práci označeny úseky zdrojového kódu a ukázky injekcí:

```
if (isset($e)) {
  $sql = "SELECT * FROM zak WHERE name = '".'"'.$e.'"'";
  $r = mysql_query($sql);
  // vypis chyby
  $error = "something wrong:".mysql_errno();
}
```

Nové pojmy jsou v práci vysvětleny pod čarou a uvedené názvy popisovaných technik jsou zvýrazněny **tučným** písmem. Názvy proměnných, funkcí, hodnot a podobně jsou v textu odlišeny *kurzívou*.

1.4 Struktura práce

V následujících kapitolách je uveden a podrobně vysvětlen útok SQL injection. V kapitole 2.1.1 jsou vyjmenovány a rozděleny do kategorií nejběžnější techniky útoku. Dále jsou popsány typické vstupy webových aplikací náchylné k zavedení slabiny SQL injection a způsoby, jakými je možné aplikaci před útoky bránit a zamezit úspěšnému provedení útoku. Jsou zmíněny i možné dopady a běžně používané postupy útočníka po nalezení slabiny. V kapitole 3 je uveden přehled v praxi používaných nástrojů pro detekci a zneužití slabin SQL injection. Kromě přehledu nástrojů je také obecně rozebrána problematika automatizovaného vyhledávání zranitelných vstupů aplikace. Další kapitola se věnuje popisu praktické části této práce. V rámci praktické části byl implementován nástroj pro detekci obtížně zjistitelných slabin SQL injection a za účelem jeho testování byla vytvořena také webová aplikace demonstrující různé varianty slabin SQL injection.

Kapitola 2

SQL injection

V současné době jsou velmi oblíbené a využívané vícevrstvé architektury typu klient-server. Vrstvení architektury informačních systémů poskytuje řadu výhod, jako je například možnost rozdělení výkonu nebo lepší škálovatelnost. Typickým zástupcem vícevrstvé architektury je architektura třívrstvá, která se skládá z prezenční vrstvy, aplikační vrstvy a vrstvy datové. Aplikační vrstva zpracovává vstup uživatele, na jehož základě poptává data z vrstvy datové. Data jsou poptávány prostřednictvím SQL dotazu, který je sestaven podle hodnoty vstupu uživatele a při nedostatečném ošetření vstupu je uživatel schopen ovlivnit sestavení výsledného SQL dotazu, čímž může získat neautorizovaný přístup k citlivým datům. V třívrstvé architektuře je tedy útok SQL injection veden z vrstvy prezenční na vrstvu datovou s využitím bezpečnostní chyby ve vrstvě aplikační.

SQL je standardizovaný jazyk práci s daty v databázích, se kterým pracuje mnoho dnes používaných databázových systémů jako Microsoft SQL Server, Oracle, MySQL a další. Webová aplikace potom spolupracuje s databázovým systémem s využitím SQL jazyka. Na základě zadání vstupu uživatelem je sestaven SQL dotaz, kde je vstup uživatele použit jako parametr nebo část SQL dotazu - taková technika je označována jako dynamické sestavování SQL dotazů. Vstup uživatele by měl být validován takovým způsobem, aby nebylo možné zpracovat vstup obsahující škodlivý kód, který by byl následně databázovým systémem vykonán. O tom, zda je aplikace zranitelná útokem SQL injection, rozhoduje kvalita validace a ošetření vstupu uživatele.

Podstatou útoku SQL injection je podvržení příkazů nebo jakékoli jiné neautorizované zasažení do SQL dotazu, který následně vykoná cílový systém řízení báze dat. K útoku je využíván vstup aplikace, který aplikace dále využívá pro sestavení SQL dotazu. K očekávané hodnotě je přidán škodlivý kód, jehož provedení může být okamžité, zde se jedná o přímý útok, nebo je škodlivý kód proveden později při jiné akci aplikace. Útok je úspěšný v případě nedokonalé validace nebo absence validace vstupů aplikace, které pocházejí z nedůvěryhodných zdrojů, jakým může být například HTTP požadavek.

Jedná se o útok na datové úložiště, ke kterému má aplikace přístup. Škodlivý kód je tak prováděn se stejným oprávněním, jakým disponuje napadená aplikace. Útok je veden přes jednu z vrstev vícevrstvé aplikace, která nemá implementovánu dostatečnou ochranu proti tomuto útoku. U webových aplikací se typicky jedná o selhání bezpečnosti na úrovni aplikační vrstvy.

Pro příklad lze uvést webovou aplikaci, která ve své veřejné části umožňuje procházet seznam zaměstnanců a zobrazovat jejich veřejné profily. Profil zaměstnance s interním identifikátorem 10 lze zobrazit na URL:

```
http://example.com/public/profile?id=10
```

Po přijetí požadavku aplikace načte hodnotu parametru URL a použije ji k sestavení SQL dotazu, jehož výsledkem budou informace k poptávanému profilu. Následuje typická ukázka zdrojového kódu aplikace, kde není ošetřena hodnota vstupu, ze kterého je dále sestaven SQL dotaz:

```
$r = mysql_query("SELECT * FROM emp WHERE id = $_GET[id]");
while($r && $row = mysql_fetch_array($r))
{
  $contentBuffer .= " ID: ".$row['id']." - ". $row['name'];
}
if ($r == 0)
  $contentBuffer .=  "Zadny zaznam.";
```

V uvedeném případě je útočníkovi umožněno přímým způsobem ovlivňovat podobu SQL dotazu, který bude předán databázovému systému. Takovou chybu lze dále zneužít například k získání všech dat, uložených v dané databázi. V následujících kapitolách jsou podrobně popsány v praxi používané způsoby a postupy zneužití slabiny SQL injection.

2.1 Kategorizace technik útoků

Útoky SQL injection mohou mít mnoho podob a mnoho možných způsobů provedení. Vždy záleží na míře zabezpečení cílové architektury, kde vyšší míra zabezpečení vyžaduje větší kreativitu útočníka. Kategorizace SQL injection útoků není v literatuře jednoznačná a v detailech se různí. V této kapitole bude uvedeno obecné rozdělení bez uvedení podkategorií, které často bývají zaměňovány a nemají ustálené označení. Konkrétní ukázky a různé varianty útoků spadajících do jednotlivých kategorií jsou uvedeny v kapitole 2.2.

Techniky útoků typu SQL injection mohou být rozděleny podle způsobu provedení útoku na následující:

Tautologie - útočník se snaží modifikovat cílový SQL dotaz takovým způsobem, aby podmínka WHERE cílového dotazu byla vždy pravdivá. Využívá se přidání pravdivé disjunkce typicky za účelem neautorizovaného získání dat nebo překonání autentizace.

Využití chybových výpisů - útok spočívá v úmyslném vyvolávání chyb databázového systému typicky za účelem získání přehledu o struktuře cílové databáze z poskytnutého popisu vyvolané chyby. Tyto útoky jsou uváděny pod názvem **error-based**.

Přiložení vlastního dotazu - útočník je schopný vložit oddělovač dotazů a přiložit vlastní dotaz, který je cílovým databázovým systémem proveden. Tato technika bývá označována jako **Stacked Queries** nebo také jako **Piggy-backed Queries**.

Využití konstrukce UNION - při útoku je využívána konstrukce UNION umožňující sjednocení výsledků dotazu za účelem efektivního získávání dat z cílové databáze.

Využití uložených procedur - technika bývá často součástí komplexnějších útoků a spočívá ve využití uložených procedur cílového databázového systému. Lze využít již existující uložené procedury nebo si pro další útok vytvořit procedury vlastní.

Maskování injekce - útočník změní podobu injekce takovým způsobem, že není odhalitelná případným bezpečnostním systémem filtrujícím požadavky obsahující běžně používané injekce. Při útoku je využíváno různého kódování hodnot (například do jejich hexadecimální reprezentace) nebo jsou také maskovány mezery v injekci pomocí blokových komentářů. Možnosti maskování injekce se liší pro různé databázové systémy, princip útoku však vždy spočívá v nalezení alternativ odmítnutých injecký.

Odvozování reakcí aplikace - technika je v literatuře často označována pod názvem **Inference** a její podstatou je vynucování změny chování cílového databázového systému. Aplikace může zakrývat do jisté míry reakce databázového systému, v některých případech však není zakrytí úplné a lze podle odpovědi rozpoznávat úspěšnost injekce. Pokud jsou zakryty chybové výpisy cílového databázového systému, tak se jedná o techniku **blind**. Specifickou variantou techniky **blind** jsou případy, kdy lze rozpoznávat pouze reakce na pravdivost podmínky WHERE, a taková varianta bývá označována jako **blind-boolean**. Další technikou odvozování reakce aplikace je technika **time-based**, která spočívá ve vkládání příkazu pro zpoždění odpovědi cílového databázového systému a následného porovnání času odpovědi aplikace s jejím běžným časem. Ukázky a podrobnější popis technik odvozování reakcí aplikace je uveden v kapitole 2.5.2.

Později vykonané injekce - útočník nemusí využít k útoku pouze jednu akci aplikace. Například u akcí, kde je prováděno vkládání hodnoty vstupu do databáze, útočník vloží injekci, která bude vykonána až při jiné akci, kdy si aplikace z databáze zjistí uloženou hodnotu a použije ji při dynamickém sestavení SQL dotazu. Typickým příkladem této techniky bývá situace, ve které útočník vloží místo jména uživatele injekci a injekce se vykoná v jiné akci, kde aplikace na základě jména sestavuje SQL dotaz pro zobrazení záznamů vztahujících se k danému jménu. Popsaná technika bývá v literatuře uváděna pod názvem **Second Order SQL Injection**.

Výše uvedené techniky bývají často kombinovány a málokdy si útočník vystačí pouze s jednou technikou [9]. Útoky SQL injection je možné rozdělit také podle způsobů získávání informací a dat z cílové databáze do následujících třech kategorií:

Inband - Jedná se o nejpřímější útok s využitím slabiny SQL injection. Aplikace vrací data stejnou cestou, jakou byl obdržen vstup, ze kterého byl sestaven SQL dotaz. V případě webové aplikace se může jednat o situaci, kdy na základě vstupu předaného přes HTTP požadavek je vrácen výsledek v odpovědi na obdržený požadavek.

Out of band - Informace a data z cílové databáze jsou vráceny jinou cestou než tou, přes kterou bylo provedeno podvržení příkazů. Může se jednat o případ, kdy je podvržený příkaz poslán v HTTP požadavku a výsledek je vrácen přes e-mail.

Inferential - Útočníkovi nejsou vráceny data přímo, ale i tak je možná jejich rekonstrukce například podle reakce serveru na podvržené injekce [5].

2.2 Detekce a varianty zranitelností

Pro ověření zranitelnosti aplikace je možné provést revizi zdrojových kódů, prověřit validace a použití vstupů aplikace. Mnohdy však nemusí být k dispozici potřebná dokumentace nebo je cílová architektura informačního systému natolik rozsáhlá, že není možné ve stanoveném termínu provést testování typu white-box. Často se tak provádí pouze testování typu black-box, což může vést k přehlédnutí slabiny, protože odhalení některých variant vyžaduje hlubší znalost problematiky SQL injection. Teoreticky může existovat také slabina, kterou nebude možné odhalit testováním typu black-box, a to ani s využitím pokročilých technik detekce, protože pro danou akci nebude možné odvodit odlišnosti v reakcích cílového databázového systému.

Lze rozlišovat různé varianty a techniky detekce SQL injection slabin a to například na základě obsahu odpovědi, specifické reakce pro existující záznam v databázi nebo podle chování aplikace při chybě vyvolané například nesprávnou syntaxí SQL dotazu.

Základní technikou pro odhalení slabiny SQL injection může být záměrné poškození syntaxe cílového SQL dotazu. Například v případech, kde aplikace vypisuje záznamy z databáze na základě číselného identifikátoru, je možné zaměnit číselnou hodnotu za řetězec. Pro databázový systém MySQL bude vrácen následující chybový výpis:

```
1054: Unknown column 'retezec' in 'where clause'
```

V praxi se často zkouší znaky apostrofu nebo dvojitých uvozovek, které mají pro mnoho databázových systému speciální význam ukončení řetězce. Neočekávané ukončení řetězce naruší syntaxi výsledného SQL dotazu a lze očekávat reakci aplikace na chybu databázového systému.

Náchylnost vykonání operace

Výše uvedená technika však není použitelná v případech, kdy je chyba databázového systému zakryta a není možné jednoznačně rozpoznat reakci na chybu. Například je po výskytu chyby vrácena stejná odpověď jako pro neexistující záznam. V takových případech se jedná o variantu útoku SQL injection s přívlastkem **blind**. Odhalení blind variant bývá náročnější a spočívá ve vyhledání injekce, která slabinu prokáže. V případě číselných datových typů se může jednat o jiné vyjádření stejné hodnoty, například následujícím způsobem:

```
http://www.example.com/?page=multi-int&entry=3-2
http://www.example.com/?page=multi-int&entry=6-5
```

V ukázce je testovaným vstupem parametr entry v URL. Pro jiné datové typy je možné využít jiné operace, u řetězců například konkatenace. Pokud je pro oba případy vrácen stejný záznam z databáze, je velmi pravděpodobné, že se jedná o slabinu SQL injection. Mohlo by se stát, že aplikace vrací daný záznam i po chybě nebo pokud není nalezen jiný požadovaný záznam a v takových případech je nutné otestovat parametr entry stejným způsobem pro jinou hodnotu.

Náchylnost modifikace podmínky

Další technikou může být pokus o modifikaci podmínky WHERE cílového SQL dotazu. Je možné se pokusit o tautologii vhodnou injekcí. Pro akce, kde aplikace vrací výpis obsahu tabulky na základě vstupu, je možné sledovat rozdílné výpisy pro následující URL:

```
http://www.example.com/?page=multi-int&entry=2 or 1=1
http://www.example.com/?page=multi-int&entry=2 and 1=1
http://www.example.com/?page=multi-int&entry=2 and 1=2
```

Pokud aplikace reaguje na injekci v parametru entry, budou pro první URL vráceny všechny položky z cílové databáze, pro druhé URL pouze jedna položka pro hodnotu 2 a ve třetím případě nebude vrácena žádná položka. V ukázce se jedná o vstup pro číselný datový typ, pro jiné datové typy budou injekce obdobné, je však nutné dát pozor na správné ukončení řetězců:

```
http://example.com/?page=multi&entry=zakaznik1' or 'x'='x
http://example.com/?page=multi&entry=zakaznik1' and 'x'='x
http://example.com/?page=multi&entry=zakaznik1' and 'y'='x
```

Základní techniky maskování injekce

V některých případech může aplikace nebo nasazený IDS systém zahazovat požadavky obsahující známé a běžně používané injekce. Můžou být například odmítány všechny hodnoty obsahující mezery. V takových případech lze podobu injekce modifikovat s použitím blokových komentářů následujícím způsobem:

```
http://example.com/?page=multi-int&entry=2/**/or/**/1=1
```

Existuje mnoho způsobů jak obcházet různé nedokonalé validace a vždy záleží na konkrétní variantě slabiny. K těmto technikám obcházení nedokonalých validací přijde vhod znalost typu a verze cílového databázového systému. Oblíbenou obranou vývojářů bývá zakázání apostrofu v hodnotě vstupu. Taková validace lze obejít pomocí funkcí databázových systémů pro sestavení řetězce z jeho hexadecimální reprezentace, čímž odpadá nutnost použití apostrofu v některých konstrukcích. V následující ukázce pro databázový systém MySQL bude vrácen obsah souboru *C:\file.ini* bez použití apostrofů v parametru funkce *LOAD_FILE()*:

```
SELECT CAST( LOAD_FILE( 0x433A5C66696C652E696E69 ) AS CHAR )
```

Ověření nálezu zranitelnosti

Pro ověření slabiny SQL injection může výborně posloužit technika využívající konstrukci UNION. S využitím konstrukce UNION je možné do výsledků vráceného databázovým systémem přidat další záznamy. Je tak možné vytvořit injekci pro přidání vlastního záznamu do výpisu z databáze. Následující ukázka injekce demonstruje příklad využití konstrukce UNION:

```
') UNION SELECT CONCAT (0x71666778786772), null;--
```

V uvedeném příkladu je záměrem injekce přidání vlastního řádku k výsledkům vyhledávání v databázi na základě parametru URL. Vlastní řádek bude mít dva sloupce, kde v prvním sloupci bude řetězec qfgxxgr odpovídající hexadecimální reprezentaci *0x71666778786772*.

Provedení injekce lze rozpoznat podle přítomnosti řetězce *qfgxxgr* v odpovědi aplikace. Ve variantách, kde aplikace zobrazuje pouze první záznam z výsledku, může přijít vhod použití konstrukce pro omezení výsledku například na druhý záznam. Tuto techniku je však vhodné automatizovat kvůli nutnosti odhadnout správné ukončení podmínky WHERE a počet sloupců pro výsledek cílového SQL dotazu.

Výše uvedený popis technik detekce není zdaleka kompletní a existuje spoustu dalších technik, jejichž popis přesahuje rozsah této práce. Pro detekci jsou použitelné i techniky popsané v kapitole 2.5.2, kde jsou uvedeny postupy zneužití, které se v určitých případech dají využít také pro hledání slabin SQL injection.

2.3 Náchylnost vstupů aplikace

Náchylnými místy jsou všechny vstupy aplikace, které jsou dále použity pro sestavení SQL dotazu. Základem pro provedení komplexní analýzy zranitelnosti aplikace je znalost všech vstupů aplikace, které jsou k sestavení SQL dotazů používány. Ne vždy je však k dispozici kvalitní dokumentace testované aplikace, a tak je nutné testovat všechny možné místa, které se v praxi využívají jako vstupy aplikace. K identifikaci potenciálně zranitelných vstupů pomůže pozorování a analýza komunikace mezi klientem a serverem s využitím specializovaných nástrojů jako je například nástroj Fiddler. Zajímavé mohou být také vstupy skrytých akcí, které se již nepoužívají, neboť sloužily například pro ladící účely. Takové akce mohou být u webových aplikací viditelné v komentářích zdrojových kódů poskytovaných klientovi.

Pro předávání hodnot mezi klientem a serverem jsou u webových aplikací využívány nejčastěji parametry URL a formulářové prvky. Některé varianty předávání hodnot můžou být natolik specifické, že je nelze detekovat automatizovaným nástrojem. Může se jednat například o část URL za znakem mřížky, kterou lze použít pro práci s technologií AJAX s využitím vlastního nestandardního formátu vstupů.

Všechny vstupy formuláře nemusí být na první pohled zřejmé. Prvky formuláře lze rozdělit z pohledu detekce vstupů webové aplikace do dvou kategorií:

- **viditelné prvky formuláře** - i když jsou ošetřeny vstupy textových polí, je možné, že programátor nepočítal s validací hodnoty prvků select a tuto hodnotu přidává bez validace do SQL dotazu
- **skryté prvky formuláře** - programátoři očekávají, že hodnota skrytého prvku nebude modifikována, a tak mnohdy tyto hodnoty nevalidují

Zranitelným vstupem webové aplikace můžou být také v podstatě všechny pole v hlavičce HTTP požadavku. Některé aplikace mohou například logovat verze a typy prohlížečů, kterými uživatelé k aplikaci přistupují, a provádět tak vložení obsahu pole User-Agent do databáze. Zranitelné bývají i vstupy v poli Cookie, což se může stát například v situacích, kdy aplikace identifikuje sezení podle některé z hodnot v tomto poli a následně na základě této hodnoty sestavuje SQL dotaz pro načtení uživatelského nastavení aplikace uloženého v databázi.

Často se stává, že je aplikace implementována s využitím nějakého frameworku[1], což lze rozpoznat stejnými reakcemi při útocích na různé vstupy aplikace. V tomto případě se vyplatí hledat programátorská řešení, která vybočují z daného frameworku - může se jednat

[1]Framework je soubor knihoven a předpřipravených řešení, které lze využít při implementaci vlastního software.

o spojení různých technologií, například využití technologie AJAX pro lokální funkcionality aplikace. Je také velmi pravděpodobné, že formuláře s odlišným vzhledem a začleněním do stránky nejsou validovány stejným způsobem, jako tomu je u ostatních formulářů.

Pro komplexní otestování aplikace na přítomnost SQL injection slabiny je nutné vytvořit seznam vstupů a to pro každou akci aplikace zvlášť. Dále je nutné otestovat každou položku vytvořeného seznamu. Protože může být množina vstupů aplikace pro každou její akci rozsáhlá, využívá se často automatizovaného testování.

2.4 Obrana

Obranu proti útokům typu SQL injection a zredukování dopadu útoku lze řešit na více úrovních. Na všech úrovních je pak vhodné zavedení logování a pravidelná kontrola logů. Příhodné je i nastavení varování administrátora při výskytu chybových stavů. Zefektivnění detekce a pohotovější reakci na probíhající útok lze řešit nasazením vhodného SIEM systému.

Kontrola vstupu a jeho ověření, zda neobsahuje škodlivý kód, by měla provádět aplikace, která sestavuje a pokládá daný SQL dotaz databázovému systému. Kromě obrany na této úrovni je však možné útokům předcházet i na dalších vrstvách architektury a to především u následujících:

Bezpečná komunikace - zavedení kryptografické ochrany komunikace mezi klientem a serverem znemožňující její čitelnost a neautorizovanou modifikaci. Je nutné věnovat pozornost správné konfiguraci použitého kryptografického řešení.

Konfigurace serveru - mnoho webových serverů není správně nakonfigurováno, což se projevuje například poskytováním zneužitelných informací o použitých technologiích a jejich verzí. Se znalostmi o použité technologii a její konkrétní verze je útočník schopen lépe cílit útok na daný systém.

Databázový systém - žádoucí je přenastavení výchozích hesel systémových účtů, minimalizace a rozdělení oprávnění pro uživatele jednotlivých aplikací nebo jejich konkrétních akcí. Databáze by měla být pravidelně zálohována.

Aplikační firewall - je vhodné kontrolovat formát příchozího požadavku, například se může jednat o validaci SOAP formátu a případný nevyhovující požadavek zahodit. Některé aplikační firewally také nabízí pokročilé možnosti detekce SQL injection útoků a poskytují předdefinovaná pravidla pro blokování podezřelých požadavků. Pokročilé aplikační firewally poskytují také obranu technikou učení, kdy je během režimu učení sestaven model, podle kterého jsou následně v ostrém režimu vyhodnocovány a případně blokovány podezřelé požadavky. Aplikační firewall je v mnoha případech schopný odrazit útok automatizovaného nástroje ještě před tím, než nástroj vyhodnotí danou slabinu a zjistí správný formát injekce.

Aplikační proxy a systémy odhalení průniku - existují nástroje, které dokážou vyhledávat v požadavcích podezřelé vstupy například na základě výskytu speciálních znaků v hodnotě vstupu. V některých případech, kdy je nutné ochránit rozsáhlou aplikaci bez validací vstupů, může být využita specifická aplikační proxy, která bude validaci vstupů provádět, což nevyžaduje zásah do zdrojových kódů aplikace. Tento přístup však nemusí být u všech případů správný a dostačující - například pokud

aplikační proxy vyhledává znaky se speciálním významem pro databázový systém a zároveň je vstup dosazen na místo numerické hodnoty ve výsledném SQL dotazu, bude útočníkovi umožněno vytvoření tautologie bez použití znaků se speciálním významem. Dále jsou k dispozici různé systémy odhalení průniku (označované jako IDS) a jejich aktivní varianty (se zkratkou IPS), které umožňují na základě definovaných událostí včas reagovat na zjištěný útok.

Obecné zásady ošetření vstupu

V samotné aplikaci by měla být pro každý vstup definována množina povolených hodnot, vůči které bude probíhat validace na straně serveru. Omezení a kontroly na straně klienta je mnohdy snadné obejít a nelze je považovat za dostačující. Je nutné provádět validace i těch hodnot, které byly poslány uživateli a neočekává se jejich změna, například hodnoty Cookie. U vstupu, kde je očekáván určitý datový typ, musí proběhnout kontrola, zda je vstupem skutečně očekávaný datový typ. Žádoucí je také kontrola délky vstupu. Po úspěšné validaci musí následovat překódování znaků se zvláštním významem pro sestavovaný SQL dotaz. Není možné pracovat dále s hodnotou, která nebyla validována. Neúspěšná validace vstupu by měla vyvolat chybový stav a veškeré chybové stavy by měly být ošetřeny vrácením obecné chybové stránky. Detaily chyby by neměly být obsaženy v odpovědi, ale měly by být logovány na serveru. Nelze se také spoléhat na dodržení očekávané posloupnosti akcí aplikace, kdy by nebyl kontrolován vstup, protože jeho kontrola měla proběhnout při akci předchozí.

Obrana na aplikační úrovni

Efektivní obranou proti útoku SQL injection může být použití parametrizovaných SQL dotazů. Jedná se o předpřipravené SQL dotazy s jedním nebo více parametry, do kterých jsou dosazeny hodnoty za běhu aplikace. Tyto hodnoty jsou dále používány takovým způsobem, že nehrozí jejich interpretace. Následuje ukázka implementace v jazyce PHP demonstrující použití parametrizovaného SQL dotazu pro databázový systém MySQL:

```
$st = $dbc->prepare("SELECT * FROM zak WHERE jmeno = ?");
$result = $st->execute(array($_GET['vstup']));
$row = $st->fetch()
```

V ukázce je nejdříve připraven SQL dotaz pro vyhledání všech zákazníků na základě jejich jména, které je později předáno databázovému systému zvlášť jako parametr předpřipraveného SQL dotazu. Hlavní komplikací pro nasazení parametrizovaných SQL dotazů u již hotových aplikací je nutnost kompletního přepsání spolupráce aplikace s databázovým systémem, což je pro rozsáhlé aplikace příliš nákladné řešení.

Využití specifických funkcí

Další možností obrany proti útokům typu SQL injection je ošetření vstupů s využitím funkce zajišťující kódování řetězcových literálů pro daný databázový systém. Nemusí být dostačující nahrazování a kontrola výskytu pouze jednoho znaku oddělujícího textový řetězec. Je nutné ošetřit všechny znaky se speciálním významem takovým způsobem, aby se zabránilo útoku SQL injection i v situacích, kdy vývojář dosadí vstup do SQL dotazu jako číselnou hodnotu nebo například jako název sloupce tabulky.

Populární programovací jazyky jako PHP poskytují jak funkce obecné, tak i funkce specifické odpovídající různým databázovým systémům. Pro příklad s využitím programo-

vacího jazyka PHP a databázového systému MySQL je možné použít funkci *mysqli::real_escape_string* pro nahrazení znaků se speciálním významem s využitím rozšíření mysqli, viz [18]. Před použitím těchto funkcí by mělo být samozřejmostí přečtení a pochopení příslušných dokumentací. Například zmíněná a v praxi často používaná funkce *mysqli::real_escape_string* neošetří dostatečně hodnoty vstupů, které jsou dosazeny na místo numerického datového typu v sestavovaném SQL dotazu, tedy bez obalení apostrofy, jak je vidět na následující ukázce:

```
$g =   $_GET["entry"]; // vstupem může být injekce 1 OR 1=1
$g = mysqli_real_escape_string($mysqli,$g);
$result = mysql_query("SELECT * FROM zak WHERE id = $g");
```

Alternativní řešení

Kromě výše uvedených možností, které jsou doporučovány například v oficiálním manuálu jazyka PHP, se vyskytují i neobvyklá řešení. Může se jednat například o vytvoření seznamu speciálních znaků, které bude nutné pro daný vstup aplikace nahrazovat. Rizikem tohoto řešení může být například nepochopení jiného vývojáře, který seznam použije pro jiný vstup, pro který nemusí být výčet speciálních znaků kompletní. V některých případech je možné daný vstup převést do jeho binární reprezentace nebo vypočítat jeho MD5 hodnotu a potom je v tomto tvaru porovnávat, jak naznačuje následující ukázka:

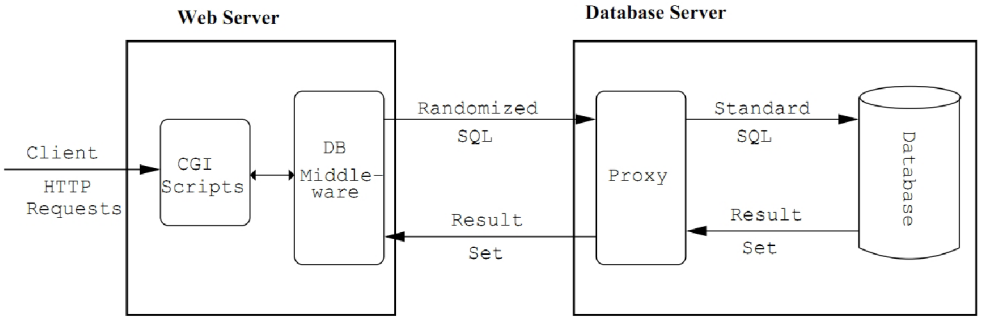
```
$where="WHERE md5(user)=".md5($name)
```

Odolnost aplikace nejen vůči útokům SQL injection může zvýšit využití již existujícího frameworku pro implementaci vlastní aplikace. Je však nutné zvážit výběr frameworku, ne všechny řešení poskytují ochranu proti útokům SQL injection. Důležité je i pochopení a dodržení konceptu zvoleného frameworku. Zvýšenou pozornost je nutné věnovat při implementaci vlastního rozšíření a úprav frameworku. Dále je vhodné sledovat aktualizace a zjištěné bezpečnostní chyby daného frameworku v rámci provozování aplikace.

Pokročilé metody obrany

Výše popsané techniky obrany před útokem SQL injection patří mezi základní. K pokročilým technikám patří například statická a dynamická analýza, kde jsou pro kontrolu výskytu škodlivého kódu využívány konečné automaty a rozbor sestaveného SQL dotazu na úrovni tokenů. Další zajímavou technikou je modifikace standardních příkazů daného databázového systému s využitím náhodných hodnot, což implementuje například nástroj SQLrand.

K celkovému zvýšení bezpečnosti pak může přispět i provedení penetračního testu a nezávislá revize zdrojového kódu implementované aplikace. Penetrační test prověří funkčnost bezpečnostních mechanizmů a může odhalit nedostatky ještě před samotným zpřístupněním aplikace z internetu.



Obrázek 2.1: Schéma fungování nástroje SQLrand, zdroj: [3]

2.5 Následky a postupy zneužití

Po odhalení SQL injection slabiny je útočník schopný zkopírovat všechna data v databázi, ke kterým má databázový uživatel přístup. Útočníkovi se nemusí nutně jednat o data související s účelem aplikace - mnoho uživatelů internetu používá stejné hesla do různých informačních systémů, a tak je na místě úvaha zneužití informačního systému za účelem získání přístupu do jiného informačního systému. I uložení hesla v jeho hash podobě (například MD5) nemusí být dostatečná ochrana pro jednoduchá hesla, protože existují nástroje pro převod hash podoby do otevřeného textu. Útočník může být dále schopen uložená data vymazat nebo je úmyslně modifikovat ve svůj prospěch nebo za účelem komplexnějšího útoku na danou službu.

2.6 Následky

Dopad útoku SQL injection závisí mimo jiné na přidělených právech uživatele databázového systému, se kterými webová aplikace do databázového systému přistupuje, a dalších komponent, které spolupracují s aplikací a databázovým systémem. Pokud má daný uživatel databázového systému práva spouštět příkazy operačního systému, potom tyto příkazy může spouštět i útočník a to s právy databázového systému. Například pro databázový systém MS SQL Server lze využít příkaz xp_cmdshell(), který spustí příkazovou řádku Windows a předá příkazy pro vykonání:

```
EXEC xp_cmdshell 'dir *.exe';
GO
```

Mnoho databázových systémů umožňuje čtení a zápis do souborů, ke kterým má databázový systém přístup. Útočník je tak schopný stáhnout soubory, které nejsou běžně dostupné. Může se jednat o například o soubor /etc/passwd. S využitím zápisu do souborů lze nahrávat na napadený server vlastní knihovny nebo přepisovat stávající soubory.

Slabina SQL injection může útočníkovi výborně posloužit pro provedení komplexnějšího útoku. Například u webových aplikací lze slabinu zneužít například k provedení útoku typu Stored XSS, kde útočník uloží do databáze svůj zdrojový kód a podvodný obsah, který se bude zobrazovat všem nic netušícím uživatelům napadené služby. Může se jednat o falešný přihlašovací formulář, který odesílá zadané údaje na jiný server. Při nedostatečném logování a kontrole logů nemusí být zásah útočníka detekován.

Kromě výše zmíněných dopadů jako je únik dat, může provozovateli aplikace hrozit poškození dobrého jména a následnou ztrátu klientů. Především u společností pracujících s citlivými údaji, by bylo samotné nalezení a zveřejnění SQL injection slabiny nežádoucí a mohlo by mít negativní dopad na pověst společnosti.

2.7 Postupy zneužití

Prvním krokem vedoucím k možnému zneužití slabiny SQL injection je nalezení správné syntaxe podvržených SQL konstrukcí a příkazů. Cílem je navázat nebo správně ukončit předchozí posloupnost konstrukcí a příkazů takovým způsobem, aby bylo možné vkládat příkazy vlastní. Za podvrženými příkazy pak stačí napsat oddělovač řádkového komentáře, což vyřeší problém s následující posloupností původních konstrukcí. Situace je o něco komplikovanější v případech, kdy se jedná o víceřádkový SQL dotaz - zde je nutné vyhledat specifickou techniku útoku pro danou variantu, například je možné za podvržené příkazy připravit začátek dalšího SQL dotazu.

Ne vždy je však možné vkládat vlastní příkazy, protože cílová architektura nemusí povolovat více než jeden příkaz v jednotlivé komunikaci aplikace s databázovým systémem. Útoky využívající vkládání vlastních příkazů jsou označovány názvem **Stacked SQL Injection**. Jedná se závažnou variantu slabiny SQL injection, protože útočník není limitován původním dotazem, což mu dává více volnosti a možností zneužití slabiny.

Využití chybových výpisů

Při manuálním hledání správné syntaxe injekce jsou velmi užitečné chybové výpisy cílové databáze. Tyto chybové výpisy obsahují alespoň částečné informace, kde je chyba syntaxe, a prozradí jména tabulek a sloupců použitých v daném SQL dotazu. Například pro databázový systém Microsoft SQL Server je možné zneužít chybové výpisy k postupné enumeraci sloupců tabulky s využitím konstrukce HAVING, jak naznačuje následující posloupnost injekcí:

```
1 HAVING 1=1 --
1 GROUP BY <sloupec1> HAVING 1=1 --
1 GROUP BY <sloupec1>,<sloupec2> HAVING 1=1 --
```

Zjištění typu a verze databázového systému

Další užitečnou znalostí je informace o tom, jaký je použit databázový systém. Pokud jsou k dispozici chybové výpisy, je odvození databázového systému snadné. Chybový výpis lze získat i u jiných vstupů aplikace, kde sice není útok SQL injection možný, ale validace je nedokonalá. Jestliže je očekáváno číslo a vstup je validován regulárním výrazem [0-9]*, je možné vyvolat chybovou zprávu příliš velkým číslem. Pokud není k dispozici chybový výpis, dá se využít rozdílností databázových systémů. Například zápis konkatenace řetězců může posloužit k vyloučení některých databázových systému.

Útočník musí posbírat co nejvíce informací o cílové databázi. Typickým záměrem útočníka je přečtení nebo modifikace uložených dat. Obtížnost a způsob získávání dat z cílové databáze se může lišit a závisí na typu cílové databáze. K běžně používaným systémům řízení báze dat existují podrobné a veřejně dostupné dokumentace. Po zjištění typu cílové databáze je vhodné daný databázový systém nastudovat. U některých databázových

systémů je možné určit i konkrétní verzi. Pokud bude následující SQL dotaz cílovým databázovým systémem vykonán bez chyby, je velmi pravděpodobné, že se jedná o databázový systém MySQL verze 3.23.02 nebo vyšší:

```
SELECT * FROM customers WHERE id=1 AND /*!32302 1 */
```

Kromě uvedeného je možné u některých variant slabin SQL injection pro databázový systém MySQL použít efektivnější řešení s využitím speciální funkce @@version:

```
http://example.com/?entry=1 UNION SELECT @@version, null
```

Využití informačních schémat

Jedním z prvních kroků bývá zjištění cílového uživatele databáze a jeho oprávnění nebo zjištění jména cílové databáze. Dále potom výčty rolí a ostatních uživatelů databáze, jejich hesel (ve formě hash) a oprávnění. Následuje výčet tabulek a jejich sloupců. Dalším krokem je zjištění hodnot v řádcích tabulek. Existuje několik technik jak získávat tyto informace, zde budou uvedeny nejpoužívanější z nich. Možnost použití uvedených technik závisí na konkrétní variantě SQL injection slabiny a také na podpoře ze strany cílového systému řízení báze dat.

Některé databázové systémy zpřístupňují různá metadata obsahující jméno databáze, existující tabulky, oprávnění a podobně. Pro MySQL tak lze snadno získat seznam tabulek v databázi následujícím SQL dotazem:

```
SELECT table_name FROM information_schema.TABLES;
```

Další možnosti zjištění struktury databáze

Jména tabulek lze také hádat pomocí slovníků často používaných jmen tabulek. Například pro databázový systém PostgreSQL poslouží funkce *PG_SLEEP()*, kde argumentem bude čas v jednotkách sekund. Lze tedy sestavit podmínku, která zjistí, zda existuje v cílové databázi tabulka se jménem *uzivatele*:

```
SELECT CASE WHEN
  (SELECT 1 FROM uzivatele LIMIT 1) > 0 THEN
  PG_SLEEP(5)
END;
```

Pokud v databázi existuje hádané jméno tabulky a dotaz je pokládán s dostatečným oprávněním k testované tabulce, bude odpověď zpožděna o 5 sekund. Místo funkcí pro zpoždění odpovědi se také někdy využívají funkce pro benchmarky. V předchozí ukázce je využita konstrukce pro podmínku, která může mít při detekci a zneužití slabiny širší využití.

Konstrukce podmínky má praktické využití také pro potvrzení nálezu slabiny SQL injection. Pro databázový systém MySQL lze pozorovat rozdílnost reakce testované aplikace na následující příkazy:

```
SELECT IF(1=1, SLEEP(5), 2)
SELECT IF(1=2, SLEEP(5), 2)
```

Získávání dat z databáze

V případech, kdy jsou dotazované hodnoty vráceny v odpovědi, lze využít konstrukce UNION. S touto konstrukcí je možné získávat data i z jiných tabulek v databázi. Aby byla injekce úspěšná, je nutné správně ukončit podmínku WHERE a doplnit správný počet sloupců. Následuje ukázka injekce s využitím konstrukce UNION, na základě které je vrácen název uživatele databázového systému:

```
x' UNION SELECT CONCAT(CAST(CURRENT_USER() AS CHAR)),NULL ;--
```

Někdy se může stát, že je aplikace vypisuje pouze první záznam výsledku vráceného databázovým systémem. V takovém případě je možné využít konstrukce LIMIT pro omezení výsledku na určitý záznam. V následující injekci, je výsledek omezen na druhý podvrhnutý záznam výsledku SQL dotazu:

```
x' UNION SELECT CONCAT (0x716667) ,null LIMIT 1,2 ;--
```

Konstrukci UNION je možné použít i pro hledání správného datového typu jednotlivých sloupců. Pokud je možné rozpoznat reakce aplikace na chybu a předchozí injekce s konstrukcí UNION proběhla bez chyby, bude možné odvodit datové typy sloupců postupnou záměnou hodnot NULL za hodnoty konkrétních datových typů. Pro každý sloupec zvlášť se tipují hodnoty různých datových typů a po bezchybné reakci aplikace je možné přejít ke sloupci dalšímu.

Určení počtu sloupců může být provedeno technikou ORDER BY. Principem techniky je vkládání příkazu pro seřazení výsledku podle určitého sloupce. Za příkaz ORDER BY se uvede číslo sloupce, podle kterého má být seřazení provedeno. Pokud je zvolené číslo v injekci větší, než je skutečný počet sloupců v cílové tabulce, lze očekávat chybovou reakci aplikace. Naopak pro existující sloupec je vrácen seřazený výsledek dotazu.

Rekonstrukce dat podle reakcí aplikace

Další možností jak získávat informace z databáze je využití odvozování hodnot po znacích. V praxi je tato technika často využívána pro útoky typu **blind-boolean**. Důležitou roli v tomto útoku sehrají funkce pro získávání ordinální hodnoty v kombinaci s funkcí vracející jeden znak z vráceného řetězce. Tyto výčty se nejčastěji provádějí hrubou silou a jsou automatizovány. V následující ukázce testována ordinální hodnota prvního znaku názvu uživatele databázového systému.

Pokud bude název uživatele začínat znakem s ordinální hodnotou 117, je možné očekávat reakci aplikace pro existující záznam s hodnotou *y*. V opačném případě lze očekávat reakci pro neexistující záznam v cílové databázi. Výsledná injekce pro databázový systém MySQL bude mít následující podobu:

```
y' AND ORD(MID(CAST(CURRENT_USER() AS CHAR),1,1))=117
AND 'x'='x
```

Ve výše uvedeném přehledu technik zneužití slabiny SQL injection jsou popsány nejčastěji používané techniky, jejichž výčet však není kompletní. Postupů zneužití je mnoho stejně jako technik detekce slabiny nebo způsobů obrany. Detailní popis všech možných a v praxi používaných technik včetně jejich variant přesahuje rámec této práce. Znalost výše uvedených technik může v mnoha situacích napomoct detekovat slabinu SQL injection.

Techniky, které byly popsány v této kapitole, jsou v praxi běžně používané. Vždy je nutné nalézt správný formát injekce, aby bylo možné danou techniku využít. Nabízí se tedy možnost výše uvedené postupy automatizovat a vyhledávat správný formát výsledné injekce například podle slovníkových definicí.

Kapitola 3

Popis a přehled současných nástrojů

Pro detekci slabiny SQL injection je možné využít automatizované nástroje. V současné době existuje mnoho nástrojů různých kvalit. Většina nástrojů umožňující komplexní otestovaní bezpečnosti aplikace má integrovánu i analýzu výskytu slabin typu SQL injection. Dále jsou k dispozici nástroje, které se specializují pouze na slabiny typu SQL injection. Pro detekci slabiny SQL injection existují i doplňky pro prohlížeče, jako je například SQL Inject Me pro prohlížeč Firefox [14].

Motivací pro automatizaci detekce zranitelnosti SQL injection je systematické otestování všech typických vstupů aplikace. Použití automatizovaného nástroje může zkrátit čas potřebný pro otestování bezpečnosti aplikace a odhalit zranitelné vstupy aplikace, které člověk přehlédne. Automatizované nástroje mnohdy navíc poskytují funkcionality pro následné využití zjištěné SQL injection slabiny - například enumeraci tabulek v databázi pomocí technik, které jsou obtížné pro manuální provedení.

Automatizované nástroje však nemusí být schopny pojmout session management testované aplikace. I přes to, že některé nástroje umožňují definovat přihlašovací sekvenci, tak může být pohyb v aplikaci natolik složitý, že výsledkem bude neustálé přihlašování a následně neúspěšný pokus o pohyb v aplikaci, který povede k odhlášení (například zneplatnění identifikátoru sezení na straně serveru). Zda nástroj pracuje správně a je schopný se ve webové aplikaci pohybovat je vhodné ověřit s využitím proxy, kde je možný náhled na zaslané požadavky a jejich odpovědi.

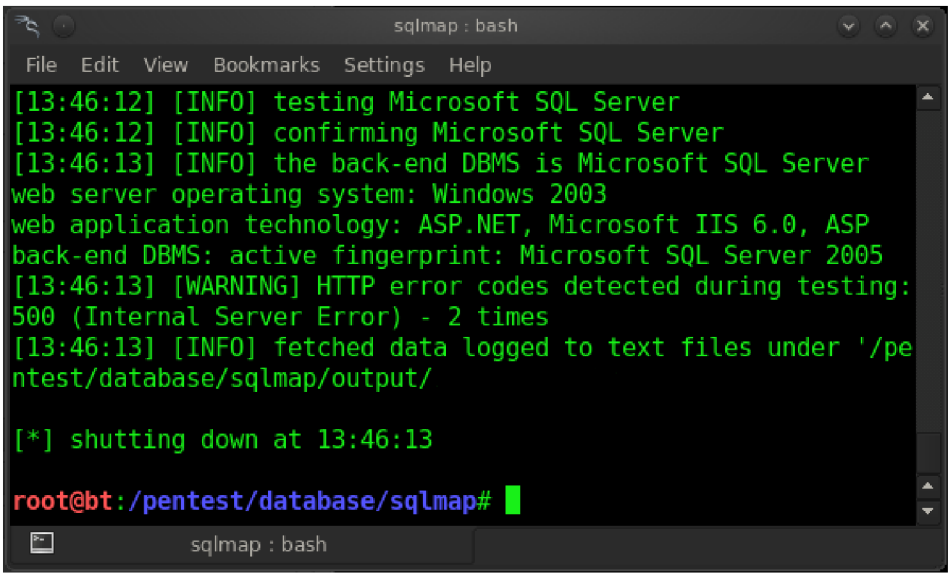
Dalším problémem je identifikace vstupů aplikace. Například v případech, kdy je v aplikaci použitý nestandardní formát parametrů v URL, ze kterého pak aplikace svým specifickým způsobem parametry získává. Také nemusí být úspěšná injekce odhalitelná automatizovaným nástrojem - například v situaci, kdy nástroj vyhodnocuje úspěšnost injekce na základě porovnávání velikosti obsahu odpovědí a zároveň aplikace vrací vždy jinak velkou odpověď. Níže jsou uvedeny některé dnes používané nástroje pro automatizovanou detekci SQL injection. Používaných nástrojů existuje mnoho, jejich výčet a podrobnější přehled je však nad rámec této práce. Byl tedy zvolen jeden z nejpoužívanějších nástrojů sqlmap a nástroj využívaný útočníky sqlninja. Další přehled nástrojů je uveden například na [6].

3.1 Nástroj sqlmap

Jedná se o projekt s otevřeným zdrojovým kódem určený pro detekci a následné využití slabiny SQL injection [2]. Nástroj je implementován v jazyce Python a je poskytován s licencí GNU General Public License [8].

Nástroj podporuje několik databázových systému včetně MySQL, Oracle, PostgreSQL a Microsoft SQL Server. K dispozici je několik technik útoku SQL injection a následně podpora eskalace práv procesu databázového uživatele. Nástroj umožňuje enumeraci uživatelů, oprávnění, rolí, databází, tabulek a sloupců. Nechybí ani možnost získání dumpu[1] tabulek. K tomuto populárnímu konzolovému nástroji existuje mnoho podrobných návodů a videí. Nástroj sqlmap je standardně obsažen v distribucích BackTrack [11].

Nástroj sqlmap je jedním z nejpokročilejších nástrojů pro práci s SQL injection slabinou, ale i přesto nepokrývá všechny možné varianty SQL injection slabin. Při experimentování s nástrojem bylo v rámci této práce zjištěno, že nástroj nedetekoval zranitelný vstup, který cílová aplikace použije ve víceřádkovém SQL dotazu jako parametr funkce cílového databázového systému PostgreSQL. Pro následnou práci s existující SQL injection slabinou bylo v tomto konkrétním případě nutné využít složitější možnosti nástroje jako psaní skriptů pro modifikaci výsledné injekce před posláním požadavku.



Obrázek 3.1: Výpis nástroje sqlmap obsahující zjištěné informace o cílové architektuře

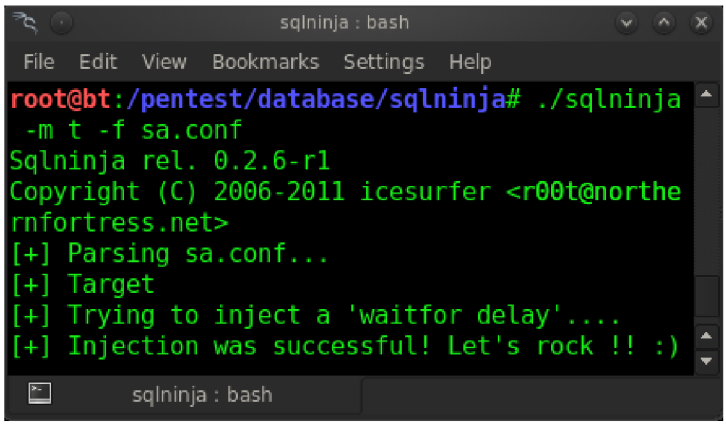
3.2 Nástroj Sqlninja

Sqlninja je nástroj zaměřený především na zneužití slabin typu SQL injection [10]. Nástroj se specializuje na databázový systém Microsoft SQL Server a poskytuje všechny běžné techniky potřebné pro převzetí kontroly nad databází a případně i nad serverem.

Jedná se o nástroj napsaný v jazyce Perl a poskytovaný pod licencí GNU General Public License. Sqlninja je primárně vyvíjený a testovaný na platformě UNIX. Na stránkách nástroje je k dispozici ucelená a poměrně přehledná dokumentace popisující jednotlivé

[1] Dumpem je myšlen výpis obsahu databázových objektů nebo celé databáze.

funkcionality a možnosti nastavení. Pro práci s nástrojem je nutné znát slabinu a definovat správný formát injekce. Po úspěšném ověření přítomnosti slabiny nástroj nabídne možnost použití technik pro obejití případného IDS systému, jako je například náhodné kódování URL nebo používání hexadecimální reprezentace hodnot v posílaných injekcích. Kromě běžných technik jsou k dispozici i pokročilé metody pro eskalaci oprávnění, nástroj například nabízí možnost zneužití bezpečnostní chyby CVE-2010-0232[2].



Obrázek 3.2: Výpis nástroje Sqlninja po ověření zranitelnosti vstupu

Z popsaných nástrojů umožňuje detekci slabiny SQL injection pouze nástroj sqlmap. Během experimentů se však ukázalo, že i tento nástroj není schopen detekovat některé obtížně zjistitelné slabiny SQL injection. V rámci praktické části této práce byl tedy vytvořen nástroj, který se specializuje na problematiku detekce obtížně zjistitelných slabin.

[2]Detaily k chybě CVE-2010-0232 lze nálezt na [15]

Kapitola 4

Nástroj pro detekci zranitelnosti SQL injection

V rámci praktické části bude implementován nástroj pro detekci zranitelnosti SQL injection ve webové aplikaci. Jedná se o netriviální aplikaci, proto bude kladen důraz nejen na kvalitní implementaci, ale i na detailní analýzu problému a návrh aplikace.

Aby bylo možné implementovaný nástroj přizpůsobit různým možnostem a variantám zranitelností SQL injection, bude vytvořena webová aplikace demonstrující různé varianty slabin SQL injection, která umožní jeho regresní testování. Webová aplikace bude poskytovat možnosti generování dynamické délky odpovědi a jiné další možnosti specifického chování pro simulaci obtížně detekovatelných zranitelností, které se v praxi vyskytují.

Před samotným návrhem nástroje byla provedena analýza jiných v současné době používaných nástrojů pro detekci SQL injection slabin a obecně problematiky automatizace procesu hledání SQL injection zranitelností. Z provedené analýzy vyplynuly požadavky na funkčnost vytvořeného nástroje, které budou během návrhu a vývoje brány v potaz. Podle zadání se bude vytvořený nástroj specializovat na detekci slabin, nebudou implementovány funkcionality pro zneužití nalezených slabin.

4.1 Funkční požadavky

Vstupem nástroje bude textový soubor obsahující HTTP požadavek s označeným místem pro prověření na výskyt SQL injection slabiny. Na místě vstupu pro prověření bude vložen řetězec v určitém formátu, kde bude možné definovat validní hodnotu, jakou cílová aplikace očekává.

Nástroj bude možné konfigurovat modifikací souboru s nastavením v příslušném adresáři. V konfiguračním souboru bude ke každému parametru uveden detailní popis a možnosti nastavení. Bude možné nastavovat parametry ovlivňující průběh analýzy jako je například maximální počet zkoušených závorek pro nalezení správného formátu injekce.

O průběhu analýzy nástroj bude informovat na standardním výstupu, chyby zjištěné v průběhu analýzy budou vypisovány na standardní chybový výstup. Úroveň detailů výpisů bude možné nastavit příslušným parametrem nástroje. Bude možné volit mezi několika stupni, kdy při nejvyšším stupni budou vypisovány i informace pro účely ladění nástroje. Posílané požadavky a odpovědi testované webové aplikace budou zapisovány do logu ve formátu XML. Po dokončení analýzy bude vygenerovan přehledný report obsahující informace o každém testovaném vstupu a jeho zjištěné zranitelnosti na útoky typu SQL injection. Re-

port bude vygenerován jako webová stránka ve formátu HTML s využitím technologie CSS a JavaScript.

4.2 Analýza a návrh nástroje

Detekce SQL injection byla rozdělena do třech fází. Každá fáze využívá a pracuje s informacemi získanými ve fázi předchozí. Hlavním cílem první fáze je získání referenční odpovědi na nepoškozený dotaz a získání odpovědi na injekci, která naruší syntaxi cílového SQL dotazu. Rozdílnosti obdržených odpovědí budou vyhodnocovány a odpovědi vyhodnocené jako užitečné pro další analýzu budou uloženy. Ve druhé fázi je manipulováno s podmínkou cílového SQL dotazu a jsou zjišťovány odlišnosti a možné reakce na vložené injekce. Ve třetí fázi je hledáno správné ukončení podmínky cílového SQL dotazu a jsou zjišťovány reakce na injekce dalších SQL konstrukcí a příkazů.

Proces vývoje

Vzhledem ke zvolenému rozdělení detekce do fází byla vybrána metodika spirálového přístupu k procesu vývoje. Implementace každé další fáze bude znamenat vytvoření dalšího prototypu. Každý prototyp tak může být samostatně testován, zda implementace další fáze neovlivnila detekce fází předchozích. V následujícím přehledu je uveden souhrn vybraných technik detekce a jejich rozdělení do jednotlivých fází:

1. Fáze zjištění reakcí testované aplikace
   - Detekce reakce na chybu aplikace po chybě databázového systému.
   - Vyhledávání chybových výpisů nebo jejich částí v odpovědi.
2. Fáze modifikace vstupního dotazu
   - Zjištění reakce na modifikaci podmínky WHERE.
   - Detekce delší odezvy po úspěšném sestavení tautologie.
3. Fáze přidávání vlastních konstrukcí
   - Detekce delší doby odpovědi po vložení příkazu pro zpoždění do podmínky WHERE.
   - Detekce delší doby odpovědi po vložení samostatného příkazu pro zpoždění.
   - Detekce podvrženého řetězce s využitím konstrukce UNION.

Pro objektově orientovanou implementaci nástroje bylo zvoleno prostředí Eclipse a programovací jazyk Java. V rámci návrhu byl vytvořen diagram tříd zahrnující nejdůležitější třídy a vztahy. Diagram je k dispozici v příloze této práce.

Koncepce nástroje

Nástroj je navržen takovým způsobem, aby bylo možné jeho rozšiřování o další funkcionality, zejména o kvalitnější craweler. Detektor slabiny SQL injection je tedy zásuvným modulem implementovaného nástroje. Takové řešení zvýšilo náročnost návrhu implementace, přinese však možnost jeho budoucího rozšíření o další zásuvné moduly. Návrh tak umožní vytvoření nástroje pro komplexní analýzu bezpečnosti webové aplikace.

Jednotlivé funkcionality nástroje jsou navrženy jako samostatné třídy, aby byla zajištěna možnost jejich doplňování. Například pro odeslání a přijetí požadavku existuje objekt Communicator a po jeho rozšíření o komunikaci protokolem HTTPS bude možné analyzovat i aplikace dostupné pouze pod protokolem HTTPS.

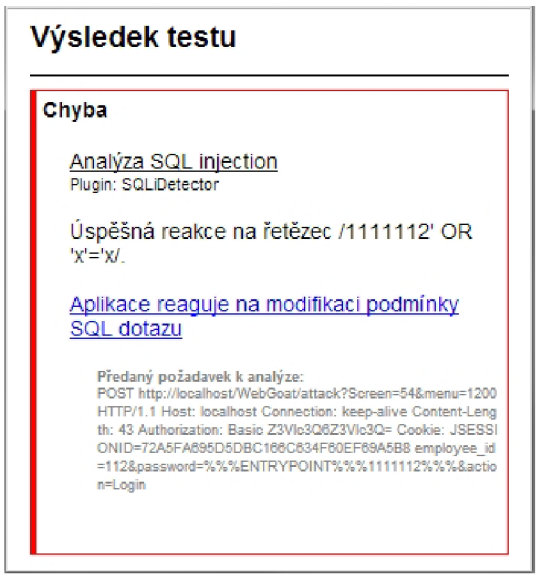
Detaily k návrhu implementace

Pro vyhodnocování úspěšnosti provedené injekce, respektive porovnávání a vyhledávání anomálií v odpovědích, je navržena samostatná třída, především kvůli zpřehlednění a oddělení těchto funkcionalit a jejich algoritmů od řízení analýzy zranitelnosti testované aplikace. Pro každý požadavek k otestování tedy bude vytvořen objekt třídy pro vyhodnocování odpovědí, kterému budou předávány získané odpovědi na různé typy injekcí. Předané odpovědi budou podle potřeby uchovávány a s postupem analýzy bude objekt poskytovat přesnější informace o úspěšnosti provedené injekce.

Definice injekcí a specifické informace k jednotlivým databázovým systémům poskytují moduly databázových systémů, které spadají pod zásuvný modul detektoru SQL injection. Detektor před analýzou zjistí všechny dostupné moduly a při analýze využije informace získané ze všech poskytnutých modulů. Každá technika proběhne pro každý modul zvlášť, a pokud je injekce získaná z daného modulu úspěšná, přidá se do reportu název databázového systému a provedená injekce.

Součástí návrhu je i volba formátů pro ukládání definicí SQL injekcí a pro uložení částí chybových výpisů jednotlivých modulů databázových systémů. Definice SQL injekcí jsou uloženy ve formátu XML a jsou rozděleny podle fází detekce slabiny SQL injection. Pro uložení částí chybových výpisů je navržen vlastní jednoduchý formát, kde obsah každého neprázdného řádku, který nezačíná sekvencí dvou pomlček, je brán jako chybové slovo.

V průběhu analýzy bude postupně vytvářen report složený z jednotlivých položek. Položky reportu jsou generovány především zásuvnými moduly a položce lze nastavit některé údaje, jako je například priorita indikující závažnost nálezu nebo jméno zásuvného modulu. Report je vygenerován po ukončení činnosti všech dostupných zásuvných modulů a obsahuje informace potřebné k replikování injekcí a manuální verifikaci nálezů.



Obrázek 4.1: Návrh výsledného reportu

4.3 Popis implementace

Program začíná a jeho řídící část je definována třídou *ScanController*. Zde jsou vytvářeny a spouštěny instance jednotlivých zásuvných modulů. Nejdříve je spuštěn zásuvný modul *Crawler* a po jeho ukončení následuje spuštění modulu *SQLiDetector*, který si přebírá požadavek k analýze z instance třídy *Crawler*. Třída *ScanController* také vytváří instanci třídy *Reporter*, která dále poskytuje funkcionality pro vygenerování reportu o provedené analýze.

Implementované zásuvné moduly *Crawler* a *SQLiDetector* jsou potomky třídy *AbstractPlugin*, definující základní metody zásuvného modulu, jako je například metoda *Start()* pro zahájení činnosti modulu nebo metodu *GetReportItems()*, ve které jsou uloženy výsledky činnosti daného modulu v podobě sekvence objektů třídy *ReportItem*. Metody pro vygenerování reportu z objektů třídy *ReportItem* poskytuje výše zmíněná třída *Reporter*.

Analýzu zranitelnosti aplikace vůči útoku SQL injection implementuje zásuvný modul *SQLiDetector*. Před spuštěním modulu je nutné předat uspořádanou kolekci požadavků k analýze metodou *SetRequestsToAnalyze()*. Jednotlivé požadavky jsou po spuštění modulu předávány metodě *AnalyzeRequest()*, kde je implementována analýza požadavku. Pro každý požadavek je vytvořena instance *Report* třídy *ReportItem* představující položku výsledného reportu. Tato instance je po provedené analýze požadavku přidána do uspořádané kolekce *LastRunReportItems*, kde jsou ukládány všechny položky reportu pro zásuvný modul *SQLiDetector*. V případě neúspěšné analýzy, například kvůli poškozenému požadavku, je vyvolána výjimka *CantContinueException*.

Práce s moduly databázových systémů

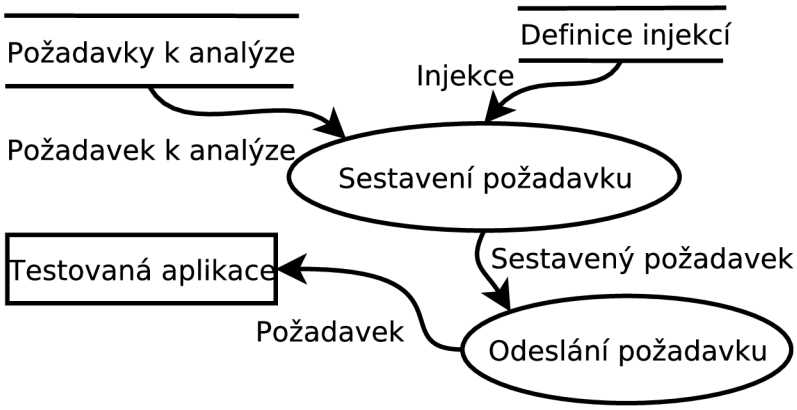
Během analýzy se pracuje s moduly databázových systémů. Inicializace modulů databázových systémů probíhá v konstruktoru zásuvného modulu *SQLiDetector*. Během vytváření zásuvného modulu je provedeno načtení parametrů pro provedení analýzy s využitím objektu třídy *WebScanSettings*. Po vytvoření instance je nutné zavolání metody *LoadDefFile()*, která obstará načtení řetězců z definičních souborů pro daný modul databázového systému. Po úspěšném načtení definicí je modul připravený poskytovat potřebné informace k analýze a sestavování SQL injekcí pro konkrétní databázový systém. Připravený modul databázového systému je dále přidán do uspořádané kolekce *DBModules*.

S moduly databázových systémů pracuje především metoda *AnalyzeRequest()*, která implementuje všechny fáze detekce slabiny SQL injection a vytváří instanci třídy *ReportItem*, do které jsou ukládány výsledky a nálezy zjištěné v průběhu analýzy. Parametrem metody *AnalyzeRequest()* je požadavek s označeným vstupem k analýze. Vstup musí být označen řetězcem ve formátu *%%%ENTRYPOINT%%%x%%%*, kde x je hodnota vstupu pro existující záznam. Hodnota x je dále využívána při analýze a pro tyto účely uložena do proměnné *ExistValue*. Pokud bude dosazena hodnota, pro kterou nevrátí cílový databázový systém žádný výsledek, bude analýza u některých technik méně efektivní, to se týká například techniky logického vylučování. V případě nedodržení formátu nebo nenalezení označení vstupu je vyvolána výjimka *CantContinueException*.

4.3.1 Přípravná fáze

Před první fází detekce slabiny SQL injection proběhnou přípravné operace a to zejména získání referenční odpovědi testované aplikace a ověření stability odpovědi. Referenční odpověď je získána na základě požadavku s dosazenou hodnotou pro existující záznam. Kromě ověření dostupnosti cílové aplikace je získaná odpověď důležitá pro další analýzu, protože se

jedná o odpověď na validní požadavek (validní z pohledu testované aplikace). Odpověď obsahuje mnoho informací, které jsou při analýze využívány, jedná se například o stavový kód nebo délku obsahu odpovědi. Principem analýzy je vyhodnocování úspěšnosti provedené injekce, probíhající na základě porovnávání informací získaných z odpovědi. Je proto nutné ověřit stabilitu porovnávaných informací a informace, které nejsou stabilní, dále při analýze nebrat v úvahu. Pro ověření stability je poslán druhý požadavek s dosazenou hodnotou pro existující záznam a získaná odpověď je porovnána s předchozí referenční odpovědí.



Obrázek 4.2: Diagram znázorňujcí zpracování požadavku k analýze

Porovnávání získaných odpovědí na modifikované požadavky a další potřebné funkcionality pro analýzu jsou implementovány ve třídě *SQLiResponseAnalyzer*. Po ověření stability odpovědí testované aplikace je vytvořena instance *RespAnalyzer* třídy *SQLiResponseAnalyzer* a referenční odpověď je jedním z parametrů konstruktoru. Dalšími parametry jsou například informace o proběhnuté kontrole stability odpovědí. Během vytváření objektu *RespAnalyzer* je také proveden výpočet běžných časů a tolerancí zpoždění odpovědi, které jsou dále využívány při technikách detekujících reakce cílové aplikace na základě času odpovědí. Výpočet tolerancí je implementován v metodě *CalcResponseTimeTolerance()*.

První využití objektu *RespAnalyzer* spočívá v ohodnocení chybových slov jednotlivých modulů databázových systémů. Pro každý modul databázového systému je možné definovat seznam chybových slov, která se typicky vyskytují v chybovém výpise daného databázového systému. Je však možné, že se v referenční odpovědi běžně vyskytují slova, která jsou definována jako chybová pro některý modul databázového systému. Metoda *EvaluateKeywordList()* objektu *RespAnalyzer* umožní upravit seznamy modulů databázových systémů na základě referenční odpovědi takovým způsobem, aby u techniky detekce typu **error-based** nevznikaly falešné nálezy slabin SQL injection.

4.3.2 Fáze zjištění reakcí testované aplikace

V první fázi detekce SQL injection jsou do vstupu testované aplikace vkládány řetězce, které by měly způsobit chybu v testované aplikaci při provádění dynamicky sestaveného SQL dotazu. Je volána metoda *GetNextDiscoveringSQLiString()* pro všechny moduly v uspořádané kolekci *DBModules*. Výsledkem volání je řetězec, který bude použit pro sestavení injekce. V případě, že modul nemá k dispozici další injekce daného typu, vyvolá příslušná metoda výjimku *NothingNextException*. Pokud modul databázového systému vrátí řetězec, je vytvořena nová instance požadavku newReq, ve kterém je na místo označeného vstupu vložen

vrácený řetězec. Po přijetí odpovědi na sestavený požadavek následuje analýza, zda byla injekce úspěšná. Pro vyhodnocení úspěchu injekce jsou volány metody objektu *RespAnalyzer*. Jedná se například o metodu *AreThereAnyKeywords()*, která vrátí součet délek chybových slov nalezených v předané odpovědi. Slova jsou vyhledávány na základě předaného seznamu, který poskytne databázový modul. Vrácené číslo je dále nazýváno jako skóre chybových slov, a čím je skóre větší, tím je jistější, že odpověď obsahuje chybový výpis popisující vyvolanou chybu cílového databázového systému, tedy že byla injekce úspěšná.

Pokud některá z kontrol úspěšnosti zjistí reakci testované aplikace na poslanou injekci, jsou do položky reportu s využitím metody *addDetail()* přidány informace o nálezu a případně navýšena její priorita. Do položky reportu jsou přidávány i detaily vyhodnocení, například při zjištění, jaký datový typ cílový databázový systém očekává. Po záměrném vyvolání chyby v testované aplikaci je příslušná odpověď předána metodou *setErrorBehalf()* objektu *RespAnalyzer*, kde dále slouží jako vzor pro další vyhodnocování.

4.3.3 Fáze modifikace vstupního dotazu

V rámci druhé fáze jsou posílány injekce s cílem ovlivnit výslednou podmínku WHERE dynamicky sestavovaného SQL dotazu. Jsou stanoveny tři druhy injekcí a každá by měla specifickým způsobem ovlivnit výsledek vrácený cílovým databázovým systémem. Nejdříve je hledána úspěšná reakce na injekci typu FALSE, která by měla ovlivnit cílový SQL dotaz takovým způsobem, aby nebyl vrácen cílovým databázovým systémem žádný výsledek. Poté následuje injekce typu ONE, která by měla zajistit vrácení právě jednoho výsledku. Nakonec jsou použity injekce typu TRUE pro vrácení všech možných výsledků. Pro každou obdrženou odpověď je volána metoda *IsSuccessfulWhereInjection()* objektu *RespAnalyzer*, která vyhodnotí úspěšnost injekce s manipulací klauzule cílového SQL dotazu. Metoda je užitečná zejména pro varianty SQL slabin, kde je poskytován výpis z cílové databáze na základě nějakého identifikátoru nebo jiných hodnot, například se může jednat o filtrování.

Před vyhodnocením je každá odpověď analyzována, zda se nejedná o odpověď indikující reakci cílové aplikace na chybu. Pokud není detekována chyba, přistoupí se k vyhodnocení úspěšnosti provedené injekce. Úspěšnost je vyhodnocována především podle délek obsahů odpovědí a to podle stanoveného pravidla, že délka odpovědi na injekci typu TRUE bude delší nebo rovna odpovědi na injekci typu ONE, která bude zároveň delší než odpověď na injekci typu FALSE. Z uvedeného pravidla vyplývá, že pokud nebude v cílové databázi ani jeden záznam, nebude tato technika detekce úspěšná. Možná rovnost délek mezi odpověďmi na injekci typu TRUE a ONE je pro případy, kdy je v cílové databázi právě jeden záznam. Pokud odpověď vyhovuje uvedené podmínce, vrátí metoda informaci o úspěšnosti a zároveň si danou odpověď uloží pro přesnější vyhodnocení dalších odpovědí. Jsou také zvlášť ukládány i odpovědi, které neindikují chybu cílové aplikace a nesplňují výše uvedenou podmínku. Například odpovědi na injekci typu TRUE jsou uloženy do uspořádané kolekce *TrueResponses* objektu *RespAnalyzer*.

Vyhodnocení odpovědí

Po odeslání a vyhodnocení všech druhů injekcí pro druhou fázi je zavolána metoda *IsWhereReaction()* objektu *RespAnalyzer*, která na základě uložených odpovědí na úspěšné injekce a kontrolních podmínek eliminujících falešné nálezy vyhodnotí možnou reakci cílové aplikace na modifikaci klauzule cílového SQL dotazu. Pokud není zjištěna reakce, následuje volání metody *CouldBeBoolBlind()* objektu *RespAnalyzer*. Metoda *CouldBeBoolBlind()* vyhodnotí možnou reakci cílové aplikace na injekce typu TRUE na základě podmínky, že některá

z uložených odpovědí na injekci typu TRUE se liší od uložených odpovědí na injekce typu FALSE. Při takovém vyhodnocení je větší pravděpodobnost výskytu falešných nálezů, proto je zjištěná reakce reportována s nižší prioritou a doporučením manuálního ověření.

Další dodatečné ověření reakce je provedeno metodou *CouldBeTRUEresponseReaction()* objektu *RespAnalyzer*. Tato metoda pracuje s předpokladem, že některá odpověď na injekci TRUE může být zpožděná oproti běžné odpovědi. Jedná se o varianty, kde injekce TRUE vytvoří dotaz na mnoho záznamů v cílové databázi, což se může projevit v delší době zpracování takového dotazu cílovým databázovým systémem. Zjištěné reakce metodou *CouldBeTRUEresponseReaction()* jsou reportovány s nižší prioritou, protože můžou vznikat falešné nálezy především kvůli možnému zpoždění odpovědi z jiných důvodů, například kvůli zpoždění na síti vymykající se vypočítané toleranci.

4.3.4 Fáze přidávání vlastních konstrukcí

Ve třetí fázi probíhá hledání možného ukončení klauzule WHERE cílového SQL dotazu. Pokud je nalezeno správné ukončení, můžou být implementované techniky detekce SQL injection slabiny v rámci třetí fáze úspěšné. Hledání správného ukončení je ovlivněno hodnotou v proměnné *PrhpsCount* určující, kolik závorek pro ukončení klauzule WHERE má být zkoušeno. Celé ukončení je výsledkem konkatenace hodnoty proměnné *ExistValue*, možného ukončení z uspořádané kolekce vrácené aktuálním databázovým modulem s využitím metody *getEndOfWHEREsqli()* a určitého počtu závorek.

Zpožďování databázového systému

První implementovanou technikou ve třetí fázi je vkládání samostatného příkazu pro zpoždění odpovědi cílového databázového systému. Technika je prováděna pro každý databázový modul zvlášť a příkazy jsou z databázového modulu načítány pomocí metody *GetNextSleepConfirmingSQLi()*. K příkazům je také definována i odpovídající doba zpoždění cílového databázového systému. Doba zpoždění je načtena s využitím metody *GetNextSleepTimeConfirmingSQLi()*.

Úspěšnost injekce se dále vyhodnocuje na základě času odpovědi, a proto pokud je definovaná doba zpoždění daného příkazu menší než součet tolerance a běžné doby odpovědi, je daný příkaz přeskočen. Potřebné údaje, jako jsou tolerance a běžná doba odpovědi, poskytne objekt *RespAnalyzer*, konkrétně jeho metody *getOrdinaryResponseTime()* a *getResponseTimeTolerance()*. Výsledná injekce je sestavena z možného ukončení, oddělovačem příkazů získaného z databázového modulu metodou *getStatementSeparator()*, příkazu pro zpoždění odpovědi cílového databázového systému a literálu komentáře získaného z databázového modulu metodou *getLineCommentLiteral()*.

Pro každý příkaz jsou sestaveny a poslány všechny varianty injekcí s využitím všech možných ukončení a každého možného počtu pravých závorek. Obdržené odpovědi jsou spolu s očekávanou dobou zpoždění předány k analýze objektu *RespAnalyzer* jako parametry metody *IsSuccessfulSleepInjection()*, která vyhodnotí, zda cílová aplikace na injekci zareagovala. Při zjištěné reakci je nález přidán do reportu s nejvyšší prioritou.

U některých databázových systémů, jako je například MySQL, je možné vložit příkaz pro zpoždění odpovědi přímo do klauzule WHERE. Pokud nebyla předchozí technika úspěšná, jsou dále sestaveny a odeslány injekce s příkazy zpoždění v klauzuli WHERE. Tyto příkazy jsou získávány z databázového modulu metodou *GetNextInSleepConfirmingSQLi()* a příslušné hodnoty zpoždění metodou *GetNextInSleepTimeConfirmingSQLi()*.

Odpovědi jsou vyhodnocovány stejně jako u předchozí techniky metodou *IsSuccessfulSleepInjection()* objektu *RespAnalyzer*.

Využití konstrukce UNION

Další technikou je detekce slabiny SQL injection s využitím konstrukce UNION. Pro techniku UNION je nutné nalézt odpovídající počet sloupců. Hledání správného počtu pak spočívá v postupném přidávání sloupců. Maximální počet sloupců, pro který má detekce probíhat, je dán hodnotou proměnné *ColumnCount*. Princip detekce spočívá v podvržení vlastního záznamu do výsledku vráceného databázovým systémem testované aplikaci. V případě úspěšného provedení injekce je možné, že bude podvržený záznam v odpovědi testované aplikace. V některých variantách může aplikace vracet pouze první řádek výsledku, proto je vhodné definovat injekci, která způsobí omezení výsledku na podvržený záznam.

Podoba záznamu, který se má v obdržené odpovědi testované aplikace objevit, je načtena z databázového modulu metodou *getUnionOut()*. Samotné injekce pro techniku UNION jsou načítány z databázového modulu s využitím metody *getUnionSQLi()*. Načtená injekce musí obsahovat značku %UN% určující místo pro vkládání sloupců. Řetězec odpovídající sloupci je získán z databázového modulu metodou *getUnionNextNull()*.

S každou injekcí jsou vytvořeny kombinace s každým možným ukončením klauzule WHERE, dále kombinace pro všechny počty závorek a pro všechny počty sloupců. Po odeslání sestavené injekce je odpověď předána jako jeden z parametrů metody *IsSuccessfulUnionInjection()* objektu *RespAnalyzer*, kde je implementováno vyhledání podvrženého záznamu. V případě nalezení podvrženého záznamu je do reportu přidán nález s nejvyšší prioritou. Implementovaná technika detekce s využitím konstrukce UNION je jedna z nejpřesnějších technik pro automatizovanou detekci slabiny SQL injection.

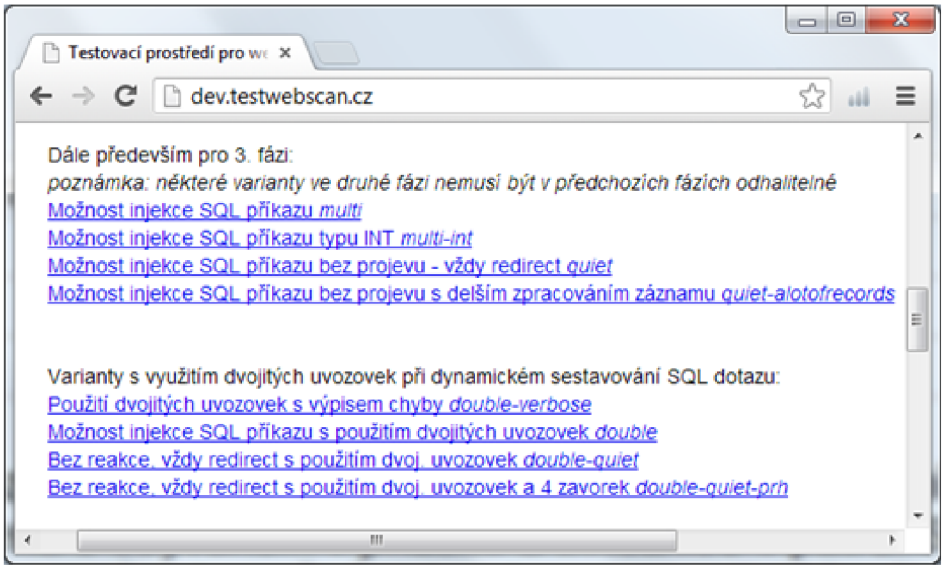
4.4 Aplikace s demonstrací variant SQL injection

V rámci praktické části byla vytvořena také aplikace testwebscan demonstrující různé varianty SQL injection slabin. Jedná se o webovou aplikaci, která posloužila také jako nástroj pro ladění a regresní testování nástroje pro detekci SQL injection slabin. V aplikaci jsou k dispozici nejrůznější varianty slabin SQL injection z pohledu jejich detekce, od nejjednodušších až po varianty obtížně zjistitelné. Jsou také k dispozici funkcionality pro ztížení automatizované detekce, například generování náhodného obsahu nebo náhodné zpoždění odpovědi.

4.4.1 Popis aplikace

V aplikaci je implementováno několik variant zpracování a následného použití vstupů aplikace pro sestavení SQL dotazu. Pro účely demonstrace jsou také implementovány varianty možných reakcí aplikace na výskyt chyby databázového systému. K dispozici jsou i varianty s využitím parametrizovaných SQL dotazů pro ukázku správného zabezpečení vstupů aplikace. Konkrétní variantu zpracování a použití vstupu lze zvolit pomocí parametru page v URL.

Jednotlivé varianty jsou orientačně rozděleny do třech fází, které odpovídají implementaci nástroje pro detekci SQL injection slabin. Některé varianty tak nemusí být odhaleny v předchozí fázi detekce, protože k jejich odhalení je nutné použít techniku implementovanou až ve fázi následující.



Obrázek 4.3: Ukázka seznamu některých variant

V první fázi jsou především varianty, u kterých je možné rozpoznat reakci webové aplikace na výskyt chyby databázového systému. Jedná se například o chybový výpis databázového systému, přesměrování nebo vrácení obecné chyby aplikace.

Do druhé fáze spadají varianty s ošetřením chyby databázového systému. Není tak možné detekovat slabinu SQL injection podle reakce webové aplikace na poškození syntaxe cílového SQL dotazu. Jedná se o varianty se specifickým chováním pro existující nebo neexistující záznam. Detekce takových případů pak vede na modifikaci klauzule WHERE cílového SQL dotazu.

Ve třetí fázi jsou zařazeny varianty bez zjistitelné reakce na chybu a také bez specifického chování pro existující nebo neexistující záznam. V praxi se může jednat o případy, kdy aplikace provádí pouze vytvoření záznamu a bez ohledu na výskyt chyby databázového systému provádí přesměrování. Takovou slabinu lze odhalit například injekcí pro zpoždění odpovědi cílového databázového systému. Dále jsou do třetí fáze zařazeny varianty určené pro testování techniky detekce typu UNION nebo varianty se složitějším ukončením klauzule WHERE, jako je například vložení vstupu na místo parametru funkce v sestavovaném SQL dotazu.

Aplikace nabízí rozšiřující možnosti zpoždění odpovědi a generování náhodně dlouhého obsahu. Tyto rozšíření je možné vyvolat přidáním příslušného parametru do URL. Pro náhodné generování obsahu lze přidat parametr *random*, pro konstantní zpoždění odpovědi parametr *sleep* a pro náhodné zpoždění odpovědi parametr *rsleep*. V možnostech aplikace lze také spouštět akce pro obnovení a naplnění některých tabulek v databázi testovacími daty.

Aplikační vrstva pracuje s malou databází MySQL a pokyny pro její konfiguraci jsou uvedeny v přiloženém souboru README. Přiložen je také dump použitelné databáze. Obtížnost detekce slabiny SQL injection může být u některých variant závislá i na počtu řádků v cílové tabulce, je tak vhodné experimentovat i s různým obsahem databáze.

4.4.2 Popis implementace

Webová aplikace je implementována v jazyce PHP a připojuje se k databázovému systému MySQL. Ve výchozí konfiguraci se aplikace připojuje k databázovému systému pod uživatelem *user* k databázi *user* pod heslem *user*. Konfiguraci je možné změnit v souboru *index.php*.

Vytvoření potřebných objektů a proměnných pro běh aplikace je implementováno v souboru *index.php*, kde je také provedeno zahrnutí dalších potřebných souborů. Vždy je zahrnut soubor *hrefs.php* se seznamem odkazů na jednotlivé varianty a soubor *view.php* zajišťující generování výsledného HTML dokumentu. Veškerý obsah generovaný za běhu aplikace je ukládán do proměnné *$contentBuffer*, která je vypsána na výstup až skriptem v souboru *view.php*.

Každá varianta je implementována v samostatném souboru, který je zahrnut při zadání příslušného názvu varianty do parametru page v URL. Pokud parametr page obsahuje neznámý název varianty, je vrácena informace o špatně zadané akci. Pro přehlednost byla oddělena implementace jednotlivých variant do samostatného adresáře *examples*. Akce obnovující některé databázové objekty jsou také implementovány jako samostatné soubory a jsou umístěny v adresáři *actions*.

4.5 Experimentální výsledky

Funkčnost nástroje a jeho schopnost detekovat zranitelnosti typu SQL injection ve webové aplikaci byla experimentálně ověřena na několika aplikacích. Závěrečný test probíhal na výukové aplikaci WebGoat. Pro účely ověření detekce nejrůznějších variant slabin SQL injection byla také implementována samostatná webová aplikace testwebscan, která byla více popsána výše, v kapitole 4.4. Výsledky experimentování s implementovaným nástrojem byly porovnány s výsledky pokročilého nástroje sqlmap, který je uveden v přehledu stávajících automatizovaných nástrojů v kapitole 3. Implementovaný nástroj i nástroj sqlmap byly spouštěny s výchozím nastavením bez přikládání dodatečných informací o správném formátu injekce.

Experimentování s aplikací testwebscan

Rozsáhlý test funkčnosti implementovaného nástroje bylo možné provést na webové aplikaci testwebscan. Nástroj byl na této aplikaci odladěn a při závěrečném testu nástroj detekoval i obtížně zjistitelné SQL injection slabiny a to i se zapnutím dynamické délky obsahu odpovědi.

Z provedených testů na aplikaci testwebscan vyplynulo, že nejúspěšnější technikou je technika vkládání příkazu zpoždění pro cílový databázový systém. Nástroj upřednostňuje techniku vkládání zpoždění jako samostatný příkaz a pouze v případě neúspěchu je testováno vložení zpoždění do podmínky WHERE. Toto upřednostnění vyplývá z větší závažnosti bezpečnostní chyby vyplývající z možnosti injekce vlastního příkazu. Techniky zpožďování odpovědi databázového systému bylo možné vyřadit pouze simulací náhodného zpožďování odpovědí aplikace. Implementovaný nástroj však počítá i s možnou nestabilitou časů odpovědí, a tak bylo možné techniku zpoždění zmást až při náhodném zpožďování jednotlivých odpovědí v rozmezí 0 až 3 sekundy.

Na některých variantách vyžadujících pokročilou detekci SQL injection byl vyzkoušen pro porovnání i nástroj sqlmap. Nástroj sqlmap nedetekoval slabinu ve variantě func, která

má zranitelný vstup v parametru URL. Jedná se o variantu, kde nejsou poskytovány informace o výsledku provedeného dotazu, vždy je vrácena stejná odpověď. Vstup je vložen do sestavovaného SQL dotazu jako parametr funkce. Následuje ukázka úspěšné injekce pro variantu func:

```
zakaznik1',%201);SELECT%20SLEEP(5);--
```

Aby bylo možné sestavit takovou injekci a následně detekovat slabinu, je nutné správně ukončit podmínku WHERE. V předchozí ukázce je jako součást ukončení kromě apostrofu přidán také další parametr. Implementovaný nástroj obsahuje definici ukončení i pro tyto varianty slabin SQL injection a na místo parametru funkce vkládá hodnotu NULL.

Časová náročnost analýzy

Implementovaný nástroj vynechává některé techniky, pokud není možné vyhledávat a rozeznávat specifické reakce na základě příslušných parametrů odpovědi testované aplikace, což se stává v případech, kdy je daný parametr nestabilní neboli různý pro dva stejné požadavky. Ve variantách, které neobsahují SQL injection slabinu a zároveň vrací stabilní odpovědi pro všechny implementované techniky detekce, nástroj s výchozím nastavením posílá celkem 710 požadavků. Takový počet je sice přijatelný při manuálním spouštění nástroje pro otestování konkrétního vstupu lokální aplikace, může však být pro některé využití nástroje nepřípustný, protože při průměrném času odpovědi 400ms bude analýza jednoho vstupu trvat až 5 minut.

Převážnou část z posílaných požadavků zabírá technika UNION, protože je nutné otestovat všechny definované formáty injekcí pro tuto techniku se všemi definovanými způsoby ukončení podmínky WHERE a pro všechny možné počty sloupců a závorek. I mnoho dalších technik pracuje s nastaveným počtem závorek a s definovanými způsoby ukončení podmínky WHERE. Je vhodné přizpůsobit nastavení nástroje a definice injekcí takovým způsobem, aby jeho analýza zbytečně neprodlužovala čas běhu celkového testování při jeho zapojení do komplexnějšího testu webové aplikace.

Experimentování s aplikací WebGoat

Funkčnost nástroje byla ověřena i na aplikaci WebGoat. Nebylo možné odhalit varianty typu **blind SQL injection**, protože tato technika vyžaduje specifické definice pro daný cílový databázový systém. I přes to, že je v aplikaci využit jiný databázový systém než je MySQL, byly některé varianty odhalitelné. Jedná se například o následující:

- **Numeric injection** - slabina odhalena technikami detekce chybových slov a modifikace podmínky WHERE
- **String injection** - slabina odhalena technikou modifikace podmínky WHERE

Testování implementovaného nástroje na aplikaci WebGoat ukázalo, že tuto aplikaci není vhodné používat k ověření funkčnosti automatizovaných nástrojů, protože se jedná o výukovou aplikaci určenou zejména pro názornou výuku s využitím manuálních postupů. Tato skutečnost mate v mnoha případech automatizované nástroje, a to například ve cvičeních, kde aplikace WebGoat detekuje úspěšnou injekci a změní své odpovědi na analyzovaný požadavek. Pro obnovu původního chování je nutné v aplikaci restartovat cvičení, na což nejsou standardní automatizované nástroje konstruovány.

Kromě uvedeného byla funkčnost implementovaného nástroje ověřena na variantě slabiny SQL injection nalezené v praxi. Jedná se o variantu, kde není použit databázový systém MySQL a navíc je vstup uživatele vkládán jako jeden z parametrů funkce cílového databázového systému. Z tohoto důvodu není jednoduché odvodit správné ukončení podmínky WHERE výsledného cílového SQL dotazu. Implementovaný nástroj odhalil slabinu na základě výskytu chybových slov, protože chybové výpisy testované aplikace obsahovaly mnohé fráze definované ve slovníku chybových slov pro modul MySQL. Tato varianta byla prověřena i nástrojem sqlmap, který slabinu SQL injection ve výchozím nastavení neodhalil.

Závěry experimentů

Funkčnost implementovaného nástroje byla ověřena na nemalém množství variant SQL injection slabin a ukázalo se, že může být plnohodnotným nástrojem penetračního testera. V porovnání s nástrojem sqlmap byl implementovaný nástroj při testech úspěšnější. I přes velké množství detekovaných variant však není možné tvrdit, že neexistuje varianta, kterou by nástroj neodhalil. Vždy se může objevit varianta vyžadující velmi specifický formát úspěšné injekce. Při podezření na nedokonalou validaci vstupu testované aplikace je vhodné daný vstup otestovat za pomoci více nástrojů a nespoléhat se na algoritmy a definice pouze jednoho nástroje.

Webová aplikace i nástroj pro detekci slabin typu SQL injcetion byly navrženy a implementovány s ohledem na jejich budoucí využití a možné rozšiřování. Během experimentů s implementovaným nástrojem a webovou aplikací se ukázalo, že kromě testování funkčnosti nástrojů pro detekci slabin lze používat webovou aplikaci také jako výbornou učební pomůcku. Na provedených experimentech bylo zřejmé, jak selhávají některé techniky detekce při zapnutí simulace nestability různých parametrů odpovědí. Implementovaný nástroj byl při experimentování porovnáván s jinými běžně dostupnými nástroji a byl schopen detekovat také varianty slabin SQL injection, které nebyly jinými nástroji odhaleny.

Kapitola 5

Závěr

V předchozích kapitolách byla představena a názorně vysvětlena problematika útoků typu SQL injection. Byly popsány nejznámější varianty útoků a v praxi používané techniky detekce slabin, které tyto útoky umožnují. Byly rozebrány výhody a nevýhody automatizované detekce slabin včetně uvedení některých v současné době používaných nástrojů. Kromě samotné detekce slabiny byly popsány základní postupy zneužití slabiny s uvedením ukázek injekcí. V práci byly také popsány možnosti obrany před útoky typu SQL injection.

V rámci práce byl navržen a implementován nástroj pro detekci slabin typu SQL injection ve webové aplikaci. Během experimentů bylo zjištěno, že implementovaný nástroj detekuje také slabiny, které jiné v praxi používané nástroje neodhalily. Vznikl tak velmi užitečný nástroj použitelný pro ověření zabezpečení vstupů webové aplikace. Pro účely ověření funkčnosti nástroje pro různé obtížně detekovatelné slabiny byla implementována také webová aplikace s malou databází demonstrující několik desítek případů nedostatečného ošetření vstupů. Webová aplikace umožňuje komplexně otestovat jakýkoliv nástroj zaměřený na hledání slabin SQL injection.

Přínosem této práce bylo také nalezení několika závažných bezpečnostních chyb ve veřejně dostupných aplikacích provozovaných pod doménou vutbr.cz. O všech nálezech byli neprodleně informováni administrátoři příslušných aplikací. Nalezené chyby spadaly do kategorií A1 a A3 metriky OWASP TOP 10 - 2013 a týkaly se například aplikace IS VUT, což poukazuje na aktuálnost řešeného problému. Některé z nalezených chyb nebyly ani po několika měsících od nahlášení opraveny, a proto nebylo možné v této práci uvést jejich detaily.

Práce poskytuje ucelený přehled jak základních, tak i některých pokročilých technik využívaných při útocích typu SQL injection. Podle popsaných postupů je možné prověřit kvalitu zabezpečení vstupů aplikace a ucelený přehled může výborně posloužit jako odborná literatura při studiu penetračního testování.

Implementovaný nástroj byl navržen s ohledem na jeho možné budoucí rozšiřování. Byl kladen důraz na snadné přidání podpory pro další databázové systémy a na srozumitelný formát slovníku definicí. Detekce slabin je implementována jako samostatný zásuvný modul, což umožňuje rozšíření nástroje o analýzy slabin jiných typů nebo také o kvalitní craweler.

Literatura

[1] Andreu, A.: *Professional Pen testing for web applications*. John Wiley & Sons, 2006, iSBN 978-0-471-78966-6.

[2] Bernardo Damele A. G. a Miroslav Stampar: Automatic SQL injection and database takeover tool [online]. `http://sqlmap.org/`.

[3] Boyd, Stephen W a Keromytis, Angelos D: SQLrand: Preventing SQL Injection Attacks [online]. `http://academiccommons.columbia.edu/catalog/ac:149138`.

[4] Clarke, J.: *SQL injection attacks and defense*. Syngress Media, 2012, iSBN 978-1-59749-424-3.

[5] David Litchfield: Data-mining with SQL Injection and Inference [online]. `http://www.databasesecurity.com/webapps/sqlinference.pdf`, 2005-09-30 [cit. 2013-05-05].

[6] Ericka Chickowski: 10 SQL Injection Tools For Database Pwnage [online]. `http://http://www.darkreading.com/database/slide-show-10-sql-injection-tools-for- da/232900180`, 2012-04-11 [cit. 2013-05-05].

[7] Fonseca, J.; Vieira, M.; Madeira, H.: Testing and comparing web vulnerability scanning tools for sql injection and xss attacks. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4459684&tag=1`, 2007 [cit. 2013-05-05].

[8] Free Software Foundation: GNU GENERAL PUBLIC LICENSE [online]. `http://www.gnu.org/licenses/gpl.txt`, 2007-01-29 [cit. 2013-05-05].

[9] Halfond, WG a Viegas, Jeremy a Orso, Alessandro: A classification of SQL-injection attacks and countermeasures [online]. `http://www.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf`, 2006 [cit. 2013-05-05].

[10] icesurfer: SQL Server injection & takeover tool [online]. `http://sqlninja.sourceforge.net`.

[11] Mati Aharoni: BackTrack Linux - Penetration Testing Distribution [online]. `http://www.backtrack-linux.org`.

[12] Meucci, M.: The OWASP testing guide v3 [online]. http://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf, 2007 [cit. 2013-05-05].

[13] Mick, J.: Nokia is the Victim of SQL Injection, Loses Developer Records [online].
http://www.dailytech.com/
Nokia+is+the+Victim+of+SQL+Injection+Loses+Developer +Records/
article22563.htm, 2011-08-29 [cit. 2013-05-05].

[14] Security Compass: SQL Inject-Me [online].
http://labs.securitycompass.com/exploit-me/sql-inject-me/.

[15] Serkan Ozkan: CVE security vulnerability database [online].
http://www.cvedetails.com, [cit. 2013-05-05].

[16] Smithline, N.: Top 10 2013-Top 10 [online].
https://www.owasp.org/index.php/Top_10_2013-T10, 2013-02-18 [cit. 2013-05-05].

[17] Splaine, S.: *Testing web security: assessing the security of web sites and applications.* Wiley, 2002, iSBN 0-471-23281-5.

[18] The PHP Group: PHP: Overview - Manual [online].
http://www.php.net/manual/en/mysqli.overview.php.

Příloha A

Obsah CD

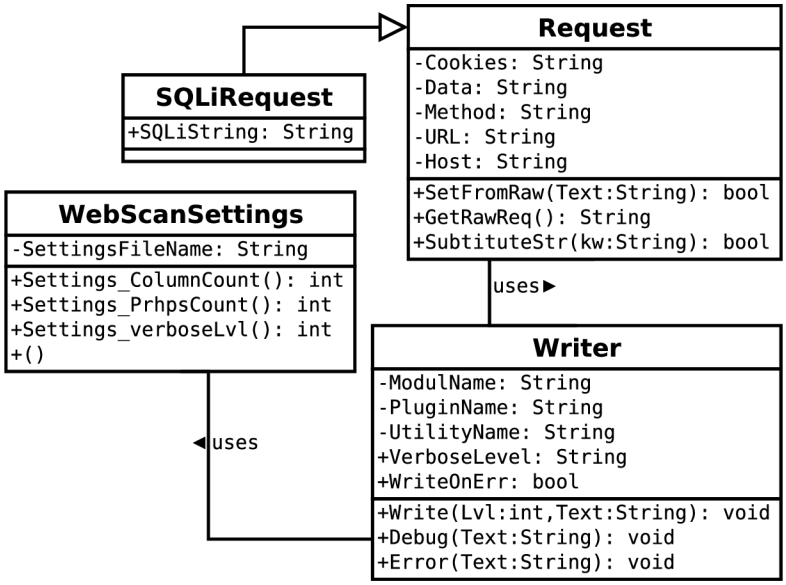
Přiložené CD obsahuje následující soubory a adresáře:

- **webscan** - adresář obsahující zdrojové kódy a zkompilované soubory nástroje pro detekci SQL injection zranitelností
- **testwebscan** - adresář obsahující zdrojové kódy a dump databáze aplikace pro demonstraci SQL injection zranitelností
- **tex** - adresář obsahující zdrojový text technické zprávy
- **technickazprava.pdf** - technická zpráva ve formátu PDF

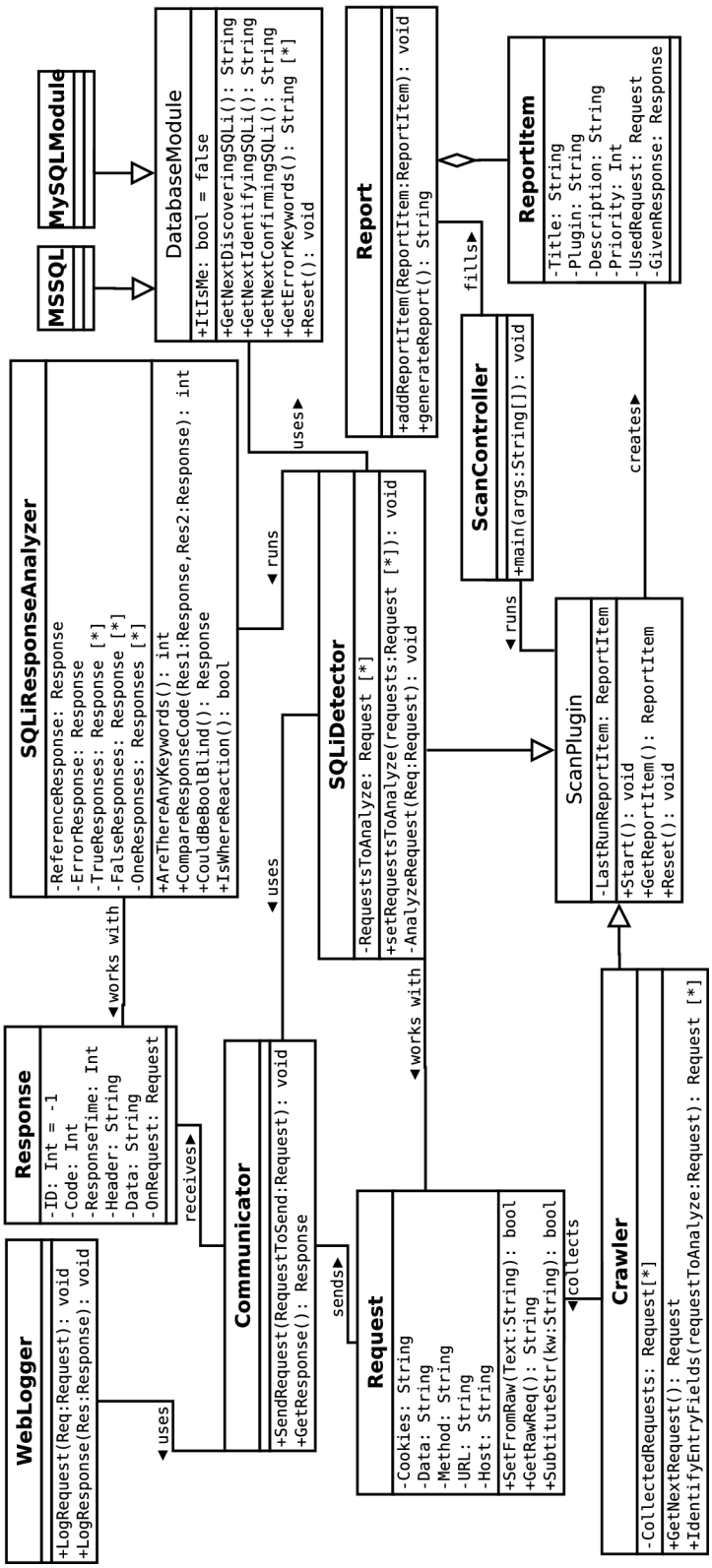
Příloha B

Diagram tříd

Níže jsou přiloženy diagramy tříd k implementovanému nástroji pro detekci SQL injection zranitelností, které znázorňují důležité třídy používané při analýze.



Obrázek B.1: Znázornění tříd spolupracujících s třídou *Request*



Obrázek B.2: Znázornění hlavních tříd spolupracujících při analýze

Příloha C

Manuál k implementovaným aplikacím

Nástroj pro detekci SQL injection zranitelností

V současné verzi je implementována analýza výskytu slabiny typu SQL injection. Na základě předaného požadavku je provedena analýza označeného vstupu webové aplikace. Aplikace byla testována v prostředí Java(TM) SE Runtime Environment (build 1.7.0_17-b02).

Přeložení a spuštění aplikace

Aplikace byla vyvíjena v prostředí Eclipse a je možné aplikaci překládat a spouštět v tomto prostředí.

Aplikaci je možné také přeložit s využitím nástroje ant (soubor build.xml je přiložen). Přeložení aplikace s přiloženým souborem *build.xml* bylo testováno s Apache Ant(TM) verze 1.9.0 kompilovanou 5.3.2013.

Následují příklady práce s nástrojem ant:

- přeložení: *ant -buildfile D:\bbb\eclipseWS\webscan\build.xml*
- spuštění: *ant ScanController -buildfile D:\bbb\eclipseWS\webscan\build.xml*

Před spuštěním analýzy je nutné do souboru *requestToAnalyze.txt* (v adresáři *config*) vložit požadavek k analýze. V požadavku musí být označen vstup řetězcem *%%%ENTRY-POINT%%%x%%%*, kde *x* je hodnota, pro kterou existuje v cílové databázi záznam. Pokud se bude jednat o hodnotu, pro kterou neexistuje v cílové databázi záznam, může být analýza v některých případech méně přesná. Uvedený soubor obsahuje vzorový POST požadavek.

V adresáři *config* se také nachází soubor *settings.txt*, kde je možné nastavit parametry analýzy. Jednotlivé možnosti nastavení jsou popsány v uvedeném souboru.

Informace o průběhu analýzy jsou vypisovány na standardní výstup a chyby na standardní chybový výstup. Po skončení analýzy je vygenerována závěrečná zpráva do souboru *report.html*. Všechny nálezy a podrobnosti potřebné k jejich ověření jsou obsaženy ve vygenerované závěrečné zprávě. Posílané požadavky a odpovědi jsou logovány do souboru *log.xml*.

UPOZORNĚNÍ

Nástroj je určen pouze pro prověření vlastní vyvíjené webové aplikace. Bez příslušného

povolení není legální na webové aplikace útočit nebo provádět penetrační testy. Za zneužití nebo způsobené škody tímto nástrojem nenese autor nástroje žádnou odpovědnost.

Webová aplikace pro demonstraci SQL injection zranitelností

Pro fungování aplikace je nutné mít instalovaný databázový systém MySQL s databází *user*, uživatelem *user* s heslem *user*. Tyto údaje je možné měnit v souboru *index.php*. Do databáze je nutné importovat testovací data - pro tyto účely je přiložen dump, který lze nalézt v adresáři *setup*. Chování aplikace je možné ovlivňovat parametry v URL, jejich popis je uveden na úvodní stránce aplikace.

UPOZORNĚNÍ

Jedná se o aplikaci se slabinami SQL injection, při nesprávné konfiguraci může útočník získat kontrolu nad serverem. Neprovozujte aplikaci veřejně, vždy pouze lokálně po nezbytně nutnou dobu a s minimálním oprávněním!