Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra systémové analýzy

Studijní program: Aplikovaná informatika

Obor: Informatika

Testování zabezpečení webové aplikace

BAKALÁŘSKÁ PRÁCE

Student: Martin Kapal

Vedoucí: Ing. Jaromír Veber, Ph.D.

Oponent: Ing. Ladislav Luc

2017

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Veškeré použité podklady, ze kterých jsem čerpal informace, jsou uvedeny v seznamu použité literatury a citovány v textu podle normy ČSN ISO 690.

V Praze dne 25. dubna 2017 Podpis: .....

Poděkování

Především bych chtěl poděkovat panu Ing. Jaromíru Veberovi, Ph.D. za užitečné rady a připomínky během psaní této práce. Také bych chtěl poděkovat své rodině a všem, kteří mě při psaní bakalářské práce podporovali.

Abstrakt

Tato práce se zabývá bezpečností webových aplikací. Účelem teoretické části je seznámit se s problematikou bezpečnosti webových aplikací v obecné rovině a poukázat na časté způsoby, kterými lze využít bezpečnostních slabin v aplikacích. První část se podrobněji věnuje projektu Open Web Application Security Project (OWASP), se zaměřením na projekt OWASP Top Ten, jež shrnuje deset nejkritičtějších bezpečnostních zranitelností webových aplikací.

Praktická část práce se věnuje testování zabezpečení vybrané aplikace pomocí penetračního testování. Nejprve je představena samotná aplikace, následuje výběr vhodných nástrojů pro testování a popis průběhu testů. Výstupem této práce je shrnutí výsledků testů dané aplikace, vyhodnocení a zabezpečení nalezených zranitelností.

Klíčová slova

Penetrační testování, bezpečnost webových aplikací, OWASP.

Abstract

This bachelor's thesis deals with the topic of web application security. The purpose of the theoretical section is to introduce the problem of web application security in general and highlight the means of exploiting the security vulnerabilities. The next part of this section is dedicated to the Open Web Application Security Project (OWASP) organization, with the primary focus on the OWASP Top Ten Project, describing the ten most critical web application security vulnerabilities.

The practical section is about testing the security of the given application using penetration testing. After introducing the application, appropriate testing tools are selected and the testing process is described. Finally, the test results are summarized and all found security weaknesses are fixed.

Keywords

Penetration testing, web application security, OWASP.

Obsah

Úvod **1**
Cíl práce . . . 1
Rešerše . . . 2

1 Bezpečnost webových aplikací **3**
1.1 Základní pojmy . . . 3
1.1.1 Webová aplikace . . . 3
1.1.2 Bezpečnost, bezpečnost dat, bezpečnost informací . . . 3
1.1.3 Autentizace a autorizace . . . 4
1.2 Bezpečnostní hrozby na webu . . . 4
1.2.1 Statické webové stránky . . . 4
1.2.2 Dynamické webové stránky . . . 5
1.2.3 Výskyt hrozeb . . . 6

2 Penetrační testování **9**
2.1 Základní pojmy . . . 9
2.2 Dělení testů . . . 9
2.2.1 Manuální vs. automatizované testy . . . 9
2.2.2 Dle úrovně znalostí struktury testovaného systému . . . 10
2.3 Metodologie testování . . . 11
2.4 Srovnání s jinými metodami testování . . . 11

3 OWASP Top Ten Project **13**
3.1 Použité pojmy . . . 14
3.2 Hodnocení rizik . . . 14
3.3 Vývoj pořadí . . . 16
3.4 Top 10 2013 . . . 16
3.4.1 Rozdíly oproti předchozí verzi . . . 17
3.4.2 A1 – Injektování . . . 17
3.4.3 A2 – Chybná autentizace a správa relace . . . 18
3.4.4 A3 – Cross-Site Scripting (XSS) . . . 19

3.4.5 A4 – Nezabezpečený přímý odkaz na objekt 20
3.4.6 A5 – Nezabezpečená konfigurace 21
3.4.7 A6 – Expozice citlivých dat 22
3.4.8 A7 – Chyby v řízení úrovní přístupů 22
3.4.9 A8 – Cross-Site Request Forgery (CSRF) 23
3.4.10 A9 – Použití známých zranitelných komponent 24
3.4.11 A10 – Neošetřené přesměrování a předávání 24
3.5 Top 10 2017 Release Candidate 25

4 Penetrační testování webové aplikace 28
4.1 Představení webové aplikace 28
4.1.1 Architektura aplikace 28
4.2 Testovací prostředí 28
4.3 Použité nástroje 29
4.4 Rozsah a povaha testů 30
4.5 Příprava a konfigurace nástrojů 30
4.5.1 Profil skenování 30
4.5.2 Spuštění testů 32
4.6 Průběh testů a vyhodnocení 33
4.6.1 A1 – Injektování 33
4.6.2 A2 – Chybná autentizace a správa relace 34
4.6.3 A3 – Cross-Site Scripting (XSS) 34
4.6.4 A4 – Nezabezpečený přímý odkaz na objekt 35
4.6.5 A5 – Nezabezpečená konfigurace 35
4.6.6 A6 – Expozice citlivých dat 35
4.6.7 A7 – Chyby v řízení úrovní přístupů 36
4.6.8 A8 – Cross-Site Request Forgery (CSRF) 36
4.6.9 A9 – Použití známých zranitelných komponent 37
4.6.10 A10 – Neošetřené přesměrování a předávání 37
4.6.11 Ostatní problémy 37

5 Právní aspekty 38
5.1 Ochrana osobních údajů 38
5.2 Penetrační testování 38

6 Závěr 40

Seznam zkratek 41

Terminologický slovník **42**

Seznam literatury **42**

Seznam obrázků **50**

Seznam tabulek **51**

Seznam rovnic **52**

Úvod

V dnešní době internetu hraje důležitou roli bezpečnost dat, a to jak z pohledu uživatele, tak vývojáře webových aplikací. Díky stále se zvyšujícímu počtu webových stránek na internetu [1] mají útočníci čím dál větší možnost využít bezpečnostních zranitelností nedostatečně zabezpečených webových aplikací, což může vést v horším případě k odcizení a zneužití údajů, které by se neměly dostat do rukou nepovolaných osob.

Cíl práce

Cílem této práce je otestovat bezpečnost konkrétní webové aplikace pomocí automatizovaného penetračního testování s využitím OWASP, vyhodnotit výsledky testování a navrhnout řešení nalezených zranitelností.

První dvě kapitoly teoretické části jsou věnovány oblasti bezpečnosti webových aplikací se zaměřením na penetrační testování. Další kapitola představuje projekt OWASP, včetně představení deseti nejkritičtějších bezpečnostních zranitelností webových aplikací. Cílem teoretické části je seznámit se s problematikou bezpečnosti webových aplikací a nejčastějšími bezpečnostními riziky v tomto odvětví.

Praktická část se zabývá testováním vybrané webové aplikace. Po představení aplikace a její architektury následuje výběr vhodných nástrojů pro testování a stanovuje se rozsah prováděných testů. Cílem praktické části je vyhodnocení výsledků testů a opravení zjištěných bezpečnostních chyb v aplikaci.

Rešerše

HANZLIKOVÁ, Martina. ***Testování bezpečnosti webových aplikací*** **[2].**

Autorka měla před sebou úkol podobný tomu, který se řeší v této práci – otestovat zabezpečení konkrétní webové aplikace dle metodiky OWASP. V teoretické části je čtenář seznámen se základními pojmy z oblasti bezpečnosti. Zároveň je zde představen projekt OWASP, kde jsou zmíněny důležité pojmy a detailní popis nejkritičtějších bezpečnostních rizik ze seznamu OWASP Top 10 – 2013.

Praktická část se věnuje manuálnímu testování webové aplikace Polarion ALM. Samotnému testování předchází představení aplikace, stanovení cílů, rozsah testů a sběr dat. Závěr praktické části patří shrnutí výsledků testů, včetně výčtu nalezených zranitelností a doporučení k jejich nápravě. Autorka výsledky následně poskytla vedení společnosti Polarion.

KIEZLER, Tomáš. ***Vývoj nástroje na automatizované penetrační testování webových aplikací*** **[3].**

Práce se věnuje automatizovanému penetračnímu testování webových aplikací. Teoretická část popisuje metody testování a předkládá výhody a nevýhody automatického testování oproti manuálnímu. Také se věnuje srovnání několika nástrojů pro automatizované testy, které byly provedeny na konkrétní ukázkové aplikaci. Autor vybíral z nástrojů na seznamu The WIVET Score of Web Application Scanners.

Na rozdíl od mé práce se autor v praktické části pouští do vývoje vlastního nástroje na automatizované penetrační testování. Nástroj je napsán převážně v jazyce PHP a ovládá se přes webové rozhraní.

Kapitola 1

Bezpečnost webových aplikací

Tato kapitola se zabývá bezpečností webových aplikací v obecné rovině. V úvodu jsou uvedeny definice pojmů z oblasti bezpečnosti. Další část popisuje častá rizika na webu z hlediska vývoje webových technologií. V závěru kapitoly je poukázáno na výskyt hrozeb v závislosti na použitém programovacím jazyce.

1.1 Základní pojmy

1.1.1 Webová aplikace

Definice webové aplikace se v různých zdrojích liší. Daniel Nations [4] ve svém článku na webu Lifewire definuje webovou aplikaci takto:

> A Web application is any application that uses a Web browser as a client. The application can be as simple as a message board or a guest sign-in book on a website or as complex as a word processor or a spreadsheet.

Jiná definice podle Jana Bittnera [5]:

> Pokud se bavíme o webové aplikaci, je většinou myšlen právě back end, tedy ta část, při které webovou stránku vytváříme. Může se jednat o nejrůznější redakční systémy, hry apod.

Pro účely této práce je *webová aplikace* definována jako aplikační software běžící na webovém serveru, který komunikuje s webovým prohlížečem klienta pomocí protokolu Hypertext Transfer Protocol (HTTP) [6]; součástí webové aplikace je i front end, neboť tato část bude sloužit jako vstup pro testování aplikace.

1.1.2 Bezpečnost, bezpečnost dat, bezpečnost informací

S bezpečností webových aplikací souvisí několik obecnějších pojmů:

Bezpečnost je „Vlastnost prvku (např. informační systém), který je na určité úrovni chráněn proti ztrátám, nebo také stav ochrany (na určité úrovni) proti ztrátám.“ [7, s. 23].

Bezpečnost dat je „počítačová bezpečnost aplikovaná na data. Zahrnuje například řízení přístupů, definování politik a procesů a zajištění integrity dat“ [7, s. 23].

Bezpečnost informací je „zajištění důvěrnosti, integrity a dostupnosti informací“ [8, s. 1926].

Zatímco *bezpečnost* označuje stav ochrany, pojem *zabezpečení* je v této práci použit zejména ve významu procesu vedoucímu k zajištění bezpečnosti. Tato práce se zabývá především bezpečností dat a informací.

1.1.3 Autentizace a autorizace

Autentizace je proces umožňující jednoznačnou identifikaci klienta využívající danou aplikaci a služby. Klient může být koncový uživatel, jiná služba, procesy nebo počítače [9; 10]. V rámci této práce bude pojem autentizace zúžen na jednoznačnou identifikaci uživatele aplikace na základě poskytnutých identifikačních údajů – např. kombinace uživatelského jména a hesla zadaná do přihlašovacího formuláře.

Autorizace odpovídá na otázku „Co smí uživatel provádět?“. Konkrétně se jedná o „udělení práv, které zahrnuje udělení přístupu na základě přístupových práv. Proces udělení práv subjektu pro vykonávání určených aktivit v informačním systému.“ [7, s. 22].

1.2 Bezpečnostní hrozby na webu

S postupným vývojem webových technologií přicházejí i nová bezpečnostní rizika, a proto je potřeba znát způsoby ošetření bezpečnostních zranitelností vyvíjených aplikací s ohledem na použité technologie.

1.2.1 Statické webové stránky

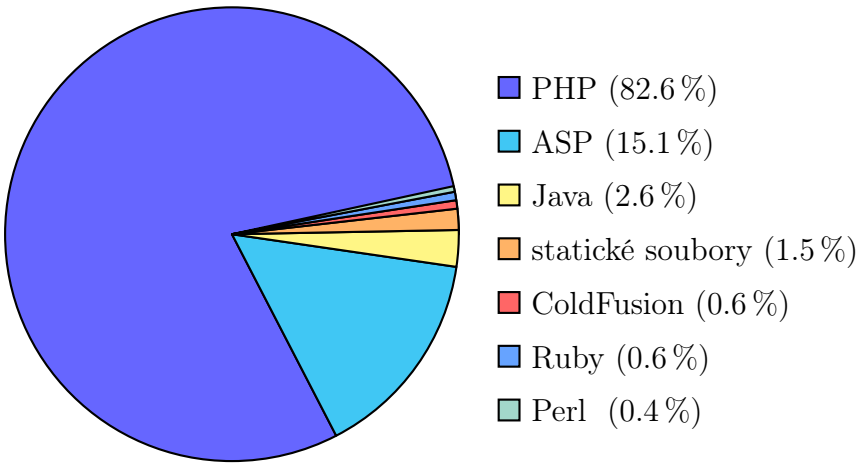
V 90. letech 20. století, kdy web jako takový byl ve svých začátcích, existovaly pouze statické webové stránky, postavené čistě na jazyce Hypertext Markup Language (HTML) [11], popř. doplněné o vizuální styly v podobě Cascading Style Sheets (CSS). Veškerá komunikace klienta se serverem spočívala pouze ve vykreslení stránky bez předchozího zpracování uživatelských vstupů nebo databázových dat na serveru.

To sice znamenalo, že se do HTML dokumentu nemohl dostat uživatelský výstup se škodlivým kódem, nicméně zde existují jiná rizika. Server může být nakonfigurován tak, že umožní uživatelům vypsat adresářovou strukturu jakékoliv složky dostupné pod veřejnou adresou Uniform Resource Locator (URL). Více o tomto riziku v podkapitole 3.4.6.

1.2.2 Dynamické webové stránky

O několik let později s nástupem dynamických programovacích jazyků PHP: Hypertext Preprocessor (PHP) [12] nebo Active Server Pages (ASP) se objevily nové druhy zranitelností. Dynamické skriptovací programovací jazyky umožňují vkládat kamkoliv do HTML stránky kód, který je před odesláním klientovi zpracován interpreterem na straně serveru, což umožňuje na stránce zobrazit prvky, které se mění v závislosti na aplikační logice. Tyto jazyky se často využívají k obsluze uživatelských vstupů z formulářů nebo pro výpis dat z databáze.

V současné době téměř každý web využívá nějaký dynamický programovací jazyk. Obrázek 1.1 znázorňuje výsledky průzkumu portálu W3Techs, který pravidelně prochází 10 milionů nejpopulárnějších webů dle společnosti Alexa [13]. Z grafu je zřejmé, že statické soubory mají v dnešní době na celkovém podílu použitých technologií minimální zastoupení, zatímco první pozici zaujímá jazyk PHP s většinovým podílem.



Obrázek 1.1: Zastoupení serverových programovacích jazyků na webu
(Pozn.: jeden web může využívat i více než jeden jazyk)
Data se vztahují k 5. dubnu 2017 (Zdroj: [14])

Jako příklad zranitelnosti dynamických webových stránek je uveden HTML dokument, do kterého je vložen skript v jazyce PHP (vyznačen zeleně kurzívou).

```
<?php $jmeno = 'Karel'; ?>
<html>
  <body>
    Jmenuji se <?php echo $jmeno; ?>
  </body>
</html>
```

Před odesláním obsahu stránky do webového prohlížeče PHP nalezne všechny výskyty skriptů uzavřených mezi značkami `<?php` a `?>`[1] a přeloží je do čistého HTML. Ve výše uvedeném kódu tedy nahradí proměnnou `$jmeno` hodnotou `Karel`. Výsledný kód, který „uvidí“ webový prohlížeč, by vypadal takto:

```
<html>
  <body>
    Jmenuji se Karel
  </body>
</html>
```

Problém je v tom, že hodnota proměnné nemusí být nutně jen prostý text, ale může obsahovat i HTML tagy nebo jakýkoliv jiný kód, což znamená, že může být na stránce spuštěn např. kód v JavaScriptu. Jazyk JavaScript, který se dostal do povědomí ve stejné době jako dynamické programovací jazyky [15], je skriptovací jazyk, který se spouští zpravidla ve webovém prohlížeči uživatele[2] a lze v něm dynamicky manipulovat s prvky v DOM (Document Object Model). Možnosti JavaScriptu skýtají velké riziko – jakýkoliv kód vložený takto do stránky se automaticky spustí v prohlížeči. Tento druh zranitelnosti se nazývá XSS (Cross-Site Scripting) a je považován za nejčastější slabinu v zabezpečení webových aplikací [16].

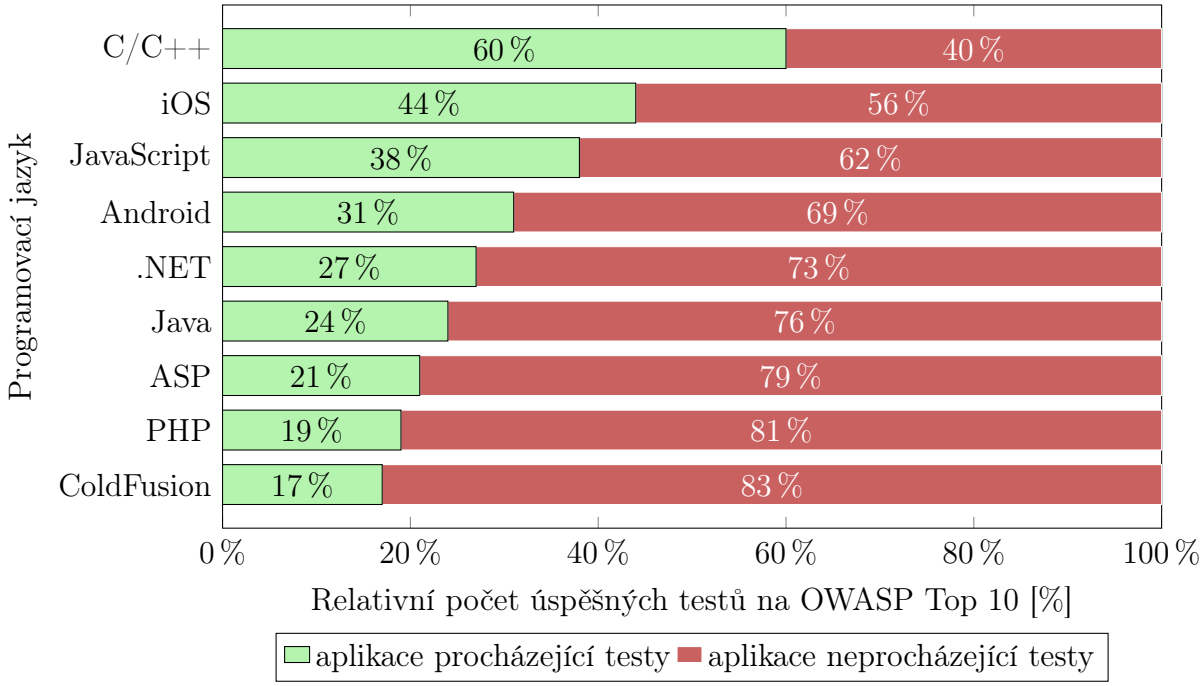
1.2.3 Výskyt hrozeb

Společnost Veracode, která více než 10 let poskytuje služby na poli bezpečnosti aplikací, provádí analýzu zdrojových kódů aplikací svých zákazníků a pravidelně sestavuje reporty o stavu zabezpečení softwaru [20; 21]. Jelikož zákazníci Veracode nejsou pouze provozovatelé webových aplikací, ale i čistě klientských programů, průzkum ukazuje srovnání napříč technologiemi bez ohledu na cílovou platformu.

[1] Až do verze PHP 7.0.0 je možné použít také ASP tagy `<%`, `%>`, `<%=` nebo tag `<script language="php">` [17]. Také je možné použít i počáteční tag `<?=` pro zkrácení příkazu `echo` [18].

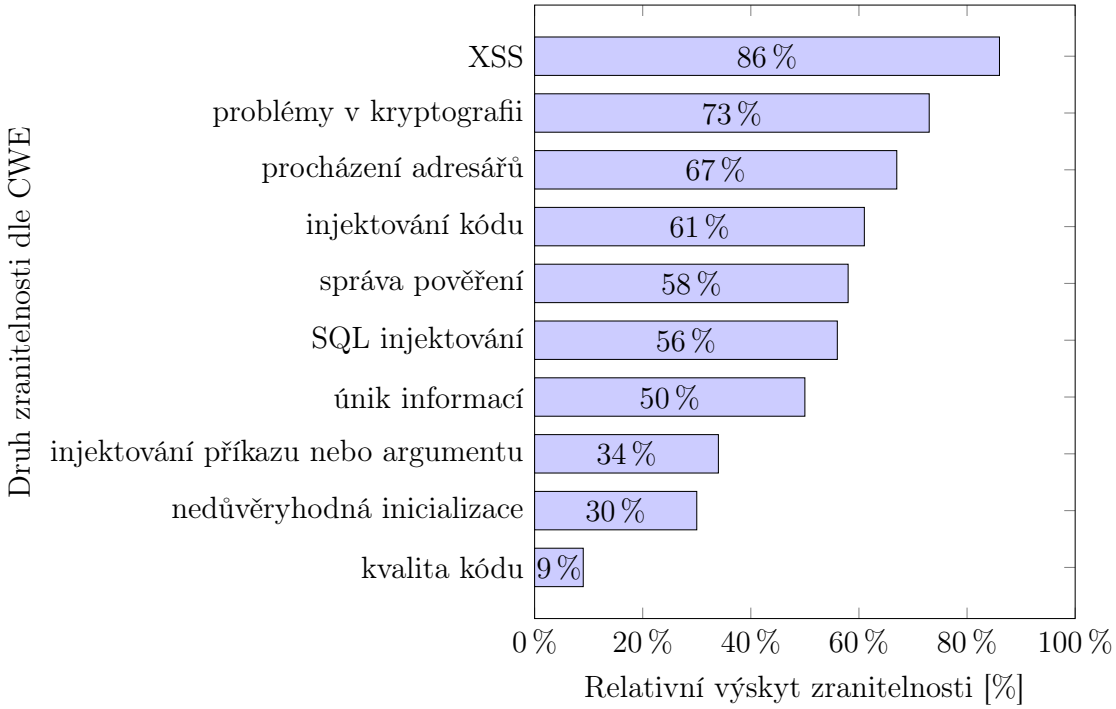
[2] JavaScript je možné použít i na straně serveru. Tuto architekturu využívá např. Node.js [19].

Vydání průzkumu z roku 2015 srovnává výskyt zranitelností podle programovacích jazyků z balíku dat od 1. října 2013 do 31 března 2015 [22, s. 15]. Jak je vidět na obrázku 1.2, aplikace využívající skriptovací programovací jazyky jako PHP nebo ASP mají výrazně nižší procento úspěšnosti v testech nejčastějších zranitelností dle OWASP Top 10 oproti aplikacím napsaných v kompilovaných jazycích, jako je C/C++, nebo mobilním aplikacím.



Obrázek 1.2: Výsledky testů OWASP Top 10 dle programovacích jazyků (Zdroj: [22, s. 5])

Obrázek 1.3 znázorňuje relativní počet webových aplikací v PHP seskupených dle kategorie nalezených zranitelností podle CWE (Common Weakness Enumeration). Tato data poukazují na skutečnost, že procházení adresářů a XSS, uvedené v částech 1.2.1 a 1.2.2, se řadí mezi nejčastější zranitelnosti v dynamických webových aplikacích.



Obrázek 1.3: Zastoupení druhů zranitelností dle CWE ve webových aplikacích v PHP
(Zdroj: [22, s. 7])

Kapitola 2

Penetrační testování

Pro zajištění bezpečnosti webové aplikace je třeba efektivně odhalit co možná nejvíce bezpečnostních zranitelností, k čemuž slouží penetrační testování. V úvodu kapitoly jsou vysvětleny základní pojmy spojené s tímto testováním a cíl penetračních testů. Následuje rozdělení zmíněných testů a popis jednotlivých fází testování. Poslední část kapitoly se věnuje srovnání tohoto druhu testování s ostatními metodami testování bezpečnosti.

2.1 Základní pojmy

Penetrační testování lze obecně definovat jako proces zhodnocení odolnosti informačního systému proti útokům, které mohou vést k neoprávněnému přístupu.

Penetrační test (někdy také známý jako „etické hackování“) je prováděn za účelem získání vyhodnocení stavu ochrany systému nebo analýzy zranitelností systému [23, s. 68]. Cílem penetračního testování je ověřit úroveň zabezpečení daného systému.

2.2 Dělení testů

Testy bezpečnosti aplikací lze obecně rozdělit na manuální a automatizované. Testy se také liší podle úrovně znalostí testovaného systému [24, s. 15–17].

2.2.1 Manuální vs. automatizované testy

Penetrační testy lze provádět manuálně nebo je možné je zautomatizovat. Oba způsoby mají své výhody a nevýhody. Manuální testy se provádějí ručně, zatímco automatizované testy využívají speciálních počítačových programů.

Manuální testy

Manuální testy jsou vykonávány ručně samotným testerem. Výhodou je, že si tester může testy navrhnout tak, aby vyhovovaly podmínkám pro danou aplikaci.

A protože testy provádí člověk a ne software, tester dokáže popsat průběh a vyhodnocení testů lidským způsobem. Na druhou stranu, manuální testy jsou obvykle časově náročnější – tester musí přesně znát vlastnosti webové aplikace z pohledu použitých technologií, aby mohl testy zacílit na ty správné zranitelnosti, což vyžaduje i určitou úroveň znalostí v daném oboru.

Automatizované testy

Největší výhodou automatizovaných testů je jejich rychlost a velký výběr možností testování. Software pro automatizované testování je obvykle uzpůsoben pro širokou škálu aplikací. Mezi nevýhody patří nutné odborné znalosti v testované oblasti kvůli správné interpretaci výsledků.

Semiautomatické testy

Existují i testy semiautomatické, které jsou kombinací manuálních a automatických testů. Snaží se kombinovat výhody obou forem testů.

2.2.2 Dle úrovně znalostí struktury testovaného systému

Black-box testy

Black-box testy simulují útok na aplikaci zvenčí, bez znalostí její vnitřní struktury. Známy jsou pouze vstupy a možné výstupy aplikace. Při provádění black-box testů není potřeba znát programovací jazyky nebo jiné technologie, které jsou použity v testované aplikaci. Tento druh testů však obvykle neodhalí chyby, které vyžadují přímý přístup do zdrojového kódu aplikace. Mezi black-box testy se řadí i penetrační testy [25, s. 14].

White-box testy

Na rozdíl od black-box testů mají white-box testy plný přístup k testované aplikaci, chyby se proto dají hledat přímo ve zdrojovém kódu. To však vyžaduje znalost použitého programovacího jazyka, což je i jednou z nevýhod tohoto druhu testů. Další nevýhodou je jejich časová náročnost. Naopak velkou výhodou je výše zmíněná znalost zdrojového kódu – na základě nalezených chyb je možné rovnou provést optimalizaci kódu a zabezpečení aplikace. Mezi white-box testy patří ruční kontrola nebo statická analýza zdrojového kódu [25, s. 19].

Grey-box testy

Grey-box (nebo také gray-box) testy kombinují výhody předchozích dvou druhů testů. Využívá se znalosti vnitřní struktury aplikace, zejména zdrojového kódu, ale testy jsou prováděny black-box metodou, tedy z pohledu uživatele aplikace.

2.3 Metodologie testování

Pravidla průběhu penetračního testování nejsou pevně určena, neboť žádné dvě webové aplikace nejsou stejné – každý systém vyžaduje odlišný přístup, jiné nástroje, záleží také na rozsahu testů nebo cílech testování. Existuje mnoho metodik popisujících postupy při penetračním testování, z nichž každá uvádí jiné fáze testování. Obecně se však dají za klíčové kroky považovat [24, s. 12]

- cíl a rozsah penetračního testu,
- sběr dat,
- skenování a exploitace,
- report.

OWASP Foundation shrnuje 7 konkrétních metodik penetračního testování [26]:

- OWASP testing guide
- PCI Penetration testing guide
- Penetration Testing Execution Standard
- NIST 800-115
- Penetration Testing Framework
- Information Systems Security Assessment Framework (ISSAF)
- Open Source Security Testing Methodology Manual (OSSTMM)

2.4 Srovnání s jinými metodami testování

Penetrační testování není jediná technika používaná pro vyhledávání bezpečnostních slabin v aplikacích. Mezi další techniky se řadí inspekce a kontroly prováděné lidmi, modelování hrozeb a revize zdrojového kódu [25, s. 13].

Je nutné zmínit, že penetrační testování zdaleka neodhalí 100 % potenciálních zranitelností. Jako názorný příklad je uvedena situace, kde webová aplikace obsahuje skript, který dovoluje provádět operace pod administrátorskými oprávněními pouhým zadáním správné hodnoty parametru v URL [25, s. 15]. Taková URL adresa může vypadat například takto:

```
http://www.mysite.net/app?magic=value
```

Parametr `magic` zde nabývá hodnoty `value`. Kód, který zpracovává tento požadavek může vypadat následovně:

```
public function applicationMethod(Request $request)
{
  if ($request->get('magic') === 'D3Oik6zvBNohhcu2fs1Z') {
    $this->doAdminStuff($request);
  }
    // pokračování logiky...
}
```

Pokud se parametr `magic` v příchozím HTTP požadavku typu GET rovná hodnotě `D3Oik6zvBNohhcu2fs1Z`, provede se operace, která je dovolená pouze administrátorům. Tento způsob úmyslného obcházení administrátorských práv se používá během vývoje a testování aplikace, nicméně se může stát, že kód zůstane v aplikaci i v produkčním prostředí.

Jestliže se daná stránka otestuje automatickým testovacím programem, zranitelnost se nejspíše neodhalí. Skener by musel projít všechny možné kombinace znaků, které mohou být obsaženy v parametru `magic`, což by při délce 20 znaků znamenalo poslat obrovský počet požadavků – při omezení na velká/malá písmena anglické abecedy a číslice by se muselo vyhodnotit až 20^{62} požadavků, než by správnou kombinaci program odhalil. Naproti tomu běžná revize zdrojového kódu by tuto bezpečnostní slabinu odhalila mnohem efektivněji.

Kapitola 3

OWASP Top Ten Project

Open Web Application Security Project (OWASP) je otevřená komunitní platforma, která si klade za cíl poskytovat ucelené a nezaujaté informace týkající se zabezpečení webových aplikací, a zlepšovat tím povědomí o stavu bezpečnosti v tomto odvětví [27]. Celý portál je založen na systému Wiki[1], a tak může do projektu přispívat každý. Jedinou podmínkou je registrace uživatelského účtu, který musí být předem schválen.

OWASP spravuje celou řadu podprojektů, z nichž každý se zabývá určitou podmnožinou úkolů (dále jen „projekt“). Mezi hlavní projekty se řadí [28]:

- Nástroje
  - OWASP Zed Attack Proxy (ZAP)
  - OWASP OWASP Web Testing Environment Project
  - OWASP OWTF
  - OWASP Dependency Check
- Zdrojový kód
  - OWASP ModSecurity Core Rule Set Project
  - OWASP CSRFGuard Project
  - OWASP AppSensor Project
- Dokumentace
  - OWASP Application Security Verification Standard Project
  - OWASP Software Assurance Maturity Model (SAMM)
  - OWASP AppSensor Project
  - OWASP Top Ten Project
  - OWASP Testing Guide Project

[1] Wiki je web, na který mohou uživatelé přímo přidávat obsah nebo jej měnit [29]

OWASP Top Ten je projekt poskytující přehled nejkritičtějších zranitelností webových aplikací, zveřejňovaný ve formě uceleného dokumentu. Je to výsledek snahy mnoha bezpečnostních expertů z celého světa, jež se snaží zvýšit povědomí o častých hrozbách. První vydání OWASP Top 10 bylo zveřejněno v roce 2003, první revize následovala hned o rok později a od té doby se objevovaly nové verze každé tři roky až do roku 2013, kdy vyšla zatím poslední verze dokumentu [30].

3.1 Použité pojmy

Popis každého rizika na seznamu OWASP Top 10 je rozdělen na pět částí, které znázorňují postupný průchod útočníka aplikací s cílem způsobit škodu. Následují jejich definice:

Původce hrozby *(threat agent)* je jakýkoliv prostředek (předmět, látka, software, člověk atd.) schopný způsobit újmu na majetku [31, s. 37].

Vektor útoku *(attack vector)* je cesta nebo prostředek, jež může umožnit útočníkovi využít bezpečnostních zranitelností systému k doručení škodlivého obsahu nebo dosažení škodlivého výsledku. Mezi vektory útoku se řadí viry, e-mailové přílohy, webové stránky, vyskakovací okna nebo také podvod [32].

Bezpečnostní slabina *(security weakness)* je slabé místo v systému, které může představovat bezpečnostní zranitelnost, je-li zneužita útočníkem pro škodlivou činnost.

Technický dopad *(technical impact)* je dopad na systém jako důsledek úspěšného útoku. Obecně se jedná o míru oslabení bezpečnosti informací [33].

Business dopad *(business impact)* je dopad na společnost z hlediska byznysu. Může se jednat o snížení zisků, ztrátu reputace společnosti, úbytek zákazníků nebo ztrátu na lidských životech [34].

3.2 Hodnocení rizik

Až do vydání z roku 2007 byl seznam OWASP Top 10 řazen výhradně podle rozšířenosti zranitelností. Počínaje rokem 2010 OWASP používá pozměněnou metodiku výpočtu vycházející z OWASP Risk Rating Methodology, jež se soustřeďuje na hodnocení bezpečnostních rizik plynoucích ze zranitelností.

Riziko se obecně vypočítá podle následující rovnice [35]:

$$\text{riziko} = \text{pravděpodobnost} \times \text{dopad} \tag{3.1}$$

OWASP Top 10 definuje pravděpodobnost jako průměr zneužitelnosti, rozšířenosti a odhalitelnosti – viz rovnice 3.2. Dopad vyjadřuje pouze technický dopad na systém [36, s. 20].

$$\text{riziko} = \frac{\text{zneužitelnost} + \text{rozšířenost} + \text{odhalitelnost}}{3} \times \text{dopad} \tag{3.2}$$

Čím menší vyjde priorita rizika, tím výše v seznamu je riziko zařazeno. Nutno dodat, že OWASP tato rizika řadí na základě průzkumu a statistické analýzy velkého množství dat, a tak není možné tento seznam brát jako směrodatný pro každou aplikaci. Výpočet rizika nezahrnuje původce hrozeb nebo business dopady, které jsou často specifické pro konkrétní systém. Pro společnosti, které si chtějí zjistit přesnější hodnocení rizik, OWASP doporučuje sestavit žebříček odpovídající reálnému riziku s využitím informací o původcích hrozeb a business dopadu dané společnosti. [36, s. 21].

Metodika výpočtu pořadí rizik v OWASP Top 10 tedy pracuje celkem se čtyřmi faktory, které spadají pod jednotlivé kategorie založené na obecné metodice OWASP pro hodnocení rizik – viz tabulka 3.1. Každému faktoru je přiřazena hodnota od 1 do 3, kde 1 znamená nejvyšší a 3 nejnižší pravděpodobnost výskytu daného faktoru. Výjimku tvoří položka A3 – Cross-Site Scripting (XSS), které je v rozšířenosti přiřazena hodnota 0 (velmi rozšířená).

Kategorie	Faktor	Stupeň hodnocení		
		1	2	3
Původci hrozeb	—		—	
Vektory útoku	zneužitelnost	jednoduchá	průměrná	obtížná
Bezpečnostní slabiny	rozšířenost	rozšířená	běžná	vzácná
	odhalitelnost	snadná	průměrná	obtížná
Technické dopady	dopad	vážný	přiměřený	nevýznamný
Business dopady	—		—	

Tabulka 3.1: Faktory výpočtu hodnocení rizik OWASP Top 10
(Zdroj: [36, s. 5, 20])

3.3 Vývoj pořadí

V průběhu let se přirozeně objevují nová rizika a jiná zanikají, nebo už nejsou tak rozšířená. Například riziko „Injektace“ se ze začátku drželo v dolní polovině seznamu, nicméně v posledních deseti letech se díky své rozšířenosti postupně dostalo až na první pozici. Oproti tomu ještě do roku 2004 bylo běžné kontrolovat vstupy od uživatelů webových aplikací pouze na straně klienta, což umožňovalo obejít kontrolní mechanismy odesláním vlastních HTTP požadavků – proto se v prvních dvou verzích objevila položka „Neošetřené vstupy“, a to hned na prvním místě [37]. V roce 2007 se OWASP rozhodl toto riziko ze seznamu odebrat, zároveň však zmiňuje kontrolu vstupních dat jako doporučení při řešení některých dalších rizik [38].

3.4 Top 10 2013

Výsledky penetračního testování aplikace v praktické části budou kategorizovány dle poslední schválené verze OWASP Top 10 z roku 2013. Pořadí rizik vychází z analýzy 8 datasetů od 7 společností zaměřujících se na bezpečnost aplikací, včetně čtyř poradenských společností a tří dodavatelů softwarových nástrojů nebo SaaS [36]. Pořadí 10 nejkritičtějších rizik dle OWASP Top 10 2013 vypadá následovně:

- A1 – Injektování
- A2 – Chybná autentizace a správa relace
- A3 – Cross-Site Scripting (XSS)
- A4 – Nezabezpečený přímý odkaz na objekt
- A5 – Nezabezpečená konfigurace
- A6 – Expozice citlivých dat
- A7 – Chyby v řízení úrovní přístupů
- A8 – Cross-Site Request Forgery (CSRF)
- A9 – Použití známých zranitelných komponent
- A10 – Neošetřené přesměrování a předávání

3.4.1 Rozdíly oproti předchozí verzi

Vydání z roku 2013 doznalo několika změn v pořadí položek oproti předchozí verzi – viz tabulka 3.2. Položka „Chybná autentizace a správa relace“ se posunula na druhou pozici, neboť se na tento typ rizika klade větší důraz. Cross-Site Request Forgery (CSRF) naopak klesl o tři příčky na 8. místo. OWASP to zdůvodňuje domněnkou, že společnosti a vývojáři frameworků se na tento druh útoku soustředí natolik, že počet zranitelností vůči CSRF výrazně klesl.

Pořadí	OWASP Top 10 2010	OWASP Top 10 2013	Změna
A1	Injektování	Injektování	
A2	Cross Site Scripting (XSS)	Chybná autentizace a správa relace	↑
A3	Chybná autentizace a správa relace	Cross Site Scripting (XSS)	↓
A4	Nezabezpečený přímý odkaz na objekt	Nezabezpečený přímý odkaz na objekt	
A5	Cross-Site Request Forgery (CSRF)	Nezabezpečená konfigurace	↑
A6	Nezabezpečená konfigurace	Expozice citlivých dat	↑
A7	Nezabezpečené kryptografické úložiště	Chyby v řízení úrovní přístupů	↓
A8	Nesprávné řízení přístupu přes URL	Cross-Site Request Forgery (CSRF)	↓
A9	Nedostatečná ochrana transportní vrstvy	Použití známých zranitelných komponent	
A10	Neošetřené přesměrování a předávání	Neošetřené přesměrování a předávání	

nově přidané položky sloučené položky

Tabulka 3.2: Srovnání rizik OWASP Top 10 2013 oproti verzi z roku 2010 (Zdroj: [36, s. 4])

3.4.2 A1 – Injektování

Injektování spočívá ve vkládání škodlivého kódu do místa v aplikaci, ze kterého je možné tento kód spustit pomocí interpreteru. Vkládat lze SQL, LDAP, XPath či NoSQL dotazy nebo spouštět systémové příkazy. Injektování může vést ke ztrátě nebo nechtěnému pozměnění dat, v horších případech může útočník získat přístup k celému hostitelskému prostředí. Z business hlediska to může znamenat ztrátu citlivých dat [36, s. 4].

Obzvláště rozšířená je technika SQL injektování, která v závislosti na úrovni zranitelnosti aplikace umožňuje spustit libovolný dotaz v jazyce Structured Query Language (SQL). Pomocí SQL se lze dotazovat na databázové úložiště přístupné přes webovou aplikaci – nejčastěji se jedná o CRUD operace, které lze používat v rámci jednotlivých řádků/sloupců databázových tabulek, ale i celých databází.

Příklad zranitelnosti

Webová aplikace obsahuje stránku, která umožňuje zobrazit údaje o uživateli. Pro identifikaci uživatele využívá hodnotu URL parametru `id`, který se porovnává s hodnotou primárního klíče `id` v databázi uživatelů (`users`). Následující PHP kód vezme parametr `id` z HTTP požadavku a vloží jej přímo do SQL dotazu.

```
$result = $mysqli->query('SELECT * FROM users WHERE id = ' . $_GET['id']);
```

Útočník toho může zneužít tak, že za hodnotu parametru `id` dosadí například řetězec `1 OR 1 = 1`. Výsledná podmínka `WHERE` se tudíž vyhodnotí jako pravdivá a místo jednoho uživatele se ve výsledku dotazu objeví všichni uživatelé v databázi.

Další způsob zneužití spočívá ve vložení dalšího příkazu hned za příkaz `SELECT`, neboť v SQL lze řetězit více příkazů za sebou pomocí středníku. Pokud tedy bude chtít útočník smazat celou databázi uživatelů, lze za parametr `id` dosadit hodnotu `1; DROP DATABASE users` a výsledný řetězec, který se odešle jako SQL příkaz databázi, bude vypadat takto:

```
SELECT * FROM users WHERE id = 1; DROP DATABASE users
```

Obrana proti útoku

Nejjednodušším způsobem obrany proti injektování je vyhnout se přímému přístupu k interpretovaným příkazům. Velké množství systémových příkazů lze provádět pomocí specializovaných knihoven. Mnohdy se však nelze interpretovaným příkazům vyhnout, jako při přístupu k databázi přes SQL. V takovém případě se doporučuje využít tzv. stored procedures nebo prepared statements, které berou vstupní hodnoty jako data, nikoliv jako část zdrojového kódu. Další možností je vytvoření „white listu“, který dovolí pouze vybrané bezpečné vstupní hodnoty; white list se využívá např. při řazení databázových záznamů na stránce. Jedním z doporučení je provést escapování problematických znaků ve vstupních datech před vložením do příkazu, nicméně tento způsob má oproti výše zmíněným technikám nižší účinnost [39].

3.4.3 A2 – Chybná autentizace a správa relace

Webové aplikace využívají autentizaci při správě uživatelských přístupů. Autentizace uživatele obvykle začíná zadáním uživatelského jména a hesla do webového formuláře, které aplikace porovnává s úložištěm na serveru. Jelikož je HTTP protokol bezstavový [6], pro uchování stavu mezi požadavky se využívá relace (session). Správa relací nemusí být vždy implementována správně, a tak může dojít k odcizení

identifikátoru relace, což dovolí útočníkovi vydávat se za jiného uživatele. Útočník se může také zmocnit uživatelských hesel, pokud nejsou ukládána pomocí hashovacího algoritmu nebo využívají příliš slabé algoritmy [36, s. 4].

Příklad zranitelnosti

Webová aplikace sloužící jako diskuzní fórum vkládá ID relace přímo do URL pomocí parametru `sessid`.

```
http://forum.net/topic?id=422891&sessid=2nl9gueq0mva58e7bg6dvk4s1h
```

Uživatel přihlášený do aplikace může tento odkaz odeslat jiným lidem, aniž by věděl, že se v něm nachází i identifikátor jeho relace. Kdokoliv tento odkaz použije, bude zároveň přihlášen pod identitou daného uživatele.

Obrana proti útoku

Relace by měly být ukládány ve formě cookies ve webovém prohlížeči uživatelů. Cookies jsou díky svým možnostem nastavení data vypršení a řadou dalších nastavitelných omezení preferovaným způsobem ukládání relací. Řada webových frameworků nabízí svá vlastní řešení správy relací, která by měla být upřednostněna před vlastní implementací [40].

Co se týče ukládání hesel, doporučuje se pro generování hashe použít dostatečně bezpečný algoritmus. Více o ukládání hesel a dalších citlivých údajů v části A6 – Expozice citlivých dat, viz 3.4.7.

3.4.4 A3 – Cross-Site Scripting (XSS)

XSS je nejrozšířenější zranitelností webových aplikací. Nastává tehdy, kdy jsou do prohlížeče odeslána neošetřená uživatelská data, nebo data bez escapování. Takto může útočník spustit libovolný skript v prohlížeči oběti a zmocnit se tak uživatelských relací nebo přesměrovat oběť na web se škodlivým nebo podvodným obsahem. Webový prohlížeč nedokáže rozpoznat, zda se jedná o skript pocházející přímo ze zdrojového kódu aplikace, nebo škodlivý kód vložený útočníkem, protože odpověď od serveru obsahuje pouze výsledný HTML dokument [36, s. 9].

Existují tři druhy XSS – stored, reflected a DOM Based. Při stored XSS je uživatelský vstup ukládán na server, nejčastěji do databáze v podobě komentáře, příspěvku apod. a může být kdykoliv znovu zobrazen na stránce [41]. Reflected XSS spočívá v jednorázovém zobrazení uživatelského vstupu v podobě chybové hlášky, výsledku hledání nebo jiné odpovědi od serveru obsahující neošetřený uživatelský

vstup, který nebyl předtím nikam trvale uložen. DOM Based XSS funguje pouze na straně prohlížeče uživatele v podobě modifikovaného DOM – žádná uživatelská data se neodesílají na server.

Příklad zranitelnosti

Příkladem stored XSS může být HTML formulář, do kterého uživatel vyplňuje uživatelské jméno. Část formuláře s polem pro uživatelské jméno může vypadat takto:

```
<input type="text" name="username">
```

Vyplněné jméno se po úspěšném odeslání formuláře zobrazí uživateli na stránce. Ve zdrojovém kódu by výpis jména vypadal takto:

```
Vaše uživatelské jméno je: $_POST['username'];
```

Útočník toho může zneužít tak, že jako hodnotu uživatelského jména pošle řetězec `<script>alert('útok');</script>` a oběti se po odeslání formuláře objeví vyskakovací okno s titulkem „útok“. Pomocí JavaScriptu lze manipulovat s celým DOM nebo cookies uživatele, které lze odesílat na vzdálený server útočníka.

Obrana proti útoku

Primární obrana proti XSS je správné escapování uživatelských dat v závislosti na kontextu v HTML dokumentu. Existují rozdílné způsoby escapování obsahu uvnitř tagu `<body>`, v atributu, uvnitř JavaScriptu, v CSS nebo URL.

3.4.5 A4 – Nezabezpečený přímý odkaz na objekt

Přímé odkazy na objekty ve webové aplikaci (soubory, složky, záznamy v databázi) se považují za nezabezpečené, pokud nejsou dostatečně chráněny systémem řízení přístupu – útočník má možnost získat přístup k objektům, které mu nepatří [36, s. 10].

Příklad zranitelnosti

Webová aplikace nabízí možnost zobrazit přílohy od uživatele. Tyto přílohy jsou soukromého charakteru a neměly by být viditelné nikým jiným, než uživatelem, který je nahrál.

URL pro zobrazení přílohy vypadá takto:

```
http://webapp.net/viewattachment?name=moje-priloha.pdf
```

Za parametr `name` lze dosadit název souboru uloženého na serveru ve složce se všemi přílohami. Útočník toho může zneužít tak, že vyzkouší jiné názvy souborů, a pokud soubor existuje, zobrazí se mu ve webovém prohlížeči, nehledě na původního vlastníka souboru. Další riziko spočívá v možnosti procházení adresářové struktury mimo složku s přílohami – nezabezpečená aplikace tedy může dovolit zadat např. `../../tajny-soubor.txt`, což odkazuje na soubor umístěný o dvě úrovně výše.

Obrana proti útoku

Je potřeba zajistit, aby uživatel přistupující k danému objektu měl k jejímu zobrazení dostatečná oprávnění. Místo odkazování na skutečné názvy souborů je vhodné názvy ukládat v databázi a odkazovat se na ně přes unikátní ID, které bude použito jako URL parametr místo názvu souboru. Tím se zabrání nechtěnému přístupu k celé adresářové struktuře.

3.4.6 A5 – Nezabezpečená konfigurace

Provoz webové aplikace závisí na mnoha faktorech. Roli zde hraje zvolená konfigurace operačního systému, nastavení webového a databázového serveru nebo použitý framework. Nesprávné nastavení kterékoliv komponenty zodpovědné za běh aplikace může představovat bezpečnostní riziko.

Příklad zranitelnosti

Server může být nakonfigurován tak, že při zadání cesty k existujícímu adresáři do URL se v prohlížeči vypíše obsah adresáře. Útočník je tak schopen nalézt soubory obsahující citlivé údaje nebo aplikační soubory, které lze využít k odhalení dalších bezpečnostních zranitelností [36, s. 11].

Dalším příkladem je špatná konfigurace serveru aplikace, kdy se při chybě v aplikaci uživateli zobrazí v prohlížeči veškeré podrobnosti o chybě, včetně cest ke zdrojovým souborům, použitých proměnných nebo dokonce částí zdrojového kódu. Většinou se jedná o pozůstatek z vývojářského prostředí; v produkčním prostředí by se neměl koncový uživatel aplikace dozvědět o vzniklé chybě žádné podrobnosti.

Obrana proti útoku

Veškerý software související s aplikací by měl být udržován aktuální – této problematice se věnuje také položka A9 – Použití známých zranitelných komponent, viz část 3.4.10. Vývojové a produkční prostředí by se mělo lišit co nejméně, aby se zabránilo nesrovnalostem mezi konfiguracemi [36, s. 11]. Všechny informace o chybách v aplikaci by se měly ukládat do log souborů umístěných mimo veřejně dostupnou část aplikace. Zobrazování chyb uživatelům na stránce by mělo být zakázáno v konfiguraci serveru.

3.4.7 A6 – Expozice citlivých dat

Velké množství webových aplikací dostatečně nechrání citlivá data, jako jsou přihlašovací údaje nebo čísla platebních karet uživatelů. Únik citlivých údajů může pro společnost představovat ztrátu reputace a v mnoha případech může být i v rozporu s legislativou [36, s. 12].

Příklad zranitelnosti

Webová aplikace nepoužívá SSL na stránkách s autentizací uživatele. Pokud se oběť nachází např. ve veřejné síti Wi-Fi, útočník je schopen odchytávat síťový provoz a zmocnit se cookie s ID relace oběti. Útočník tuto cookie použije ve svém prohlížeči, čímž získá plný přístup k soukromým údajům oběti [36, s. 12].

Jiný případ úniku citlivých dat může nastat, když je díky chybě v aplikaci útočník schopen získat přístup k souboru se všemi hesly.

Obrana proti útoku

Pokud je to možné, doporučuje se využít šifrované spojení minimálně při přihlašování uživatele nebo při zadávání jakýchkoliv citlivých údajů.

Hesla by měla být ukládána pomocí bezpečného algoritmu. Algoritmy jako SHA-256 [42] jsou sice bezpečné, nicméně pro hashování hesel existují specializované algoritmy (např. bcrypt, PBKDF2 nebo scrypt), které jsou odolnější vůči útokům. Zároveň se doporučuje při vytváření hashe využít tzv. sůl, která zamezí útoku pomocí rainbow table [36, s. 12].

3.4.8 A7 – Chyby v řízení úrovní přístupů

Webové aplikace mnohdy využívají řízení uživatelských přístupů – ať už se jedná o jednoduchou registraci a přihlášení uživatele, nebo složitější autorizaci v podobě

přístupových práv pro jednotlivé uživatele. Podle toho, k jakým aplikačním funkcím má daný uživatel přístup, se zobrazí jiné prvky v GUI [36, s. 13].

Příklad zranitelnosti

Webová aplikace zobrazuje odkaz na administrační část webu pouze uživatelům s příslušným oprávněním: `http://webapp.net/?action=admin`. Útočník je schopen získat přístup ke stejné funkcionalitě pouhým odesláním stejného požadavku bez využití grafického rozhraní a pokud není zajištěna kontrola přístupů v logické části aplikace, může útočník získat neoprávněný přístup k interním údajům.

Obrana proti útoku

Vždy je třeba provést kontroly oprávnění i na straně serveru / v business logice aplikace, neboť uživatel se k dané funkcionalitě může dostat více způsoby. Pouhé skrytí prvků na front endu nestačí.

3.4.9 A8 – Cross-Site Request Forgery (CSRF)

CSRF je druh útoku, který přinutí oběť provést určitou akci bez jejího vědomí ve webové aplikaci, ke které je oběť přihlášena. Pokud útočník zná strukturu požadavků, které se odesílají na server, může oběti podstrčit odkaz umístěný na jiném webu nebo např. v e-mailu. Protože je uživatel přihlášený do aplikace, s každým požadavkem se přenáší i jeho cookies s relací. Takto může útočník vykonávat akce, které jsou normálně povoleny pouze uživatelům s příslušným oprávněním, a změnit tak stav aplikace – upravit osobní údaje nebo provádět platby [36, s. 14].

Příklad zranitelnosti

Aplikace umožňuje platbu on-line přes webový formulář, který je přístupný pouze přihlášeným uživatelům. Po odeslání formuláře vypadá požadavek takto:

```
http://webapp.net/transaction.php?amount=40000&toAccount=5478467
```

Takto odeslaný požadavek odešle 40 000 jednotek měny na účet č. 5478467. Útočník může podobnou URL umístit jako neviditelný obrázek na svůj web:

```
<img
 ↪  src="http://webapp.net/transaction.php?amount=100000&toAccount=1234567"
 ↪  style="width: 0; height: 0">
```

Jakmile oběť tuto stránku navštíví, webový prohlížeč se pokusí obrázek stáhnout, nehledě na URL adresu v atributu `src`. Pokud je oběť v tu dobu přihlášena do své aplikace, na účet útočníka se připíše částka 100 000, aniž by oběť zjistila, že se transakce provedla.

Obrana proti útoku

Nejlepší obranou proti CSRF je vložení tzv. anti-CSRF tokenu jako skryté pole ve formuláři. Token musí být unikátní a nesmí být možné jej uhodnout. Hodnotu tokenu si server nějakým způsobem zapamatuje, buď uložením v relaci nebo cookie. Při odeslání formuláře se hodnoty porovnají a pokud se neshodují, s největší pravděpodobností se jedná o CSRF útok a akce se neprovede.

3.4.10 A9 – Použití známých zranitelných komponent

V dnešní době se většina webových aplikací skládá z mnoha komponent – více než 47 % všech webových stránek využívá nějaký redakční systém [43], který se mnohdy skládá s dalších doplňků a balíčků. Také existuje řada frameworků a knihoven usnadňující vývoj aplikací. Tyto komponenty také často závisí na dalších komponentách. Pokud se v některém s komponent objeví zranitelnost, útočník má šanci prolomit i jinak zabezpečenou aplikaci.

Při vývoji webové aplikace se mnohdy neklade dostatečný důraz na aktuálnost komponent. Škála možných zranitelností je široká – slabina v komponentě může umožnit injektování, XSS a nesprávné řízení přístupu [36, s. 15].

Obrana proti útoku

Vývojáři webových aplikací by měli používat nejnovější verze všech komponent. Staré verze mohou obsahovat známé bezpečnostní zranitelnosti, kterých může využít útočník. Je nutné sledovat jak základní komponenty aplikace, tak jejich závislosti [36, s. 15].

Důležité je také monitorování vývoje jednotlivých komponent i během provozu aplikace – jakmile se objeví v nějaké komponentě bezpečnostní slabina, je třeba zjistit, zda již k ní existuje záplata nebo provést výměnu za jinou, pokud to aplikace dovoluje.

3.4.11 A10 – Neošetřené přesměrování a předávání

Ve webových aplikacích se často využívá přesměrování na jinou stránku aplikace, např. po přihlášení uživatele do systému. Aplikace může URL adresu přesměrování

přijímat přímo z HTTP požadavku, který může obsahovat nedůvěryhodná data. Útočník toho může využít tak, že do požadavku odešle odkaz na podvodnou stránku, přes kterou se pokusí odcizit údaje od oběti. Na adresu útočníka se může dostat i celý požadavek pocházející z aplikace, který může obsahovat důvěrné informace o uživateli [36, s. 16].

Příklad zranitelnosti

Webová aplikace přesměrovává požadavek na jinou stránku, kterou lze specifikovat v parametru URL:

```
header('Location: ' . $_GET['redirectUrl']);
```

Za hodnotu parametru `redirectUrl` může útočník dosadit libovolnou URL adresu a přesměrovat uživatele na podvodný web:

```
http://webapp.net/success?redirectUrl=http://attackersite.com
```

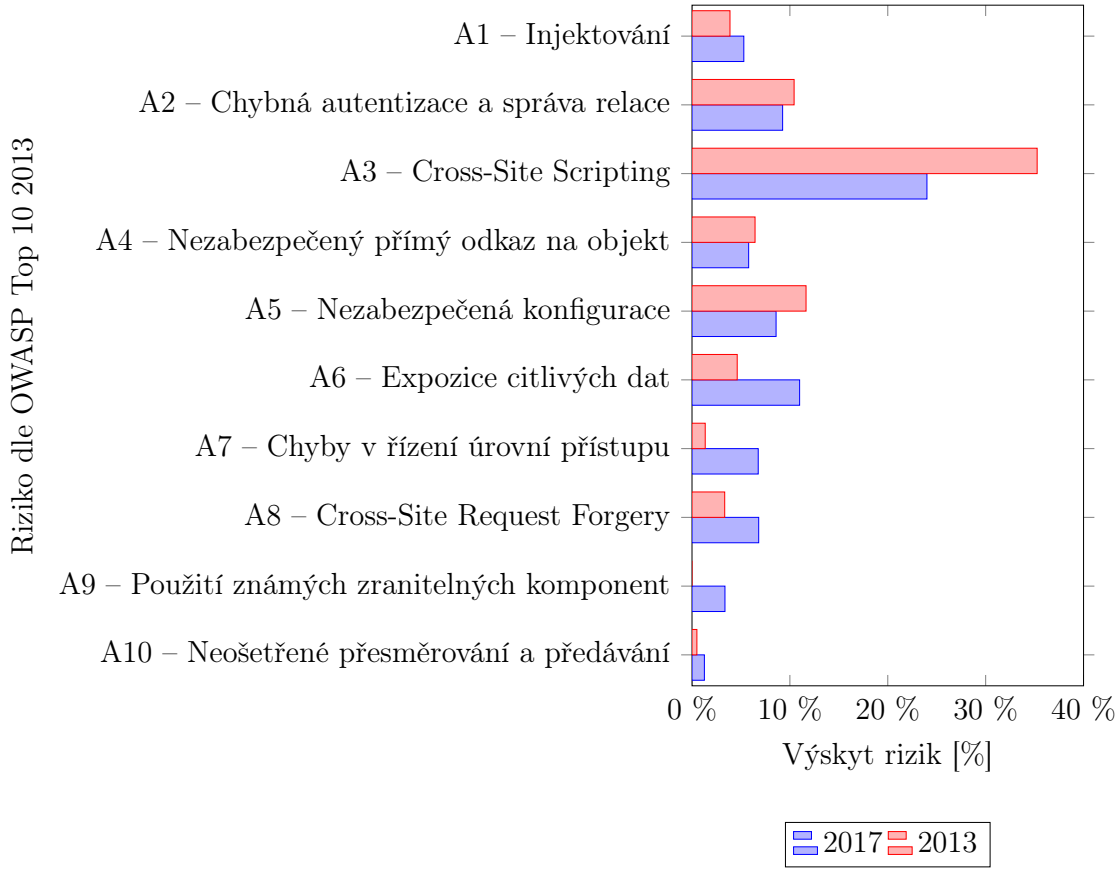
Obrana proti útoku

Nejjednodušší obranou je úplně se vyhnout přesměrovávání nebo předávání požadavků v aplikaci. To ale ve většině případů není možné, proto se doporučuje nepoužívat uživatelský vstup jako cíl přesměrování. Je-li nutné tyto parametry použít, je třeba se ujistit, že je hodnota platná [36, s. 16]. Další možností je v parametru předávat pouze mapovací hodnotu, která ve vnitřní logice odpovídá reálné cílové URL adrese.

3.5 Top 10 2017 Release Candidate

V květnu roku 2016 OWASP zveřejnil formulář, ve kterém měli organizace možnost odeslat statistické údaje o zranitelnostech webových aplikací [45]. Průzkumu se zúčastnilo celkem 10 organizací, které v součtu poskytly data o 50 895 webových aplikacích z let 2014 a 2015. 87 % všech dat pochází od společnosti Veracode [46].

Jak je znázorněno na obrázku 3.1, ve výskytu rizik OWASP Top 10 oproti průzkumu z roku 2013 nejvíce ztratilo riziko A3 – Cross-Site Scripting, jejíž relativní výskyt se v roce 2017 snížil o 11,26 procentních bodů. To jej stále řadí mezi nejrozšířenější rizika, nyní však je jeho podíl na celku méně než čtvrtinový. Naopak relativně nejvíce se rozšířilo riziko A6 – Expozice citlivých dat, které zaujímá necelých 11 % všech měřených rizik.



Obrázek 3.1: Srovnání výskytu rizik ze seznamu OWASP Top 10 2013 v průzkumech z let 2013 a 2017 (Zdroj: [46])

Na základě tohoto průzkumu byl 10. dubna 2017 vydán OWASP Top 10 – 2017 Release Candidate [44] a až do 30. června 2017 je možné se veřejně zapojit do diskuze o případných změnách v novém dokumentu. Navrhovaný seznam 10 nejkritičtějších rizik v aktualizovaném vydání vypadá takto [45]:

- A1 – Injektování
- A2 – Chybná autentizace a správa relace
- A3 – Cross-Site Scripting (XSS)
- **A4 – Chybné řízení přístupů**

- A5 – Nezabezpečená konfigurace
- A6 – Expozice citlivých dat
- **A7 – Nedostatečná ochrana proti útokům**
- A8 – Cross-Site Request Forgery (CSRF)
- A9 – Použití známých zranitelných komponent
- **A10 – Nedostatečně zabezpečené API**

Seznam obsahuje tři změny oproti Top 10 2013 (vyznačené tučně). OWASP se rozhodl spojit „Nezabezpečený přímý odkaz na objekt“ a „Chyby v řízení úrovní přístupu“ zpět na A4 – Chybné řízení přístupů poté, co byly v roce 2007 rozděleny.

Dále byly do seznamu přidány dvě nové položky. Na základě posledního průzkumu bylo zjištěno, že většina aplikací se nedokáže dostatečně bránit jak manuálním tak automatizovaným útokům, nebo neobsahuje mechanismy pro detekci a prevenci těchto útoků. Proto byla do seznamu zahrnuta položka A7 – Nedostatečná ochrana proti útokům.

Druhou přidanou položkou je A10 – Nedostatečně zabezpečené API. Důvodem přidání tohoto rizika je skutečnost, že moderní aplikace se často připojují k jiným službám přes externí API, které často obsahují řadu bezpečnostních zranitelností. Důsledkem zařazení tohoto rizika bylo odebrání položky A10 – Neošetřené přesměrování a předávání. Rizika spojená s neošetřeným přesměrováním již však nejsou tak rozšířená, a tak se mezi 10 nejrozšířenějšími riziky roku 2017 neobjeví [45, s. 5].

Kapitola 4

Penetrační testování webové aplikace

Tato kapitola se věnuje penetračnímu testování vybrané webové aplikace. Cílem této kapitoly je otestovat webovou aplikaci, vyhodnotit nalezené bezpečnostní zranitelnosti a opravit chyby. Výsledky testování jsou řazeny do kategorií podle dokumentu OWASP Top 10 2013.

První část kapitoly obsahuje představení samotné aplikace – popis, architektura a použité technologie. Ve druhé části je popsáno prostředí, ve kterém budou testy probíhat, z pohledu hardwarového a softwarového vybavení. Dále jsou vybrány vhodné testovací nástroje dle stanovených kritérií. Poslední dvě části této kapitoly jsou věnovány samotnému průběhu testování a vyhodnocení výsledků.

4.1 Představení webové aplikace

Testovaná aplikace funguje jako interní zakázkový systém pro společnost Atlas Advertising Group s.r.o., která se zabývá výrobou a montáží reklamních materiálů. Zaměstnanci společnosti mohou po přihlášení vytvářet kalkulace s produkty pro zákazníky, zakládat zakázky a zobrazovat přehledy s analytickými údaji o založených zakázkách. Údaje o zákaznících a fakturách jsou propojeny s účetním systémem. Rozsah funkcionality systému je závislý na přiřazené roli každého uživatele.

4.1.1 Architektura aplikace

Aplikace je vyvinuta v jazyce PHP s využitím frameworku Nette. Front end je tvořen v šablonovacím systému Latte (rovněž součást frameworku Nette). Aplikace také využívá AJAX pro dynamické zobrazování prvků na stránce. Pro správu dat aplikace využívá databázi MySQL.

4.2 Testovací prostředí

Veškeré testy jsou prováděny na lokálním počítači, na kterém běží i samotný webový server s aplikací. Tento způsob testování byl zvolen, protože z bezpečnostních

důvodů není možné provádět penetrační testy přímo na produkčním serveru. Popis testovacího prostředí je uveden v tabulce 4.1.

Hardware	
Procesor	Intel® Core™ i5-4690K – 3,5 GHz
RAM	DDR3 12 GB
SSD	Crucial MX100 – 256 GB
Software	
Operační systém	Windows 10 Home 64-bit
Webový prohlížeč	Google Chrome 57.0.2987.133
Webový server	UwAmp 3.1.0
	Apache 2.4.18
	PHP 7.0.3
	MySQL 5.7.11
Software pro testování	Arachni Framework v1.5.2 & WebUI v0.5.12

Tabulka 4.1: Hardwarové a softwarové vybavení v testovacím prostředí webové aplikace
(Zdroj: autor)

4.3 Použité nástroje

Hlavní kritéria výběru nástrojů pro testování webové aplikace byla následující:

- volná dostupnost
- aktuálnost
- možnost seskupení výsledků testů dle OWASP Top 10

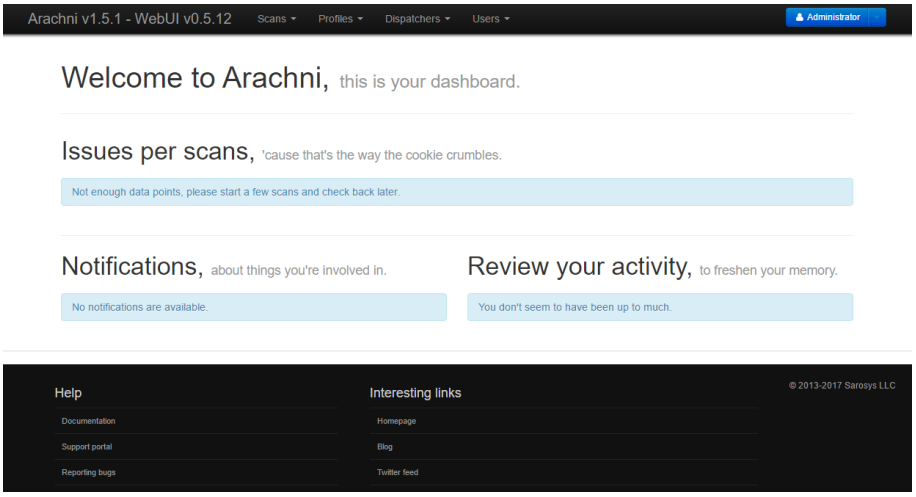
Pro automatizované penetrační testování byl použit nástroj Arachni. Arachni je multiplatformní open-source framework napsaný v jazyce Ruby zaměřený na penetrační testování moderních webových aplikací. Lze jej ovládat jak z příkazové řádky, tak přes webové rozhraní. K procházení webových stránek využívá integrované prostředí PhantomJS, které je schopno testovat zranitelnosti i v aplikacích využívajících JavaScript, AJAX nebo HTML5 [49]. Arachni byl vybrán kvůli jeho dostupnosti zdarma a možnosti generování podrobných reportů včetně OWASP Top 10. Tento software měl zároveň dobré výsledky ve srovnání nástrojů pro penetrační testování, zejména v úspěšnosti detekce zranitelností [48; 3].

4.4 Rozsah a povaha testů

Bezpečnost aplikace bude prověřována automatizovaným penetračním testováním pomocí vybraných nástrojů. Zároveň budou některé bezpečnostní aspekty testovány manuálně metodou white-box, neboť existuje možnost přímého přístupu ke zdrojovému kódu aplikace. Výsledky testů budou seskupeny výhradně dle OWASP Top 10 2013. Při hodnocení výsledků bude brán ohled na rozdíly oproti produkčnímu prostředí.

4.5 Příprava a konfigurace nástrojů

Po stažení nástroje Arachni z webu `http://www.arachni-scanner.com` je potřeba rozbalit instalační balíček. Pro jednodušší ovládání je zvoleno webové prostředí spuštěním dávkového souboru `arachni_web.bat` v adresáři `bin`. Tím se spustí webový server, který nabízí webové rozhraní na URL adrese `http://localhost:9292`. Webové rozhraní je chráněno přístupovými údaji (e-mailová adresa a heslo). Výchozí hodnoty pro přihlášení jsou vypsány v souboru `README.txt`. Pro testování je použit výchozí administrátorský účet s údaji „admin@admin.admin“ a „administrator“. Po přihlášení se zobrazí úvodní stránka – viz obrázek 4.1.



Obrázek 4.1: Úvodní stránka webového rozhraní frameworku Arachni po přihlášení (Zdroj: autor)

4.5.1 Profil skenování

Skeny, které v Arachni lze spouštět, vždy vycházejí z přednastaveného profilu. Výchozí profil skenování je nastaven na „Default“. Pro účely testování je založen nový profil, který umožní konfigurovat testy přímo na danou aplikaci a zefektivnit tak průběh testů. Nový profil vychází z profilu „Default“ – na stránce detailu

byla zvolena možnost vytvoření nového profilu na základě zvoleného profilu pomocí modrého tlačítka pro kopírování.

Dále bude popsána konfigurace vlastního profilu – pro zjednodušení budou zmíněna pouze ta nastavení, která se od výchozího profilu liší. Členění popisu konfigurace kopíruje členění na stránce „Create a profile“.

Scope

V této části je vyplněna položka „Scope auto redundant paths“ na hodnotu „2“. Důvodem je zjištění, že testovaná aplikace obsahuje velké množství nastavení pomocí URL parametrů, které však závažně nemění podstatu zobrazovaného obsahu. Nejčastěji se jedná o nastavení počtu zobrazovaných záznamů na stránku. Nízká hodnota by tedy měla zajistit efektivnější procházení stránek s těmito parametry.

HTTP

Zde je potřeba vyplnit platné cookies do pole „Http cookies“, neboť celá aplikace je přístupná pouze po přihlášení. Cookies se vyplňují ve tvaru `název=hodnota`, každá cookie na samostatný řádek. Jelikož aplikace požaduje dodatečné ověření přihlášení zadáním číselného kódu, je potřeba do pole zadat i cookie, která představuje „zapamatované“ zařízení.

Platform fingerprinting

Tato část slouží k zefektivnění procházení označením pouze těch komponent, které aplikace skutečně využívá. Tabulka 4.2 obsahuje označené položky v každém sloupci této části.

Sloupec	Vybrané položky
Operating systems	MS Windows
Databases	MySQL
Web servers	Apache
Programming languages	PHP
Frameworks	Nette

Tabulka 4.2: Vybrané položky v sekci Platform fingerprinting v konfiguraci profilu skenování v Arachni
(Zdroj: autor)

Checks

V této části jsou *vynechány* následující kontroly:

- Active
  - LDAPInjection
  - NoSQL Injection
  - Blind NoSQL Injection
- Passive
  - NoSQL Injection (captcha)
  - LDAPInjection (directory_listing)
  - WebDAV

4.5.2 Spuštění testů

Testy se spouští z nabídky „Scans“ vybráním položky „New“. Zobrazí se stránka „Start a scan“, kde je potřeba vyplnit cílovou URL adresu, která je v případě testované aplikace `http://localhost`. Dále je nutné vybrat nově vytvořený profil. Poté stačí test spustit stisknutím tlačítka „Go!“.

4.6 Průběh testů a vyhodnocení

Jako nevýhoda Arachni se ukázala velká časová náročnost testů. Test byl ukončen po 6 hodinách a bylo nalezeno celkem 54 problémů. Dlouhý čas skenování je přisuzován jak povaze nástroje, tak delší průměrnou dobou odezvy webového serveru aplikace, která se pohybovala kolem 3 vteřin. Tabulka 4.3 shrnuje výsledky testů dle závažnosti problémů.

Druh problému	Počet výskytů	OWASP Top 10
Cross-Site Request Forgery	29	A6, A8
HTTP TRACE	1	A5
Nezašifrovaný formulář s heslem	1	A6
Formulář s automatickým vyplňování hesla	1	—
Zajímavá odpověď od serveru	21	—
Nahrávání souboru přes formulář	1	—

Závažnost problému: vysoká střední nízká informativní

Tabulka 4.3: Shrnutí výsledků testů webové aplikace v nástroji Arachni (Zdroj: autor)

Následuje vyhodnocení výsledků testů podle OWASP Top 10 2013. Ne všechny druhy rizik je možné dostatečně otestovat pomocí automatizovaných nástrojů – nástroj Arachni není spolehlivě schopen detekovat tyto kategorie:

- A4 – Nezabezpečený přímý odkaz na objekt
- A7 – Chyby v řízení úrovní přístupů
- A9 – Použití známých zranitelných komponent

U těchto rizik byla provedena manuální inspekce zdrojového kódu.

4.6.1 A1 – Injektování

Automatizovaný test nenalezl žádná rizika spojená s injektováním kódu. Pro komunikaci s SQL databází aplikace používá knihovnu `Nette\Database`, jež tvoří nadstavbu nad PHP Data Objects (PDO).

Jako obrana proti SQL injektování jsou použity speciální funkce této knihovny, které skládají SQL dotazy bezpečným způsobem. Výběr údajů o zakázce se zadaným ID vypadá takto:

```
$commissionRow = $this->database->table('commission')->get($id);
```

4.6.2 A2 – Chybná autentizace a správa relace

Test nenalezl žádné problémy s autentizací nebo správou relací uživatele. Identifikátory relací jsou ukládány do cookies klienta a údaje o relacích jsou uloženy na straně serveru. Nette framework navíc ignoruje identifikátory relací v URL, pokud se v ní vyskytnou [50].

4.6.3 A3 – Cross-Site Scripting (XSS)

V aplikaci nebyly nalezeny žádné případy XSS. Framework Nette zajišťuje dostatečnou ocharnu proti XSS pomocí escapování problematických znaků v závislosti na kontextu (Context-Aware Escaping) [53]. Pokud tedy např. uživatel zadá kód v JavaScriptu do formulářového pole a formulář uloží, kód se nespustí a zobrazí se pouze jako prostý text na stránce – viz obrázek 4.2.

Obrázek 4.2: Příklad ochrany proti XSS ve webové aplikaci
(Zdroj: autor)

4.6.4 A4 – Nezabezpečený přímý odkaz na objekt

Tento druh rizika nebylo možné otestovat automatizovaným nástrojem, manuální kontrola kódu však odhalila možné riziko. Aplikace umožňuje uživatelům systému nahrávat přílohy k zakázkám a na svůj profil. Na tyto přílohy vedou přímé odkazy. Přílohy je tak možné načíst i bez přihlášení do systému, ale pouze za předpokladu, že URL adresa přílohy je předem známá.

Řešení problému

Bezpečným řešením je umístit složku s přílohami mimo veřejně dostupný adresář a pro zobrazení příloh použít aplikační funkci, která načte přílohu na stránku ze „skrytého“ adresáře. Kontrola autorizace v této funkci zajistí nutnost přihlášení uživatele při zobrazení přílohy. K přílohám se místo jejich názvu v URL přistupuje pomocí jednoznačného číselného identifikátoru.

4.6.5 A5 – Nezabezpečená konfigurace

Automatizovaný test nalezl v této kategorii existenci HTTP metody TRACE. Využitím této metody lze pomocí útoku Cross-Site Tracing (XST) odcizit cookies uživatele [51]. Nicméně Apache tuto možnost útoku zpochybňuje [52]:

> Despite claims to the contrary, enabling the TRACE method does not expose any security vulnerability in Apache httpd. The TRACE method is defined by the HTTP/1.1 specification and implementations are expected to support it.

Řešení problému

HTTP metodu TRACE lze deaktivovat v konfiguračním souboru Apache `httpd.conf` přidáním řádku `TraceEnable off` (výchozí hodnota v konfiguraci je nastavena na „on“ – zapnuto).

4.6.6 A6 – Expozice citlivých dat

Program Arachni zařadil do této kategorie všechny nálezy s nedostatečnou ochranou proti útokům typu Cross-Site Request Forgery. Více o CSRF v části 4.6.8. Dále se do této kategorie řadí absence šifrovaného spojení při odesílání přihlašovacího formuláře. Aplikace nevyužívá pro spojení protokol HTTPS, a tak je možné přenášená data odposlouchávat.

Návrh řešení problému

S ohledem na závažnost rizika a možné dopady je vhodné zvážit podporu šifrovaného připojení přes HTTPS.

4.6.7 A7 – Chyby v řízení úrovní přístupů

Automatizovaný test nebyl schopen odhalit chyby v této kategorii. Aplikace využívá pro řízení přístupů Access Control List (ACL) knihovnu `Nette\Security\Permission` poskytovanou frameworkem Nette. Inspekcí zdrojového kódu bylo zjištěno, že aplikace implementuje kontroly oprávnění i na úrovni aplikační logiky. Například metoda zpracovávající stránku s editací zakázky neumožní upravit zakázku uživateli, který nemá přidělená příslušná oprávnění, a místo toho uživatele přesměruje na přehled zakázek:

```
public function actionHome($id, $productId) {
  if (!$this->user->isAllowed('commission-module', 'edit')) {
    $this->flashMessage('Nemáte oprávnění pro úpravu zakázky.', 'info');
    $this->redirect('Overview:home');
  }
  // pokračování logiky...
}
```

4.6.8 A8 – Cross-Site Request Forgery (CSRF)

Ačkoliv program nalezl celkem 29 výskytů zranitelností vůči CSRF, mnoho nalezených stránek jsou ve skutečnosti identické formuláře, pouze s odlišnými vstupními daty (např. stránky s detailem zakázky). Sloučením těchto duplicit se počet snížil na 5 různých míst v aplikaci. Manuální kontrolou kódu bylo však zjištěno, že ochrana proti CSRF chybí u všech formulářů.

Řešení problému

Nette framework nabízí jednoduché řešení ochrany proti CSRF – ke každému formuláři je třeba připojit volání funkce, která k němu automaticky doplní kontrolu anti-CSRF tokenu [53]. Příklad opravy problému ve formuláři pro editaci zakázky (3. řádek ve zdrojovém kódu):

```
protected function createComponentCommissionForm() {
  $form = new Form($this, 'commissionForm');
  $form->addProtection();
}
```

4.6.9 A9 – Použití známých zranitelných komponent

Tento druh rizika nebylo možné spolehlivě otestovat pomocí automatizovaného nástroje, nicméně bylo zjištěno, že aplikace používá nejnovější verzi frameworku Nette, tedy 2.4. Všechny ostatní používané knihovny jsou rovněž aktuální.

4.6.10 A10 – Neošetřené přesměrování a předávání

Automatizovaný test nenašel žádná rizika týkající se neošetřených přesměrování nebo předávání. Po kontrole kódu bylo zjištěno, že aplikace v přesměrováních nepoužívá uživatelské vstupy, a tak tento druh útoku není proveditelný.

4.6.11 Ostatní problémy

Test nalezl jeden další problém, který nebyl přímo zařazen do OWASP Top 10. Jedná se o přihlašovací formulář, u kterého je povoleno automatické vyplňování hesla. Tato funkce umožňuje uživatelům pohodlnější přihlašování, neboť jsou údaje načteny z cache webového prohlížeče. Pokud je však automatické vyplňování povoleno, potenciální útočník, který získá přístup k počítači uživatele, tím získá přístup i k dané aplikaci bez nutnosti zadání e-mailu a hesla.

Řešení problému

Automatické vyplňování formuláře je možné vypnout přidáním parametru `autocomplete="off"` do tagu `<form>`:

```
<form method="post" action="/form" autocomplete="off">
```

Nicméně většina nových webových prohlížečů toto nastavení ignoruje. Důvodem je bezpečnostní přínos funkce automatického vyplňování, který umožňuje vybírat bezpečnější hesla, neboť si je uživatel nemusí pamatovat [54].

Kapitola 5

Právní aspekty

5.1 Ochrana osobních údajů

Webové aplikace často nějakým způsobem zpracovávají osobní údaje. Může se jednat např. o vyplňování webového formuláře při registraci nového uživatele nebo objednávku zboží v e-shopu. Tyto údaje jsou aplikací zpracovávány a ukládány do úložiště (databáze), aby se s nimi mohlo dále pracovat. Nedostatečně zabezpečená aplikace tak může dovolit útočníkovi, aby tyto údaje získal a případně zneužil.

Práva a povinnosti při zpracování osobních údajů řeší Zákon č. 101/2000 Sb., o ochraně osobních údajů [55]. Zákon mimo jiné stanovuje tuto povinnost při zabezpečení osobních údajů v § 13:

> Správce a zpracovatel jsou povinni přijmout taková opatření, aby nemohlo dojít k neoprávněnému nebo nahodilému přístupu k osobním údajům, k jejich změně, zničení či ztrátě, neoprávněným přenosům, k jejich jinému neoprávněnému zpracování, jakož i k jinému zneužití osobních údajů. Tato povinnost platí i po ukončení zpracování osobních údajů.

5.2 Penetrační testování

Přestože cílem penetračního testování není získání soukromých informací ze systému, je třeba předem získat souhlas k provedení testů od kompetentních osob testované aplikace. Testování aplikace bez souhlasu druhé strany se dá považovat za protiprávní jednání. V současné době sankce při tomto jednání upravuje Zákon č. 40/2009 Sb., trestní zákoník v těchto paragrafech [56, s. 406–407]:

- § 230 Neoprávněný přístup k počítačovému systému a nosiči informací
- § 231 Opatření a přechovávání přístupového zařízení a hesla k počítačovému systému a jiných takových dat
- § 232 Poškození záznamu v počítačovém systému a na nosiči informací a zásah do vybavení počítače z nedbalosti

Kapitola 6

Závěr

Cílem této práce bylo otestovat bezpečnost vybrané webové aplikace s využitím bezpečnostní platformy OWASP, vyhodnotit výsledky testování a navrhnout řešení nalezených zranitelností.

Teoretická část se zabývala bezpečností webových aplikací obecně s ohledem na vývoj technologií. Dále se tato část věnovala platformě OWASP a především projektu OWASP Top Ten. Byl zde popsán způsob hodnocení bezpečnostních rizik a bylo poukázáno na změny v aktuálním dokumentu OWASP Top 10 oproti předchozí verzi. Následně bylo představeno všech 10 nejkritičtějších rizik, včetně příkladů zranitelností a způsobů, jak se proti útokům bránit. Tato část sloužila jako základ pro správnou interpretaci výsledků bezpečnostních testů webové aplikace.

Cílem praktické části bylo otestování vybrané aplikace a vyhodnocení výsledků s využitím znalostí o OWASP Top 10. Penetrační testy odhalily několik zranitelností, které byly následně opraveny, nebo byly navrženy možnosti, jak je opravit. I přes nalezené zranitelnosti byla většina častých bezpečnostních problémů v aplikaci již ošetřena, a to především díky tomu, že je aplikace postavena na moderním frameworku a využívá pomocných knihoven, jež pomáhají chránit aplikaci před známými druhy útoků.

Dalším krokem v ověřování bezpečnosti webové aplikace, který tato práce již nezahrnuje, by mohlo být zařazení bezpečnostních testů do celého životního cyklu vývoje aplikace. Touto problematikou se hlouběji věnuje bakalářská práce Bc. Ondřeje Bukovského [48]. Pravidelné testy by pomohly odhalit bezpečnostní zranitelnosti rychleji a efektivně.

Seznam zkratek

ACL Access Control List 36

AJAX Asynchronous JavaScript and XML 28, 29

API Application Programming Interface 27

ASP Active Server Pages 5–7

CRUD Create, Read, Update, Delete 17

CSS Cascading Style Sheets 4, 20

GUI Graphical User Interface 23

HTML Hypertext Markup Language 4–6, 19, 20, 29

HTTP Hypertext Transfer Protocol 3, 12, 16, 18, 25, 33, 35

LDAP Lightweight Directory Access Protocol 17

OWASP Open Web Application Security Project 1, 2, 4, 5, 7, 11, 13–17, 25–30, 33, 37, 40, 50–52

PBKDF2 Password-Based Key Derivation Function 2 22

PDO PHP Data Objects 33

PHP PHP: Hypertext Preprocessor 2, 5–8, 18, 28, 29, 50

SaaS Software as a Service 16

SHA Secure Hash Algorithm 22

SQL Structured Query Language 17, 18, 33, 34

SSL Secure Socket Layer 22

URL Uniform Resource Locator 5, 12, 18–21, 23–25, 30–32, 34, 35

Terminologický slovník

bezpečnostní hrozba „Potenciální příčina nežádoucí události, která může mít za následek poškození systému a jeho aktiv, např. zničení, nežádoucí zpřístupnění (kompromitaci), modifikaci dat nebo nedostupnost služeb“ [7, s. 25] 4

bezpečnostní riziko „Souhrn možností, že hrozba využije zranitelnost aktiva nebo skupiny aktiv a tím způsobí organizaci škodu.“ [7, s. 99] 4, 14, 21

bezpečnostní zranitelnost „Úmyslná chyba nebo neúmyslný nedostatek či závada v software obecně nebo ve firmware zařízení komunikační infrastruktury, která může být zneužita potenciálním útočníkem pro škodlivou činnost“ [7, s. 25] 1, 4, 9, 14, 21, 24, 27, 28

CWE (Common Weakness Enumeration) Komunitní projekt spravující seznam běžných typů bezpečnostních zranitelností softwaru [57] 8, 50

DOM (Document Object Model) Rozhraní umožňující programům a skriptům dynamicky měnit obsah, strukturu a styl dokumentu [58] 6, 20

escapování Využití zástupných znaků s cílem alternativní interpretace následujících znaků v řetězci [59] 18–20, 34

framework Forma znovupoužitelného softwaru obsahující ucelené architektury optimalizované pro použití v konkrétních aplikacích [60, s. 29] . 17, 19, 21, 24, 28, 29, 34, 36, 40

front end Prezentační část aplikace, kterou využívá uživatel pro interakci s webovými stránkami prostřednictvím grafického uživatelského rozhraní [61] .. 3, 23, 28

Seznam použité literatury

[1] NETCRAFT. March 2017 Web Server Survey. *Netcraft | Internet Research, Anti-Phishing and PCI Security Services* [online]. 2017 [cit. 2017-03-27]. Dostupné z: `https://news.netcraft.com/archives/2017/03/24/march-2017-web-server-survey.html`

[2] HANZLIKOVÁ, Martina. *Testování bezpečnosti webových aplikací* [online]. Praha, 2013 [cit. 2017-03-24]. Dostupné z: `https://insis.vse.cz/zp/portal_zp.pl?prehled=vyhledavani;podrobnosti_zp=42003;zp=42003;download_prace=1`. Bakalářská práce. Fakulta informatiky a statistiky Vysoké školy ekonomické v Praze, Katedra informačních technologií. Vedoucí práce Ing. Jan Mittner.

[3] KIEZLER, Tomáš. *Vývoj nástroje na automatizované penetrační testování webových aplikací* [online]. Praha, 2016 [cit. 2017-03-23]. Dostupné z: `https://insis.vse.cz/zp/portal_zp.pl?prehled=vyhledavani;podrobnosti_zp=55987;zp=55987;download_prace=1`. Bakalářská práce. Fakulta informatiky a statistiky Vysoké školy ekonomické v Praze, Katedra informačního a znalostního inženýrství. Vedoucí práce Ing. Jiří Hradil.

[4] NATIONS, Daniel. Understanding the Purpose of Web Applications: What Is a Web Application? *Lifewire* [online]. 2016 [cit. 2017-04-05]. Dostupné z: `https://www.lifewire.com/what-is-a-web-application-3486637`

[5] BITTNER, Jan. *Vývoj webové aplikace* [online]. Česká Lípa, 2016 [cit. 2017-04-06]. Dostupné z: `https://github.com/HoBi/rocnikova-prace/blob/master/rocnikova-prace.pdf`. Ročníková práce. Střední průmyslová škola Česká Lípa. Vedoucí práce Bc. Jan Fojtík.

[6] FIELDING, Roy T et al. Hypertext Transfer Protocol – HTTP/1.1. *The Internet Engineering Task Force (IETF)* [online]. 1999 [cit. 2017-04-06]. Dostupné z: `https://tools.ietf.org/html/rfc2616`

[7] JIRÁSEK, Petr, Luděk NOVÁK a Josef POŽÁR. *Výkladový slovník kybernetické bezpečnosti* [online]. 3. aktualizované vyd. Praha: Policejní akademie ČR v Praze, 2015 [cit. 2017-04-06]. ISBN 978-80-7251-436-6. Dostupné z: `https://www.govcert.cz/download/aktuality/container-nodeid-665/slovnikkb-cz-en-1505.pdf`

[8] Zákon č. 181/2014, o kybernetické bezpečnosti a o změně souvisejících zákonů (zákon o kybernetické bezpečnosti). In: *Sbírka zákonů České republiky*. Česká republika, 2014, částka 75. ISSN 1211-1244.

[9] TURNER, Dawn M. Digital Authentication - the basics. *Security Solutions - Cryphomathic* [online]. 2016 [cit. 2017-04-06]. Dostupné z: `https://www.cryptomathic.com/news-events/blog/digital-authentication-the-basics`

[10] MEIER, J. D. et al. Chapter 1- Web Application Security Fundamentals. MICROSOFT CORPORATION. *Learn to Develop with Microsoft Developer Network / MSDN* [online]. 2003 [cit. 2017-04-06]. Dostupné z: `https://msdn.microsoft.com/en-us/library/ff648636.aspx`

[11] CONNOLLY, Dan. A Little History of the World Wide Web. *World Wide Web Consortium (W3C)* [online]. 2000 [cit. 2017-03-26]. Dostupné z: `https://www.w3.org/History.html`

[12] THE PHP GROUP. History of PHP. *PHP: Hypertext Preprocessor* [online]. 2017 [cit. 2017-03-26]. Dostupné z: `http://php.net/manual/en/history.php.php`

[13] W3TECHS. Technologies Overview. *W3Techs - extensive and reliable web technology surveys* [online]. 2017 [cit. 2017-04-05]. Dostupné z: `https://w3techs.com/technologies`

[14] W3TECHS. Usage of server-side programming languages for websites. *W3Techs - extensive and reliable web technology surveys* [online]. 2017 [cit. 2017-04-05]. Dostupné z: `https://w3techs.com/technologies/overview/programming_language/all`

[15] WEB EDUCATION COMMUNITY GROUP. A Short History of JavaScript. *Web Education Community Group* [online]. 2012 [cit. 2017-03-26]. Dostupné z: `https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript`

[16] OWASP FOUNDATION. Top 10 2013-A3-Cross-Site Scripting (XSS). *OWASP* [online]. 2013 [cit. 2017-04-05]. Dostupné z: `https://www.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_(XSS)`

[17] THE PHP GROUP. PHP Tags. *PHP: Hypertext Preprocessor* [online]. 2017 [cit. 2017-04-05]. Dostupné z: `https://secure.php.net/manual/en/language.basic-syntax.phptags.php`

[18] THE PHP GROUP. Echo. *PHP: Hypertext Preprocessor* [online]. 2017 [cit. 2017-04-05]. Dostupné z: http://php.net/manual/en/faq.general.php

[19] NODE.JS FOUNDATION. About Node.js®. *Node.js* [online]. 2017 [cit. 2017-04-05]. Dostupné z: https://nodejs.org/en/about/

[20] VERACODE. Contact Us. *Application Security | Veracode* [online]. [2017] [cit. 2017-04-08]. Dostupné z: https://info.veracode.com/web-contact-us.html

[21] VERACODE. State of Software Security Reports. *Application Security | Veracode* [online]. 2016 [cit. 2017-04-08]. Dostupné z: https://www.veracode.com/resources/state-of-software-security

[22] VERACODE. State of Software Security: Focus on Application Development. *Application Security | Veracode* [online]. 2015 [cit. 2017-04-08]. Dostupné z: https://www.veracode.com/sites/default/files/Resources/Reports/state-of-software-security-focus-on-application-development.pdf

[23] KRUTZ, Ronald L. a Russel Dean VINES. *The CISSP® and CAPCM Prep Guide: Platinum Edition*. Indianapolis (Indiana): Wiley Publishing, 2007. ISBN 978-0-470-00792-1.

[24] SELECKÝ, Matúš. *Penetrační testy a exploitace*. Brno: Computer Press, 2012. ISBN 978-80-251-3752-9.

[25] OWASP FOUNDATION. *OWASP Testing Guide v4* [online]. 2015 [cit. 2017-03-27]. Dostupné z: https://www.owasp.org/images/1/19/OTGv4.pdf

[26] OWASP FOUNDATION. Penetration testing methodologies. *OWASP* [online]. 2016 [cit. 2017-04-08]. Dostupné z: https://www.owasp.org/index.php/Penetration_testing_methodologies

[27] OWASP FOUNDATION. *OWASP* [online]. 2017 [cit. 2017-03-24]. Dostupné z: https://www.owasp.org/index.php/Main_Page

[28] OWASP FOUNDATION. OWASP Project Inventory: Flagship Projects. *OWASP* [online]. 2016 [cit. 2017-04-09]. Dostupné z: https://www.owasp.org/index.php/OWASP_Project_Inventory#Flagship_Projects

[29] DENNIS, Michael Aaron. Wiki: Web site. *Encyclopedia Britannica | Britannica.com* [online]. 2008 [cit. 2017-04-09]. Dostupné z: https://www.britannica.com/topic/wiki

[30] OWASP FOUNDATION. Category:OWASP Top Ten Project: Project Details. *OWASP* [online]. 2017 [cit. 2017-04-11]. Dostupné z: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=Project_Details

[31] THE OPEN GROUP. *Risk Taxonomy: Technical Standard* [online]. 2009 [cit. 2017-04-12]. ISBN 1-931624-77-1. Dostupné z: http://pubs.opengroup.org/onlinepubs/9699919899/toc.pdf

[32] ROUSE, Margaret. What is attack vector? - Definition from WhatIs.com. In: *Fuel Your Pipeline Faster - TechTarget* [online]. 2012 [cit. 2017-04-12]. Dostupné z: http://searchsecurity.techtarget.com/definition/attack-vector

[33] OWASP FOUNDATION. Category:Technical Impact. *OWASP* [online]. 2008 [cit. 2017-04-12]. Dostupné z: https://www.owasp.org/index.php/Category:Technical_Impact

[34] OWASP FOUNDATION. Category:Business Impact. *OWASP* [online]. 2008 [cit. 2017-04-12]. Dostupné z: https://www.owasp.org/index.php/Category:Business_Impact

[35] OWASP FOUNDATION. OWASP Risk Rating Methodology. *OWASP* [online]. 2016 [cit. 2017-04-13]. Dostupné z: https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

[36] OWASP FOUNDATION. *OWASP Top 10 - 2013* [online]. 2013 [cit. 2017-03-27]. Dostupné z: https://www.owasp.org/images/f/f8/OWASP_Top_10_-_2013.pdf

[37] OWASP FOUNDATION. A1 2004 Unvalidated Input. *OWASP* [online]. 2010 [cit. 2017-04-14]. Dostupné z: https://www.owasp.org/index.php/A1_2004_Unvalidated_Input

[38] OWASP FOUNDATION. OWASP Top 10 2007. *OWASP* [online]. 2007 [cit. 2017-04-14]. Dostupné z: https://www.owasp.org/images/e/e8/OWASP_Top_10_2007.pdf

[39] OWASP FOUNDATION. Injection Flaws. *OWASP* [online]. 2015 [cit. 2017-04-15]. Dostupné z: https://www.owasp.org/index.php/Injection_Flaws

[40] OWASP FOUNDATION. Session Management Cheat Sheet. *OWASP* [online]. 2016 [cit. 2017-04-16]. Dostupné z: https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

[41] OWASP FOUNDATION. Cross-site Scripting (XSS). *OWASP* [online]. 2016 [cit. 2017-04-16]. Dostupné z: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

[42] STEVENS, Marc et al. Announcing the first SHA1 collision. GOOGLE. *Google Online Security Blog* [online]. 2017 [cit. 2017-04-16]. Dostupné z: https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html

[43] W3TECHS. Usage of content management systems for websites. *W3Techs - extensive and reliable web technology surveys* [online]. 2017 [cit. 2017-04-17]. Dostupné z: https://w3techs.com/technologies/overview/content_management/all

[44] OWASP FOUNDATION. Category:OWASP Top Ten Project: OWASP Top 10 for 2017 Release Candidate. *OWASP* [online]. 2017 [cit. 2017-04-17]. Dostupné z: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2017_Release_Candidate

[45] OWASP FOUNDATION. OWASP Top 10 - 2017 rc1. GITHUB. *GitHub* [online]. 2017 [cit. 2017-04-17]. Dostupné z: https://github.com/OWASP/Top10/raw/master/2017/OWASP%20Top%2010%20-%202017%20RC1-English.pdf

[46] WICHERS, Dave. OWASP Top 10 - 2017 Data Call-Public Release. GITHUB. *GitHub* [online]. 2016 [cit. 2017-04-17]. Dostupné z: https://github.com/OWASP/Top10/blob/master/2017/datacall/OWASP%20Top%2010%20-%202017%20Data%20Call-Public%20Release.xlsx?raw=true

[47] *Atlas AG* [online]. [cit. 2017-04-17]. Dostupné z: http://intranet.atlas-ag.cz

[48] BUKOVSKÝ, Ondřej. *Testování bezpečnosti webových aplikací* [online]. Praha, 2012 [cit. 2017-04-16]. Dostupné z: https://insis.vse.cz/zp/portal_zp.pl?prehled=vyhledavani;podrobnosti_zp=35512;zp=35512;download_prace=1. Bakalářská práce. Fakulta informatiky a statistiky Vysoké školy ekonomické v Praze, Katedra informačních technologií. Vedoucí práce Ing. Luboš Pavlíček.

[49] SAROSYS LLC. *Home - Arachni - Web Application Security Scanner Framework* [online]. 2017 [cit. 2017-04-16]. Dostupné z: http://www.arachni-scanner.com

[50] NETTE FOUNDATION. Sessions. *Rychlý a pohodlný vývoj webových aplikací v PHP | Nette Framework* [online]. 2017 [cit. 2017-04-18]. Dostupné z: https://doc.nette.org/cs/2.4/sessions

[51] OWASP FOUNDATION. Cross Site Tracing. *OWASP* [online]. 2016 [cit. 2017-04-18]. Dostupné z: https://www.owasp.org/index.php/Cross_Site_Tracing

[52] THE APACHE SOFTWARE FOUNDATION. Apache Core Features: TraceEnable Directive. *Welcome to The Apache Software Foundation!* [online]. 2017 [cit. 2017-04-19]. Dostupné z: https://httpd.apache.org/docs/2.4/mod/core.html#traceenable

[53] NETTE FOUNDATION. Zabezpečení před zranitelnostmi. *Rychlý a pohodlný vývoj webových aplikací v PHP | Nette Framework* [online]. 2017 [cit. 2017-04-18]. Dostupné z: https://doc.nette.org/cs/2.4/vulnerability-protection

[54] DEAUDELIN, Mathieu et al. How to Turn Off Form Autocompletion. MOZILLA DEVELOPER NETWORK AND INDIVIDUAL CONTRIBUTORS. *Mozilla Developer Network* [online]. 2016 [cit. 2017-04-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Security/Securing_your_site/Turning_off_form_autocompletion

[55] Zákon č. 101/2000 Sb., o ochraně osobních údajů a změně některých zákonů. In: *Sbírka zákonů České republiky*. 2000, částka 32. ISSN 1211-1244.

[56] Zákon č. 40/2009 Sb., trestní zákoník. In: *Sbírka zákonů České republiky*. 2009, částka 11. ISSN 1211-1244.

[57] THE MITRE CORPORATION. CWE Overview. *CWE - Common Weakness Enumeration* [online]. 2017 [cit. 2017-04-08]. Dostupné z: https://cwe.mitre.org/about/index.html

[58] W3C®. Document Object Model (DOM). *World Wide Web Consortium (W3C)* [online]. 2005 [cit. 2017-04-06]. Dostupné z: https://www.w3.org/DOM/

[59] THE PHP GROUP. Escape sequences. *PHP: Hypertext Preprocessor* [online]. 2017 [cit. 2017-04-19]. Dostupné z: https://secure.php.net/manual/en/regexp.reference.escape.php

[60] PASETTI, Alessandro. *Software Frameworks and Embedded Control Systems*. Berlin: Springer, 2002. ISBN 3-540-43189-6.

[61] OXFORD UNIVERSITY PRESS. Definition of front end in English. *Oxford Dictionaries - Dictionary, Thesaurus, & Grammar* [online]. 2017 [cit. 2017-04-19]. Dostupné z: https://en.oxforddictionaries.com/definition/front_end

Seznam obrázků

1.1 Zastoupení serverových programovacích jazyků na webu 5
1.2 Výsledky testů OWASP Top 10 dle programovacích jazyků 7
1.3 Zastoupení druhů zranitelností dle CWE ve webových aplikacích v PHP . 8

3.1 Srovnání výskytu rizik ze seznamu OWASP Top 10 2013 v průzkumech z let 2013 a 2017 26

4.1 Úvodní stránka webového rozhraní frameworku Arachni po přihlášení . . 30
4.2 Příklad ochrany proti XSS ve webové aplikaci 34

Seznam tabulek

3.1 Faktory výpočtu hodnocení rizik OWASP Top 10 15
3.2 Srovnání rizik OWASP Top 10 2013 oproti verzi z roku 2010 17

4.1 Hardwarové a softwarové vybavení v testovacím prostředí webové aplikace 29
4.2 Vybrané položky v sekci Platform fingerprinting v konfiguraci profilu skenování v Arachni 31
4.3 Shrnutí výsledků testů webové aplikace v nástroji Arachni 33

Seznam rovnic

3.1 Obecný výpočet rizika dle OWASP Risk Rating Methodology 15
3.2 Výpočet rizika v OWASP Top 10 15