**Univerzita Hradec Králové**
**Fakulta informatiky a managementu**
**Katedra informatiky a kvantitativních metod**

# Nástroje pro penetrační testování webových aplikací a jejich praktické využití

Diplomová práce

Autor: Josef Čejka
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Pavel Kříž, Ph. D.

Hradec Králové duben 2017

# PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 23. dubna 2017 .....

Josef Čejka

## PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Pavlu Křížovi, Ph.D za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.


V Hradci Králové dne 23. dubna 2017 .....

Josef Čejka

## Anotace

Diplomová práce se zabývá problematikou bezpečnostních testů webových aplikací a stránek. Cílem práce je seznámit čtenáře s bezpečnostními zranitelnostmi, které se běžně objevují u mnohých aplikací. Představuje volně dostupné nástroje určené k využití při penetračním testování, a zároveň popisuje možnosti jejich použití pro daný účel. Cílem je zvýšit povědomí vývojářů a běžných uživatelů, kteří mají o tuto problematiku zájem, o možnostech provádění penetračních testů.

## Annotation

**Title:** Penetration testing tools and their usage

This diploma thesis deals with security tests of web applications and web pages. The main aim is to familiarize the reader with security vulnerabilities that commonly occur in many applications. Thesis presents freely available tools designed for use in penetration testing, and also describes the possibility of their use for that purpose. The aim is to raise awareness of developers and ordinary users who are interested in this domain, the possibility of performing penetration tests.

# OBSAH

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# ÚVOD

Internet a jeho využívání je dnes každodenní součástí života většiny z nás. Ať už to je v podobě emailové komunikace, internetového bankovnictví, či provádění nákupů nejrůznější položek na nejrůznějších stránkách k tomu určených. Exponenciální povaha růstu odvětví spojených s internetem je doložena počtem uživatelů internetu, kdy dle údajů Mezinárodní telekomunikační unie (ITU – International Telecommunication Union) bylo v roce 2000 online 400 milionů lidí, v roce 2005 bylo dosaženo první miliardy a tento trend růstu pokračuje až k dnešnímu stavu, kdy je k internetu připojeno téměř 3,5 miliardy uživatelů[1]. Nárůst uživatelů je samozřejmě těsně spojen s nárůstem poptávky po nepřeberném množství služeb. Tato poptávka je uspokojována vznikem internetových stránek a aplikací, které tyto služby přímo poskytují nebo slouží k jejich zprostředkování. Užívání služeb je spojeno s konkrétními osobami, což vyžaduje shromažďování informací o těchto osobách. Každý uživatel, který dává data o své osobě k dispozici předpokládá, že je uděláno vše, aby se nemohla dostat do nepovolaných rukou. V tomto bodě je potřeba si uvědomit, že tomu tak mnohdy není. Zde se dostává ke slovu otázka bezpečnosti a ochrany soukromí. Studie Carnegie Mellon Univerzity zjistila, že na každých 1000 řádek kódu připadá 20 až 30 chyb. To znamená, že na 50 miliónů řádek kódu připadá 1 až 1,5 miliónu chyb, které mohou být zneužity. [1] To je v přímém rozporu s přesvědčením, že aplikace a potažmo uživatelská data, která obsahují, jsou dobře zabezpečena proti zcizení.

Tato práce se zabývá bezpečnostním testováním webových aplikací a stránek. Klade si za cíl seznámit nejen vývojáře, ale kteréhokoliv čtenáře, s bezpečnostními zranitelnostmi, které se běžně objevují u mnohých aplikací. Přestavuje nástroje, které jsou volně dostupné, a zároveň popisuje techniky, s pomocí kterých je možné využít tyto nástroje k odhalení bezpečnostních chyb aplikací. Cílem je zvýšit povědomí o možnostech provádění penetračních testů a představení možností jejich začlenění do vývoje aplikací.

Práce je členěna do čtyř kapitol, kdy první dvě kapitoly jsou úvodem do problematiky internetové bezpečnosti a penetračních testů. Třetí kapitola je věnována volně dostupným nástrojům určeným k provádění penetračních testů, jejich popisu a představení možností použití, které jsou doplněny krátkou praktickou ukázkou. Čtvrtá kapitola je věnována možnosti začlenění penetračních testů do procesu průběžné integrace při tvorbě webových aplikací.

[1] `http://www.itu.int/en/ITU-D/Statistics/`

# 1 BEZPEČNOST WEBU

Internetová bezpečnost je důležitým tématem této doby, které získává na významu s postupným přemisťováním stále větší části našich životů do digitálních světa. Protože to, co běžně na webu vidíme jsou funkční stránky a aplikace, mnoho z nás nenapadne se nad otázkou jejich bezpečnosti zamyslet. O to citelnější pak mohou být dopady, když dojde k bezpečnostnímu incidentu, jehož důsledkem může být například únik informací o bankovních účtech uživatelů.

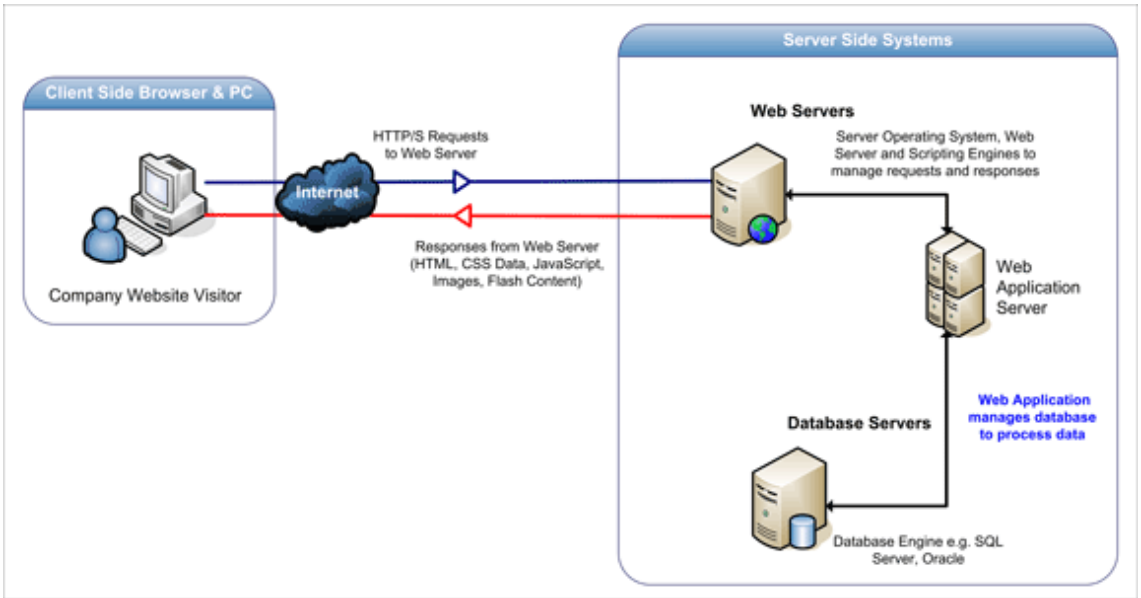
Tato kapitola přestavuje úvod do problematiky webových aplikací a jejich bezpečnosti. V první části se věnuje vysvětlení obecných principů fungování webových aplikací. Druhá část se věnuje vybraným pojmům bezpečnosti a kyberbezpečnosti. Vysvětluje, čím se tato disciplína zabývá a probírá některé základní pojmy z této oblasti. Zároveň jsou zde nastíněna možná zákonná omezení při testovaní webových aplikací. Poslední část kapitoly se zabývá komunitním projektem OWASP (Open Web Application Security Project).

## 1.1 Webová aplikace

Webová aplikace v softwarovém inženýrství je aplikace poskytovaná uživatelům z webového serveru přes počítačovou síť Internet, nebo její vnitropodnikovou obdobu (intranet). Webové aplikace jsou populární především pro všudypřítomnost webového prohlížeče jako klienta. [2] Ten se pak nazývá tenkým klientem, neboť sám o sobě logiku aplikace nezná. V současnosti se v však masivně rozšiřuje použití Javacriptových frameworků jako například AngularJS nebo React, které jsou navrženy tak, aby ve jménu rychlosti přemístili část práce ze serveru do internetového prohlížeče. Díky tomu již není dělící čára mezi tlustým a tenkým klientem tak výrazná.

Schopnost aktualizovat a spravovat webové aplikace bez nutnosti šířit a instalovat software na potenciálně tisíce uživatelských počítačů je hlavním důvodem jejich popularity, která je znásobena stále se zlepšující dostupností internetového připojení. Webové aplikace jsou používány pro implementaci mnoha podnikových i jiných informačních systémů, ale i freemailů, internetových obchodů, online aukcí, diskusních fór, webblogů [2] nebo internetového bankovnictví.

O tom jak webová aplikace funguje si lze udělat základní představu z obrázku 1.1, kde je znázorněn třívrstvý model fungování takové aplikace.

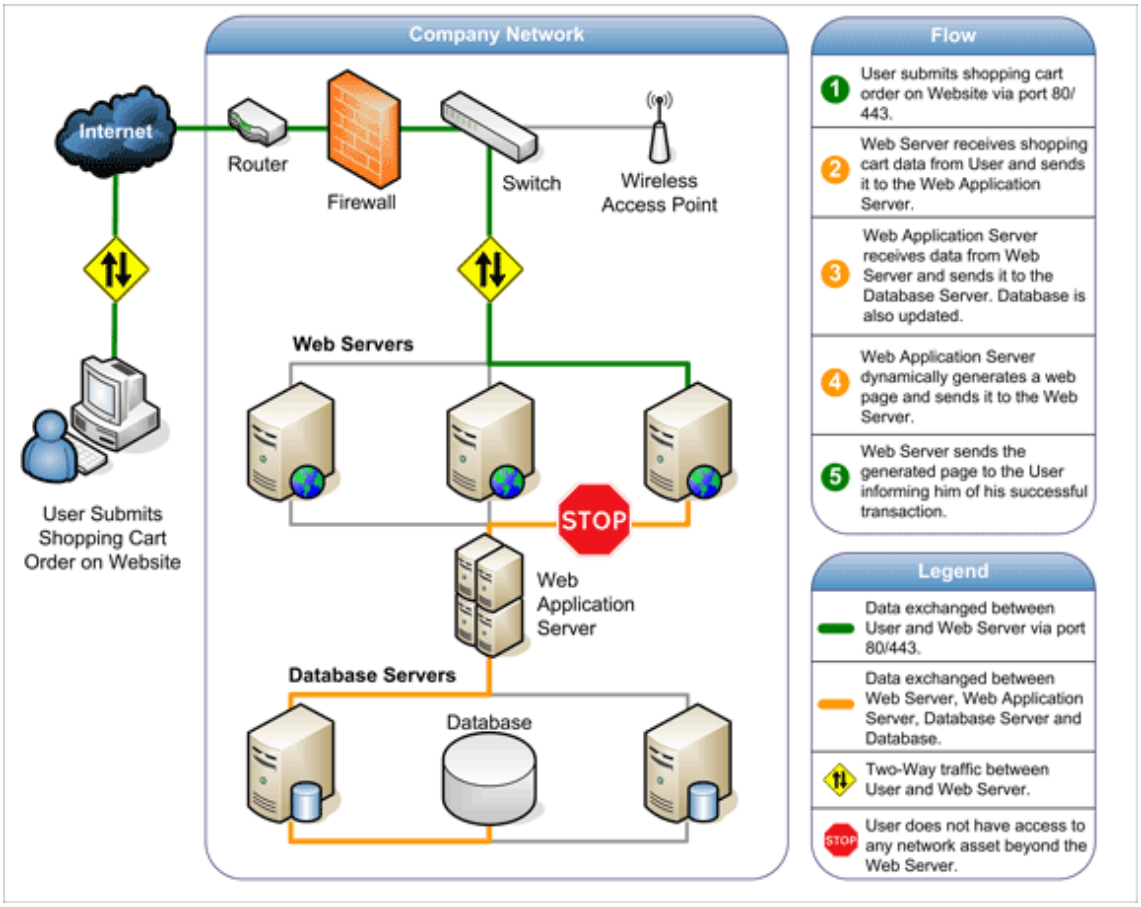



Obr. 1.1: Třívrstvý model fungování webové aplikace [4]

První vrstvu tvoří prezentační vrstva, kterou představuje například webový prohlížeč. Druhou vrstvu tvoří aplikační vrstva, kterou představuje technologie pro vytváření dynamického obsahu. Jako příklad můžeme úvesté technologie Java servlets nebo ASP (Active Server Pages). Tato vrstva zajišťuje výpočty a operace mezi vstupně-výstupními požadavky a daty. Třetí vrstvu tvoří tzv. datová vrstva. Ta zajišťuje práci s daty, může se nazývat také databázová vrstva. [4]

Dalším ilustračním obrázkem, který se snaží přiblížit princip fungování webové aplikace, je obrázek 1.2. Ten schématicky znázorňuje řetěz událostí, který spustí uživatel vytvořením požadavku (requestu) ve svém webovém prohlížeči.

Na obrázku jsou viditelná možná zařízení, která jsou na síti zapojena do zpracování tohoto požadavku. Legenda u pravého okraje obrázku popisuje jednotlivé události, které jsou vykonány v příslušných fázích zpracování. Ať už se jedná o síťová zařízení, servery nebo na nich provozované aplikační servery a databázové systémy, u všech těchto prvků existuje potenciální bezpečnostní riziko, jehož naplnění může vést k bezpečnostnímu incidentu.




Obr. 1.2: Schéma zpracování požadavku z webového prohlížeče [4]

## 1.2 Bezpečnost a kyberbezpečnost

Bezpečnost je dle terminologického slovníku pro krizová řízení a plánování obrany státu stav, kdy je systém schopen odolávat známým a předvídatelným (i nenadálým) vnějším a vnitřním hrozbám, které mohou negativně působit proti jednotlivým prvkům (případně celému systému) tak, aby byla zachována struktura systému, jeho stabilita, spolehlivost a chování v souladu s cílovostí. Je to tedy míra stability systému a jeho primární a sekundární adaptace. [6]

Kyberbezpečnost je pak soubor technologií, procesů a postupů určených k ochraně sítí, počítačů, programů a dat před útoky, poškozením nebo neoprávněným přístupem.

Součásti kyberbezpečnosti jako disciplíny mohou být dále:

- aplikační bezpečnost
- síťová bezpečnost
- disaster recovery – zotavení po hávarii a záložní systém
- busineess contiuity planning (plánování kontinuity)
- vzdělávání uživatelů

Problematickým prvkem kyberbezpečnosti je neustále se vyvíjející povaha bezpečnostních rizik. V praxi běžně praktikované soustředění většiny zdrojů na nejdůležitější části systému a ochrana proti největším rizikům se zanedbáváním méně důležitých systémových komponent je v současnosti nedostačující. Vhodnější je aktivní přístup, který vyžaduje adaptaci na nové hrozby, které se mění rychleji, než naše představy o riziku. V praxi je doporučováno jako vhodnější průběžné sledování a vyhodnocování hrozeb v reálném čase v kombinaci s aktivním sledováním informací o nových hrozbách.

Mezi možné zdroje pro sledování aktuálního dění na poli bezpečnosti, lze doporučit stránky týmu CSIRT.CZ[1] nebo stránky `kyberbezpecnost.cz`[2]. Bezpečnostní tým CSIRT.CZ je národní CSIRT (Computer Security Incident Response Team) tým, který provozuje sdružení CZ.NIC, správce domény .CZ. Hlavním posláním CSIRT.CZ je koordinace řešení bezpečnostních incidentů v počítačových sítích provozovaných v České republice a plnění role národního CSIRT týmu podle Zákona o kybernetické bezpečnosti. [9][3]

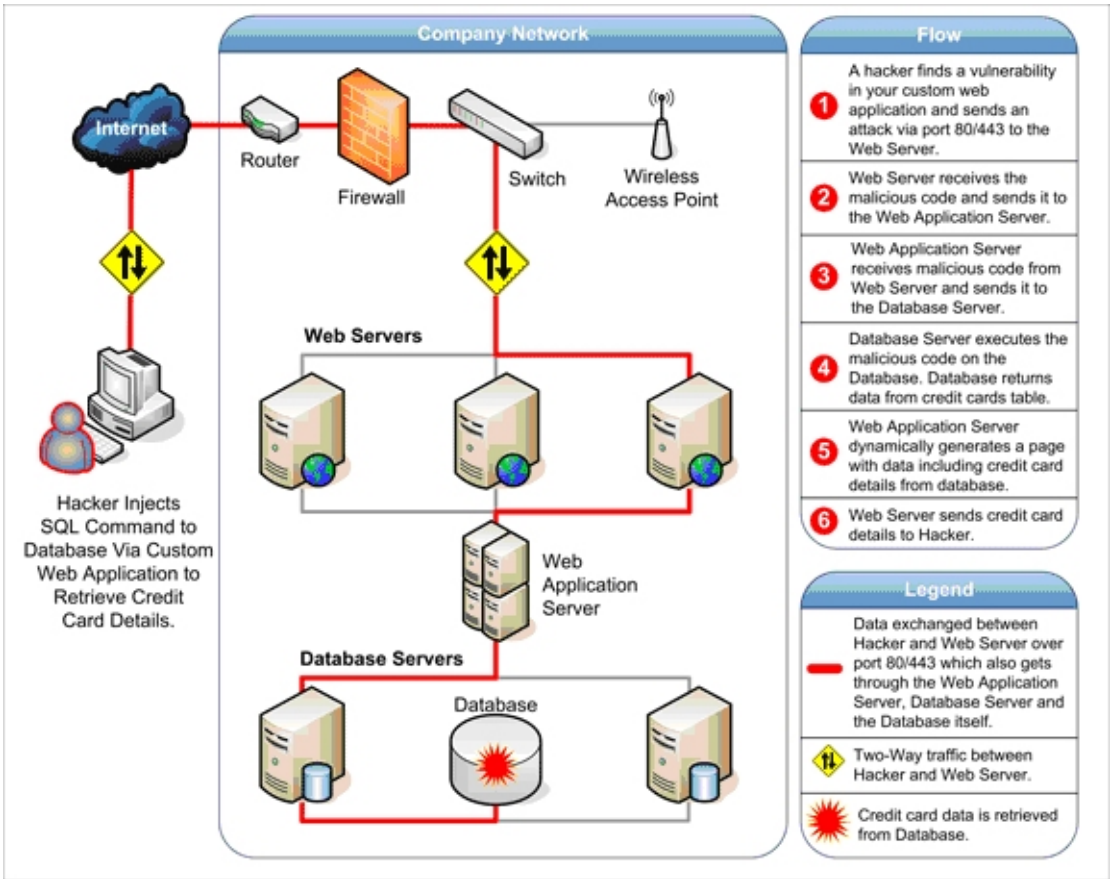
### 1.2.1 Triáda informační/aplikační bezpečnosti

V souvislosti s informační nebo aplikační bezpečností je často zmiňovaná trojice pojmů související s požadavky, které jsou kladeny na informační systémy a podnikovou bezpečnost obecně. Nežádoucí odhalení, modifikace nebo zničení určitých informací může vést k finanční ztrátě, poškození dobrého jména společnosti a v nejhorším případě i k ohrožení života jejich zaměstnanců nebo klientů. [7] Z těchto důvodů je nutné, aby byla na všech úrovních zajištěna:

- Důvěrnost (confidentiality) souvisí se schopností ujistit se, že je vynucena nezbytná úroveň míry utajení v každém okamžiku, kdy dochází ke zpracování dat a je zajištěna prevence jejich neautorizovaného vyzrazení.
- Integita (integrity) je udržena, když je zajištěno, že data jsou přesná, se zaručeným obsahem a jsou provedena opatření proti jejich neautorizované změně.

[1] `https://csirt.cz/`
[2] `https://www.kyberbezpecnost.cz/`

- Dostupnost (availability) je spolehlivá a včasná dispozice dat a zdrojů autorizovaným jednotlivcům.




Obr. 1.3: Schématické znázornění útoku na webovou aplikaci [4]

Na obrázku 1.3 je schématicky znázorněn možný průběh útoku na data uživatele. Legenda popisuje útok, který v konečném důsledku vede ke zcizení informací o uživatelově kreditní kartě a porušení důvěrnosti, která měla zabezpečit utajené uložení těchto informací.

### 1.2.2 Obecné základní pojmy

Jako součást úvodu do problematiky kyberbezpečnosti jsou v této podkapitole uvedeny a objasněny některé často používané termíny s ní spojené.

#### Útočník

Jako útočník (threat agent) se u kybernetických útoků označuje osoba, která realizuje samotný útok. V mainstreamových médiích se však setkáváme spíše s pojmem

*hacker*. Termín *hacker* je však v tomto kontextu používán neoprávněně, neboť jeho původní význam označoval počítačové specialisty a programátory s detailními znalostmi fungování systému, kteří ho dokázali výborně používat, ale především si ho i upravit podle svých potřeb. Ve významu, který pojmenovává útočníka, je vhodnější použití termínu *cracker*. Dále je možné v tomto kontextu zmínit termíny „Black hat“ a „White hat“. Kdy první termín označuje útočníka, který provádí útoky za účelem napáchání škod. Druhý termín označuje útočníka, který útoky provádí za účelem nalezení zranitelností, avšak se záměrem učinit po jejím nalezení kroky, které povedou k jejich odstranění.

### Vektor útoku

Vektor útoku (attack vector) je v zásadě způsob, jakým dochází ke zneužití zranitelnosti a kompromitaci cílového systému ať už formou vykonání škodlivého kódu, nebo například za použití technik sociálního inženýrství, případně kombinací obojího.

Jako konkrétní příklady vektorů útoku můžeme uvést:

- nevyžádanou poštu a SPAM - příloha mailu obsahuje škodlivý kód
- trojanizovanou aplikaci - aplikace obsahuje škodlivý kód
- vyjímatelná média - škodlivý kód se nachází na vyjímatelném médiu
- drive-by download malware - škodlivý kód se nachází na webu, kde se uživatel nedopatřením dostane ke stránce s exploitem

### Zranitelnost

Zranitelnost (vulnerability) je obvykle nějaká chyba v softwaru, ale může se jednat i o nedostatečné bezpečnostní povědomí, či selhání člověka, který je zpravidla nejslabším článkem celého systému. Nedostatečné znalosti administrátora můžou vést například ke špatné konfiguraci webové serveru, jehož zabezpečení jinak neobsahuje známou zranitelnost.

### Exploit

Exploit je zpravidla jednoduchý kód, ale může být i silně polymorfní a obfuskovaný (obfuskace je libovolná technika, jejímž cílem je znemožnění čtení zdrojového kódu), který zneužívá nějaké konkrétní zranitelnosti v systému nebo aplikaci a doručuje payload.

### Payload

Payload je kód, který je spuštěn poté, co byla zneužita daná zranitelnost a útočník např. unikl ze sandboxu prohlížeče a může zapisovat na disk. Zatímco exploit zneužívající konkrétní zranitelnost je v zásadě stejný kód, tak payload může být pokaždé jiný.

### Shellcode

Shellcode bylo původně označení pro kód, který se spouštěl v rámci payloadu v shellu na napadeném systému, kde se podařilo zneužít nějakou zranitelnost. Dnes už to ale nemusí být jen akce v příkazovém řádku.

Útočníci mohou na napadeném stroji provádět různé akce. Zatímco jeden útočník zneužije např. známou zranitelnost v Adobe Flash Playeru k šíření ransomware, tak druhý útočník ji může zneužít k šíření bankovního malwaru a třetí zase může napadený stroj začlenit do botnetu a následně z něj vést DDoS útok. [8]

### Hrozba

Hrozba je skutečnost, událost, síla nebo osoby, jejichž působení (činnost) může způsobit poškození, zničení, ztrátu důvěry nebo hodnoty aktiva. Hrozba může ohrozit bezpečnost (např. přírodní katastrofa, hacker, zaměstnanec aj.)

### Riziko

Riziko je pravděpodobnost, s jakou bude daná hodnota aktiva zničena nebo poškozena působením konkrétní hrozby, která působí na slabou stránku této hodnoty. Je to tedy míra ohrožení konkrétního aktiva. [10]

Hodnocení rizik je komplexní záležitost a důležitá součást vyhodnocení penetračního testování. K tomuto účelu byly v minulosti vyvinuty různé techniky. Mezi nejznámější patří systém hodnocení bezpečnostních rizik užívaný firmou Microsoft s názvem DREAD [11] (název je odvozen z počátečních písmen hodnocených kategorií *Damage, Reproducigbility, Exploitability, Affected users, Discoverability*) nebo technika OWASP Risk Rating Methodology [12]. Druhá zmíněná technika je podrobněji popsána v kapitole 3.5.9, neboť tato technika je používána ke klasifikaci zranitelností ve testovacích reportech nástroje OWASP ZAP.

**Zranitelnost**

Je 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. Tyto zranitelnosti jsou buď známé a publikované, ale výrobcem ještě neošetřené, nebo skryté a neobjevené. V případě skrytých zranitelností je důležité, zda je objeví dříve útočník, výrobce, bezpečnostní analytik, či uživatel. Bezpečnostní zranitelnosti jsou proto potenciálními bezpečnostními hrozbami. [13]

### 1.2.3 Zákon a testovaní bezpečnosti webových aplikací

Při ověřování bezpečnosti webových aplikací je nutné brát na zřetel také právní stránku věci. V rámci legislativy ČR je relevantní především Zákon č. 40/2009 Sb., trestního zákoníku. [14] Jedná se zejména o části:

Část druhá, Hlava II: Trestné činy proti svobodě a právům na ochranu osobnosti, soukromí a listovního tajemství; Díl 2 § 180 – 184 Trestné činy proti právům na ochranu osobnosti, soukromí a listovního tajemství:

- § 180 Neoprávněné nakládání s osobními údaji
- § 182 Porušení tajemství dopravovaných zpráv
- § 183 Porušení tajemství listin a jiných dokumentů uchovávaných v soukromí

Část druhá, Hlava V: Trestné činy proti majetku:

- § 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

Při provádění penetračních testů nebo třeba i během firemních workshopů k tématu bezpečnosti webových aplikací je vhodné zohlednit právní dopady těchto aktivit. Především je zapotřebí mít povolení aplikaci testovat, držet odpovídající právní jištění. Také je potřeba zohlednit, jestli testování probíhá na ostrých datech, případně jestli svojí aktivitou nemůžeme někoho poškodit.

K účelům tréningu nebo demonstrace zranitelností je proto v tomto kontextu nejvhodnější použít trenažéry zranitelností. Jedná se o webové aplikace, které záměrně obsahují zranitelnosti. Ty jsou volně dostupné k stažení a testování. Jejich stručný přehled je v této práci k nalezení v kapitole 3.3.

## 1.3 OWASP

OWASP (Open Web Application Security Project) je projekt, jehož hlavní náplní je bezpečnost webových aplikací. OWASP byl zahájen v roce 2001 a od roku 2004 funguje jako oficiální nezisková organizace OWASP Foundation ve více než 100 zemích světa. Jedná se o komunitu s více než 42 000 členy, díky čemuž představuje nejspíše největší komunitu na světě, která se zabývá bezpečností aplikací. OWASP nabízí bezplatně bezpečnostní nástroje, standardy, výzkumné projekty, e-mailové konference, místní pobočky po celém světě, kompletní knihy o testování bezpečnosti aplikací, vývoji bezpečného kódu a bezpečnostní revizi kódu a mnoho dalšího. [15]

V rámci tohoto projektu je mimo jiné zveřejňován seznam deseti nejkritičtějších zranitelností webových aplikací OWASP top 10[3]. Tento seznam je zveřejňován přibližně každé tři roky, přičemž poslední verze je z roku 2013. Podle dostupných informací se předpokládá vydání aktualizovaného seznamu v tomto roce (2017) nicméně na základě informace, že bylo vydání z roku 2016 odloženo z důvodu minimálních změn v seznamu, lze brát verzi 2013 jako aktuální ve vztahu k současným hrozbám.

### 1.3.1 OWASP top 10 2013

Roku 2013 byla určena následující nejzávažnější rizika webových aplikací:

1 **A1: Injektování (Injection)** – ke zranitelnostem injektováním, např. SQL, OS a LDAP, dochází, když se jako součást příkazu nebo dotazu odesílají do interpretu nedůvěryhodná data. Útočníkova nepřátelská data mohou lstí přimět interpret k provedení nezamýšlených příkazů nebo k umožnění přístupu k datům bez řádné autorizace.

2 **A2: Špatná autentizace a krádež session (Broken Authentication and Session Management)** – funkce aplikací, které se vztahují k ověřování a správě relace, často nejsou provedeny správně, což útočníkům umožňuje kompromitovat hesla, klíče nebo tokeny relací anebo zneužít jiné slabiny v implementaci k tomu, aby převzali identitu jiných uživatelů.

3 **A3: Cross-Site Scripting** (XSS) – chyby typu XSS nastávají tehdy, když aplikace přijme nedůvěryhodná data a odešle je webovému prohlížeči bez řádného ověření nebo escapovaní. XSS útočníkům umožňuje spouštět skripty v prohlížeči oběti, které mohou unést uživatelské relace, přetvořit webové stránky nebo přesměrovat uživatele na nebezpečné stránky.

[3] `https://www.owasp.org/index.php/Top_10_2013-Top_10`

4 **A4: Nezabezpečené objekty dostupné přes přímé reference (Insecure Direct Object References)** – přímý odkaz vznikne, když vývojář vystaví odkaz na vnitřní objekt implementace, například soubor, adresář nebo databázový klíč. Bez kontroly řízení přístupu nebo jiné ochrany mohou útočníci manipulovat s těmito odkazy, a získat tak neoprávněný přístup k datům.

5 **A5: Chybná konfigurace zabezpečení (Security Misconfiguration)** – dobré zabezpečení vyžaduje mít definováno a nasazeno bezpečné nastavení aplikace, frameworků, aplikačního serveru, webového serveru, databázového serveru a platformy. Bezpečnostní nastavení by měla být definována, prováděna a udržována, protože výchozí hodnoty jsou často riskantní. Navíc by měl být software průběžně aktualizován.

6 **A6: Vystavování citlivých dat (Sensitive Data Exposure)** – mnoho webových aplikací nechrání náležitě citlivá data, jakými jsou čísla kreditních karet a autorizační údaje. Tato slabě chráněná data útočníci mohou krást či modifikovat, aby mohli provádět podvody s kreditními kartami, krádeže identity nebo jiné zločiny. Citlivá data si zaslouží zvláštní ochranu, např. šifrováním dat v klidu nebo v pohybu, stejně tak i zvláštní bezpečnostní opatření pro data v prohlížeči.

7 **A7: Absence kontroly oprávnění (Missing Function Level Access Control)** – většina webových aplikací ověří úroveň přístupových oprávnění k funkcím před tím, než je takto funkcionalita viditelná v uživatelském rozhraní. Přesto je zapotřebí, aby se při přístupu ke každé funkci prováděla stejná kontrola přístupu na serveru. Jestliže požadavky nejsou verifikovány, útočníci budou moci vytvořit požadavky na získání přístupu k funkcionalitě bez řádného povolení.

8 **A8: Cross-Site Request Forgery** – útoky typu CSRF donutí prohlížeč přihlášené oběti odeslat zranitelné webové aplikaci podvržený požadavek HTTP, včetně cookie relace oběti a jiných automaticky vkládaných autentizačních informací. To útočníkovi umožňuje donutit prohlížeč oběti generovat požadavky, které zranitelná aplikace považuje za legitimní požadavky oběti.

9 **A9: Používání komponent známých svou zranitelností (Using Known Vulnerable Components)** - komponenty, např. knihovny, frameworky a další softwarové moduly, téměř vždy běží s nejvyšším oprávněními. Jestliže je zranitelná komponenta zneužita, útok může usnadnit závažnou ztrátu dat nebo ovládnutí serveru. Aplikace používající komponenty se známými zranitelnostmi mohou zmařit ochranu aplikací a umožnit řadu útoků a dopadů.

10 **A10: Nevalidované přesměrování na jiné weby (Unvalidated Redirects and Forwards)** - webové aplikace často přesměrovávají a předávají uživatele na jiné webové stránky a používají k určení cílové stránky nedůvěryhodné údaje. Bez řádného ověření mohou útočníci přesměrovat oběti na phishingové nebo malwarové stránky nebo použít předání k řízení přístupu k neoprávněným stránkám. [16]

Jednotlivé body tohoto seznamu v sobě shrnují mnoho zranitelností, proti kterým existují různé způsoby útoků. Tato práce zmiňuje ty nejčastější, mezi které lze zařadit: SQL injection, Cross-Site Scripting, CSRF atd.

# 2 PENETRAČNÍ TESTOVÁNÍ

Penetrační testy tvoří nedílnou součást bezpečnostní analýzy. Za použití specializovaných nástrojů jsou prováděny penetrační testy, tedy pokusy proniknout do různých částí informačního systému zvenčí či zevnitř. Výsledkem těchto testů je odhalení slabých míst v ochraně informačního systému, což v ideálním případě vede k následnému odstranění těchto slabin. Tato kapitola představuje úvod do problematiky penetračního testování. Je rozdělena na dvě hlavní části, přičemž první část se zabývá metodologií testování a druhá část běžnými typy prováděných testů.

## 2.1 Metodologie testování

Tato kapitola člení proces penetračního testování webových aplikací do několika na sebe navazujících dílčích celků. V dostupné literatuře lze narazit na popis mnohých metodik, které se mohou lišit například v počtu kroků nebo i v jiných částech. Mezi hojně používané lze zmínit metodiku OSSTMM (Open Source Security Testing Methodology Manual) [17] nebo metodiku NIST-SP800-115 [18]. Vzhledem k tomu, že hlavním cílem práce je představení nástrojů pro penetrační testování, bylo v této práci vycházeno z obecných metodik pro penetrační testování a zvoleno členění na čtyři fáze. Pro druhou a třetí fázi budou v dalších kapitolách představeny volně dostupné nástroje. Jedná se o fáze:

- Fáze 1: Určení cíle a rozsahu penetračních testů
- Fáze 2: Průzkum a získávání informací
- Fáze 3: Skenování a exploitace
- Fáze 4: Zhodnocení provedených testů

#### Fáze 1: Určení cíle a rozsahu penetračních testů

Fáze je určena k analýze obecných zadání a vytipování detailnějších cílů na které se později zaměří penetrační testy. Vzhledem k tomu, že není prakticky možné pokrýt testy vše na 100%, je hlavním cílem vymezit priority, které pak budou cílem samotného testování.

Příklad zadání může být otestovaní webové aplikace. Je třeba určit co konkrétně je potřeba otestovat. Je potřeba otestovat bezpečnost přihlašování? Bezpečnost uživatelských sezení? Stabilitu aplikace při neočekávaném zatížení? Přístup ke konfiguračním souborům? Přístup k citlivým datům o uživatelích?

V této fázi je vhodné provést analýzu rizik a tu pak použít jako podklad ke stanovení cílů penetračních testů.

### Fáze 2: Průzkum a získávání informací

V návaznosti na výstup z fáze 1 je potřeba zjistit o konkrétních systémech maximální množství informací. Získávání informací je orientováno na základě zvoleného typu testů (black-box, white-box, grey-box). Cílem této fáze je vytvoření obrazu o testovaném subjektu, který nám následně pomůže při hledání dalších informací nápomocných při penetračním testování - příkladem může být zjištění verze CMS (Content Management System – systém pro správu obsahu) – na základě této informace pak můžeme hledat známou zranitelnost. Dalším příkladem může být zjištění verze webového serveru, které pomůže při následném skenovaní a určení adresářové struktury.

Sběr informací se může dále týkat například spřízněných společností, uživatelských e-mailových účtů, telefonních čísel, typu používaných zařízení a operačních systémů a mnoha dalších informací.

### Fáze 3: Skenování a exploitace

Na počátku třetí fáze, a s využitím zjištění z předchozí fáze, je možné provést skenovaní cíle. To je prováděno nástroji, které budou zmíněny v příslušné části této práce. Tyto nástroje jsou schopné vytvořit tzv. zrcadlovou kopii serveru, kterou pak lze dále analyzovat. Hlavní výhodu této analýzy je její pasivní charakter, kdy po vytvoření obrazu cíl neví o tom, že je zkoumán. Tester pak může nepozorovaně ověřit logickou strukturu adresářů na serveru, a v některých případech i testované webové aplikace. Při troše štěstí může i například zjistit, kam aplikace ukládá informace o nabízených produktech, kam informace o uživatelských účtech atd.

Následně přichází testy zabezpečení a pokusy o prolomení bezpečnostních mechanismů. Cílem exploitace může být kupříkladu získání přístupu do systému nebo do databáze bez validních přihlašovacích údajů, získání citlivých informací o uživatelích nebo znepřístupnění nějaké služby.

Stejně jako ve všech ostatních oblastech ani v informačních technologiích neexistuje dokonalý produkt. Při tvorbě produktu vstupuje do tohoto procesu množství proměnných a stačí drobná chyba, která může vést k vytvoření závažné bezpečnostní slabiny. Cílem této fáze je odhalení této slabiny a její následné napravení, ne zneužití.

**Fáze 4: Report**

Poslední fáze s sebou přináší tvorbu reportu, který shrnuje výsledky a zjištění předchozích třech fází. Jedná se o fázi stejně důležitou jako jsou ostatní fáze, jelikož právě na základě tohoto reportu je možné vyvodit závěry na straně zadavatele testovaní a následně zajistit napravení případných bezpečnostních nedostatků. Report by měl obsahovat informace o typu zranitelnosti, možnostech opětovného nasimulování případně zhodnocení možných dopadů této zranitelnosti. U hodnocení dopadů je možné vycházet z předchozí analýzy rizik.

## 2.2 Typy testů

Nejběžnější penetrační testy současnosti jsou testy webových aplikací. Existují různé typy testů, které můžeme řadit například podle hloubky, do které jdou, a tedy i kvality výstupu, který nabízejí. Tato kapitola zpřehledňuje jejich rozdělení z několika základních hledisek jako je úroveň znalosti systému a způsob provedení testů.

### 2.2.1 Fuzz testování

Fuzz testování, neboli anglicky fuzzing či fuzz testing, v překladu testování neplatnými vstupními daty, je metoda podstrkávání nevalidních dat za účelem aktivování chyby či vady, která by mohla vést k objevení zranitelnosti. Jde o proces předpokládání chyb v programování a způsobů, jak jich docílit. Příkladem může být formulář, kde je uživatel dotázán na věk. Do neošetřeného pole, které očekává vstup zapsaný číslicemi, se napíše věk slovně a po odeslání formuláře se sleduje, jak na to aplikace zareaguje. [19] Fuzz testování můžeme rozdělit do dvou hlavních kategorií, na rekurzivní a záměnné. V první se testují všechny možné kombinace, kde příkladem může být testovaný parametr, o kterém je známo, že se skládá z 8 znaků a povolené jsou pouze velká a malá písmena anglické abecedy spolu s číslicemi. V tomto případě existuje $62^8$ možných kombinací, které je možno otestovat. [20] Druhou kategorii tvoří takzvané vektory, kam se řadí nejrůznější řetězce často vedoucí k určité zranitelnosti. [19] Těch však mohou být stovky až tisíce a bez znalosti vnitřní struktury aplikace se může takovéto testování jevit jako zbytečné. Někdy ale stačí i jednoduchý vektor, který povede k objevení zranitelnosti. Nejrůznější vektory jsou často používány při útocích na databázi u dotazu SQL, jejichž příkladem vyvolávající nestandardní chování aplikace může být i takto krátký vektor: OR 1=1 . Podle vzniklého SQL dotazu se lze tímto vektorem, nebude-li aplikace řádně zabezpečena,

kupříkladu přihlásit jako administrátor či vypsat z databáze nechtěné záznamy. Vektory nejsou omezeny jen na SQL injekce, je s nimi možné docílit i jiných zranitelností, jako je Cross-Site Scripting (XSS) či přetečení vyrovnávací paměti. [20]

### 2.2.2 Podle úrovně znalostní o systému

Toto rozdělení zohledňuje znalost penetračního testera, kterou má o testovaném systému. Běžně jsou uváděny tři úrovně které jsou popsány níže.

#### Black box

Při použití testů black box není k dispozici přístup k programovému kódu. Software si lze v tomto případě představit jako černou skříňku, jejíž obsah (zdrojový kód) není zvenčí viditelný. Neznáme tedy, jak přesně systém pracuje s daty. Můžeme pouze sledovat, jaký výsledek získáme po vložení vstupních dat. Je vhodná na testování všech úrovní, od jednotkových testů po akceptační, a je hojně využívána při testování webových aplikací, kde je dostupný jen již serverem zpracovaný kód. Princip této metody spočívá ve vytváření nejrůznějších vstupů a sledování, jak se s jejich pomocí změní výstup. Vzhledem k tomu, že tato metoda simuluje skutečné hackování, je proto vhodná na testování bezpečnosti. Nástroje představené dále v této práci předpokládají právě tuto úroveň znalostí o testované aplikaci. U black box způsobu testování je hojně využívána výše zmiňovaná technika fuzzingu. [24]

#### White box

U testů typu white box máme zdrojový kód k dispozici. Známe tak vnitřní strukturu testované aplikace. Tato metoda vyžaduje, aby auditor byl seznámen se všemi programovacími koncepty a funkcemi použitými v testované aplikaci. Důležité je pro tuto metodu dobře vedená dokumentace zdrojového kódu, bez které by se u větších projektů provádělo testování velice obtížně a zdlouhavě, a k dispozici musí být i samotný zdrojový kód. Rovněž se u obsáhlých aplikací manuální revize kódu nemusí vyplatit z hlediska časové náročnosti, a je proto lepší kód nechat zkontrolovat nástrojem, který sám najde potenciálně nebezpečné segmenty, které je pak možné individuálně zrevidovat. [25]

#### Gray box

Z důvodu kompletnosti je třeba se zmínit ještě o metodě gray box testování. Tento způsob testování není tak obvyklý a jasná není ani jeho přesná definice, která

se liší v závislosti na zdroji. Metoda je na pomezí mezi metodami black box a white box. Definovat ji lze jako black box metodu obohacenou o náhledy na funkčnost aplikace získané přes reverzní inženýrství. V metodě figuruje hlavně analýza zkompilovaných instrukcí jazyka symbolických adres, čemuž se říká binární audit. [23]

### 2.2.3 Podle způsobu provedení

V tomto rozdělení je zohledněn způsob provádění testů, který může ovlivnit kvalitu i časovou náročnost testů. Testy lze z tohoto hlediska provádět třemi způsoby.

#### Manuální

Manuální testy, jak název napovídá, jsou testerem vykonávány manuálně. Mezi výhody lze zařadit možnost vytvořit sofistikované procedury a testy na míru pro specifické podmínky, což automatické testy někdy nedokážou. Další velkou výhodou manuálních testů je, že je provádí člověk a je schopen popsat své záměry při provádění testů. Výsledky je schopen interpretovat i nezainteresovaným osobám, které nemají o dané oblasti potřebné znalosti (top management, vedení atd.). Za nevýhody je možné považovat časovou a znalostní náročnost. Vzhledem k téměř neomezeným možnostem, jak například vytvořit webovou aplikaci, jsou nezbytné rozsáhlé znalosti testované oblasti (HTML, SQL, JavaScript atd.). Časová náročnost je dále způsobena manuálním prováděním testů. [5]

#### Automatizované

Automatizované penetrační testy nabízejí výhody v rychlosti, možnostech, rozšiřitelnosti podle vlastních potřeb a v relativně jednoduché verifikovatelnosti a reprodukovatelnosti. Nástroje, které se využívají při automatizovaném testování, byly vytvořeny profesionály, kteří v dané oblasti pracují několik let. Další z výhod v porovnání s manuálními testy je kratší čas na zaučení a následnou aplikaci testů v praxi. Je totiž jednodušší (i časově) naučit se používat aplikaci pro provádění testů než pochopit princip celého testu prováděného manuálně. Mezi nevýhody je možné zařadit neschopnost prezentovat výsledky v uživatelsky přívětivé formě, či blíže vysvětlit podrobnosti k danému problému. Pro správnou interpretaci jsou opět nutné znalosti o použité aplikaci a testované oblasti. Další nevýhodou je také nemožnost testovat některé typy zranitelných míst. [5]

### Semiautomatizované

Třetí kategorií jsou semiautomatické testy. Jde o kombinaci automatických a manuálních testů. Představují kompromis mezi oběma formami se snahou o maximální využití výhod obou forem. [5]

Závěrem je třeba připomenout, že žádná forma testů nikdy nepokrývá 100 % kódu a všechny možnosti, tudíž ani neodhalí všechna přítomná zranitelná místa. Z toho důvodu je velice důležité ve fázi analýzy stanovit nejkritičtější místa aplikace, na která se pak testy mohou zaměřit důkladněji.

# 3 NÁSTROJE PRO PENETRAČNÍ TESTOVÁNÍ WEBOVÝCH APLIKACÍ

Tato kapitola představuje přehled volně dostupných nástrojů určených k využití při penetračním testování. Většina těchto nástrojů je obsažena ve výchozí konfiguraci vybraných linuxových distribucí zaměřených na provádění penetračních testů. Tyto distribuce jsou krátce představeny v úvodu kapitoly. Dále v kapitole jsou popsány nástroje využívané v jednotlivých fázích průzkumu. Do kapitol jsou tříděny v souladu s možnostmi použití při jednotlivých fázích průzkumu. U každého nástroje je po jeho krátkém představení nastíněna vhodnost použití pro konkrétní účel, doplněná o krátký příklad. Cílem práce je přiblížit a nastínit možnosti použití těchto nástrojů, ne však podat jejich kompletní výčet. Pro všechny nástroje je na internetu či v manuálových stránkách dostupná obsáhlá dokumentace, ze které lze v případě potřeby dále čerpat.

## 3.1 Linuxové distribuce

### Kali Linux

Kali Linux je linuxová distribuce vyvíjená pro účely penetračního testování. Tento operační systém obsahuje sadu všech základních nástrojů, které penetrační tester využije v jednotlivých fázích průzkumu.

Předchůdcem Kali Linuxu byla distribuce BackTrack. Distribuce byla aktivně vyvíjena v letech 2006 - 2012 a Kali Linux je jejím přímým nástupcem, což mimo jiné dokládá i logo draka, které mají obě distribuce společné.

První Kali 1.0 byl vydán v roce 2013. V distribuci došlo k zásadním změnám, proto byl zvolen odlišný název Kali Linux. Kali 2.0 vyšlo v roce 2015 a tato verze přinesla správce systému systemd. Poslední verze z roku 2016 přináší nový model aktualizace, tzv. rolling upgrade.

Dokumentace distribuce je dostupná na url http://docs.kali.org/. V rámci distribuce existuje aktivní komunita, tudíž v případě nějakého problému je zde velká šance nalezení odpovědi, která tento problém vyřeší. Jelikož je distribuce založená na distribuci Debian, lze případně využít i dokumentaci této distribuce, kterou lze najít na adrese www.debian.org/doc/.

### Parrot Security OS

Parrot Security OS (nebo také ParrotSec) je stejně jako distribuce Kali Linux GNU/LINUX distribuce založená na distribuci Debian. Je určená a navržená pro penetrační testování, hledání zranitelností, forenzní analýzu a k zajištění anonymity v rámci internetu. Od roku 2013 je vyvíjena komunitou Frozenbox Team. Distribuce byla v roce 2017 vyhodnocena serverem www.linux.com jako nejvhodnější distribuce pro systémové administrátory. Dokumentace distribuce je dostupná na stránkách docs.parrotsec.org.

Většina nástrojů, které jsou představeny v této práci, byla použita s využitím této distribuce. Nicméně všechny nástroje, které jsou v práci použity, jsou stejnou měrou zastoupeny v distribuci Kali Linux, tudíž postupy popsané v příslušných kapitolách lze použít napříč oběma distribucemi.

## 3.2 Průzkum a získávání informací

Jak již bylo zmíněno výše, jde o fázi shromažďování informací o cíli, jehož zabezpečení se penetrační tester pokouší prolomit. Informacemi mohou být otevřené porty, běžící procesy, aplikace jako neautentizované administrační konzole s výchozími hesly a další. Jedná se o velice důležitou fázi, jejíž výstup bude ve vysoké míře použit v dalším testování. Obecně se dá říci, že čím více informací o cíli můžeme shromáždit, tím větší je šance, že se nám podaří najít trhlinu v zabezpečení aplikace.

Techniky získávání informací o cíli lze rozdělit do dvou skupin, které jsou popsány níže.

### Aktivní techniky

Aktivní techniky se dostávají do přímého kontaktu s předmětem zájmu, tedy testovanou webovou aplikací. Jedná se například o skenování portů, prohledávání adresářové struktury aplikace atd. Společným prvkem aktivních technik je, že mohou být zkoumaným subjektem detekovány. Je tedy v zájmu testera, aby použil takové techniky a takovým způsobem, ale nevzbudily v testovaném subjektu podezření. Příkladem nevhodného použití může být například vytvoření nadměrné zátěže na aplikaci, což v důsledku povede k jejímu zpomalení.

### Pasivní techniky

Pasivní techniky jsou takové, které k získávání informací využívají nástrojů třetích stran. Jedná se o webové stránky jako Google nebo Shodan, které při vhodném použití poskytnou o subjektu užitečné informace. Výhodou pasivních technik je, že při jejich použití nedochází k přímé interakci s webovou aplikací, a tudíž na straně webové aplikace potažmo serveru (např. v podobě logů) nevzniká žádná informace o započatém průzkumu.

### 3.2.1 Fierce

Fierce je open source nástroj určený ke zjištění subdomén cílové webové stránky. Jedná se o skript napsaný v jazyce Perl a lze ho najít ve výchozích instalacích obou výše zmíněných distribucí. Níže jsou uvedeny dva příklady použití.

```
fierce -dns cilovaAdresa.cz
fierce -dns cilovaAdresa.cz -wordlist mujSeznamSlov.txt
```

Pro získání seznamu subdomén používá nástroj kombinaci technik zónového transferu z DNS serveru a slovníkového útoku hrubou silou. V prvním příkazu je nástrojem použit vlastní slovník nástroje. Ve druhém příkazu nástroj použije slovník z definovaného souboru `mujSeznamSlov.txt`.

Se stejným užitkem lze použít nástroj **dig**, který je dostupný na `*nix`ových systémech. Příklad použití nástroje **dig** může být:

```
dig @<cilovy-dns-server> <cilovy-host-nebo-adresa> axfr
dig <cilovy-host> ns
```

Prvním příkazem se pokusíme z DNS serveru získat za pomocí zónového transferu seznam domén. V případě, že neznáme adresu DNS serveru, lze ji získat z cílové adresy hosta druhým příkazem.

Zónový transfer obvykle slouží k synchronizaci záznamů domén primárního a sekundárního DNS serveru. Jedná se o funkcionalitu, která by neměla být externě dostupná. Nástroj tedy využívá nedostatku v konfiguraci serveru.

### 3.2.2 theHarvester

theHarvester je open-source nástroj určený k získávání různého typu informací z veřejně dostupných zdrojů. Mezi tyto informace patří názvy subdomén, emailové

adresy, jména zaměstnanců, otevřené porty atd. K získání těchto informací používá theHarvester kombinaci pasivních a aktivních technik. Příklad použití získání informací spojených se zadanou doménou za pomocí vyhledávače Google:

```
theharvester -d cilovaDomena.cz -b google
```

Za přepínačem `-b` je uveden zdroj k získávání informací. Další podporované zdroje jsou:

```
google, googleCSE, bing, bingapi, pgp, linkedin,
google-profiles, people123, jigsaw,twitter, googleplus, all
```

Dokumentace nástroje je dostupná při zadání samostatného příkazu `theharvester`.

### 3.2.3 SubBrute

Je open-source nástroj učený k získávání seznamu subdomén. Jedná se o skript, který napsaný v jazyce Python. Příklad použití je:

```
./subbrute.py cilovaDomena.cz
```

Nástroj není přítomný ve výchozí konfiguraci ani v jedné z uvedených linuxových distribucí. Lze ho stáhnout z Git repositáře https://github.com/TheRook/subbrute. Na stejném místě je dostupná dokumentace k nástroji obsahující další možnosti a příklady použití.

### 3.2.4 CeWl

S pomocí nástroje CeWl je možné získat seznam slov spojených s danou doménou. Tento seznam lze využít například jako slovník pro útok hrubou silou. CeWl je dostupný v obou představených linuxových distribucích, případně jej lze stáhnout ze stránek jeho autora Robina Wooda https://digi.ninja/projects/cewl.php, kde je také dostupná dokumentace a mnohé příklady užití. Příklad užití je příkaz:

```
cewl cilovaDomena.cz -d 1 -w cewl.txt
```

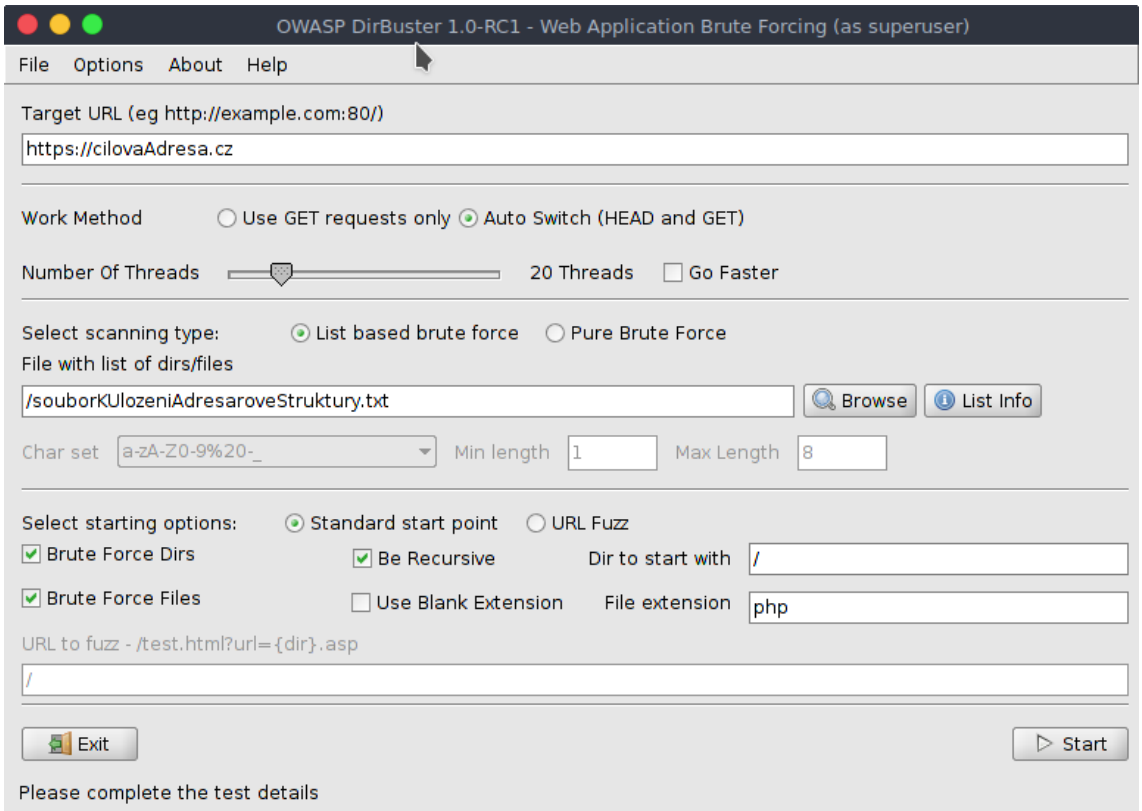
Přepínačem `-d` s parametrem 1 kontrolujeme hloubku zanoření v odkazech. Přepínač `-w` přesměruje výstup do souboru, který je předán jako parametr, v našem případě soubor `cewl.txt`. Další možnosti použití jsou dostupné v nápovědě, kterou lze vyvolat přepínačem `-h`.

### 3.2.5 DirBuster

DirBuster je nástroj, který mapuje adresářovou strukturu nacházející se na cílové adrese za pomocí útoku hrubou silou. Je napsaný v jazyce Java a vyvinutý OWASP komunitou. Nástroj disponuje grafický uživatelský rozhraním, které je vidět na obrázku 3.1.

DirBuster je schopný pracovat ve zvoleném počtu vláken, takže je schopen získávat požadované informace velice rychle. Tato funkcionalita však s sebou přináší i zvýšenou zátěž, která je vytvářená na cíl, a v případě špatné konfigurace ho je i schopná vyřadit z provozu. Pro nepozorovaný průzkum je tedy tuto konfiguraci nutné optimalizovat.

Vzhledem k typu útoku, tedy slovníkového, který nástroj využívá, mohou být některé výsledky falešně pozitivní. Je tedy doporučeno dodatečné ruční ověření výsledků poskytnutých DirBusterem.

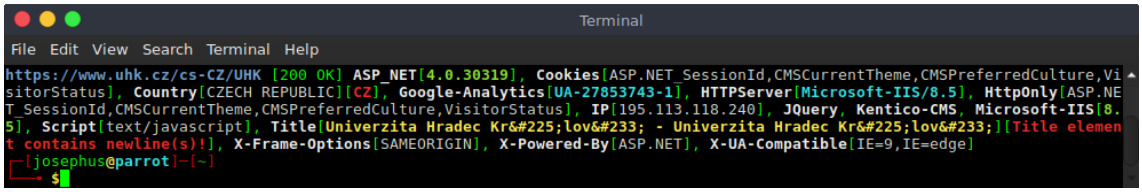



Obr. 3.1: GUI nástroje DirBuster

Alternativu k tomuto nástroji, použitelnou v příkazové řádce, nabízí nástroj `wfuzz`, který je dostupný ve výchozím nastavení popisovaných linuxových distribucí.

### 3.2.6 WhatWeb

K získání souhrnu vybraných informací je možné použít nástroj WhatWeb. Použití u domény `uhk.cz` je vidět na obrázku 3.2. Výhodou je, že nástroj poskytne prakticky okamžitě přehled rozličných informací v sumarizované podobě. To může poskytnou výchozí bod pro další analýzu a průzkum.

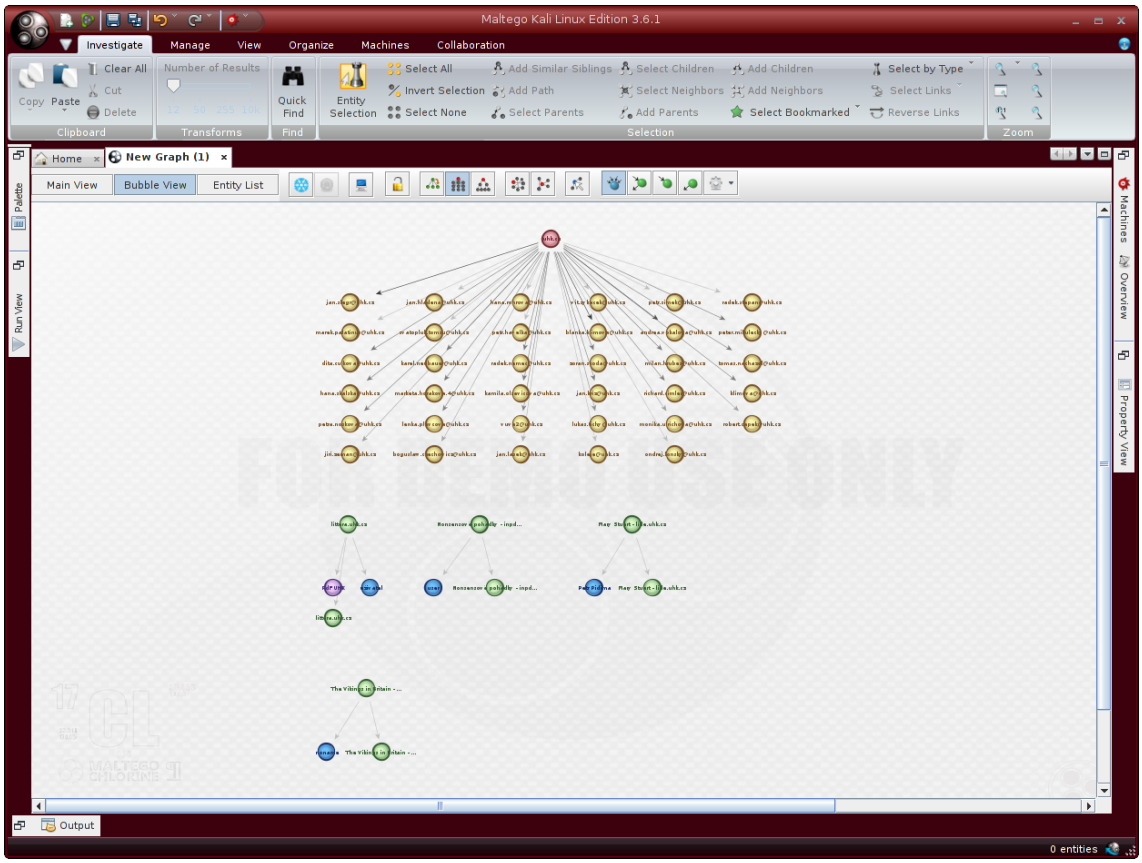



Obr. 3.2: Nástroj WhatWeb

### 3.2.7 Maltego

Maltego je učinný a komplexní nástroj vyvíjený společností Paterva. Je určený pro oblast forenzní analýzy a Open Source Inteligence. Nástroj umožňuje získávání informací o zadaném zdroji, jeho analýzu a přehlednou prezentaci v grafické podobě. Tento nástroj je v provozu od roku 2007. Z počátku existovala jak desktopová aplikace, tak webová verze nástroje. V současné době je k dispozici pouze desktopová aplikace. Aplikace je založena na platformě Java a její instalace jsou dostupné pro operační systémy Windows, OS X nebo Linux. Nástroj je ve výchozí konfiguraci dostupný v obou zde zmíněných distribucích. Jedná se o verzi Community Edition, která je poskytována zdarma po provedení registrace. Nevýhodou této instalace je její omezená funkcionalita, která však účelům, které jsou předmětem této kapitoly, poslouží dostatečně. V případě, že by pro někoho byla tato edice nedostatečná, je dostupná i zpoplatněná Commercial Edition tohoto produktu.

Grafické rozhraní nástroje usnadňuje práci a zároveň umožňuje prezentovat informace v grafech znázorňujících jejich provázanost. Nástroj pracuje s takzvanými *entitami*, což jsou informace zadané na vstupu. Takzvané *transforms* jsou jakési moduly, které zajišťují samotnou specifickou funkcionalitu. Každý modul například hledá jiný druh informací. Dále je možná instalace pluginů, které rozšiřují zdroje vyhledávání, kupříkladu plugin využívající vyhledávače Shodan. Nástroj je zobrazený na obrázku 3.3.




Obr. 3.3: Uživatelské prostředí nástroje Maltego

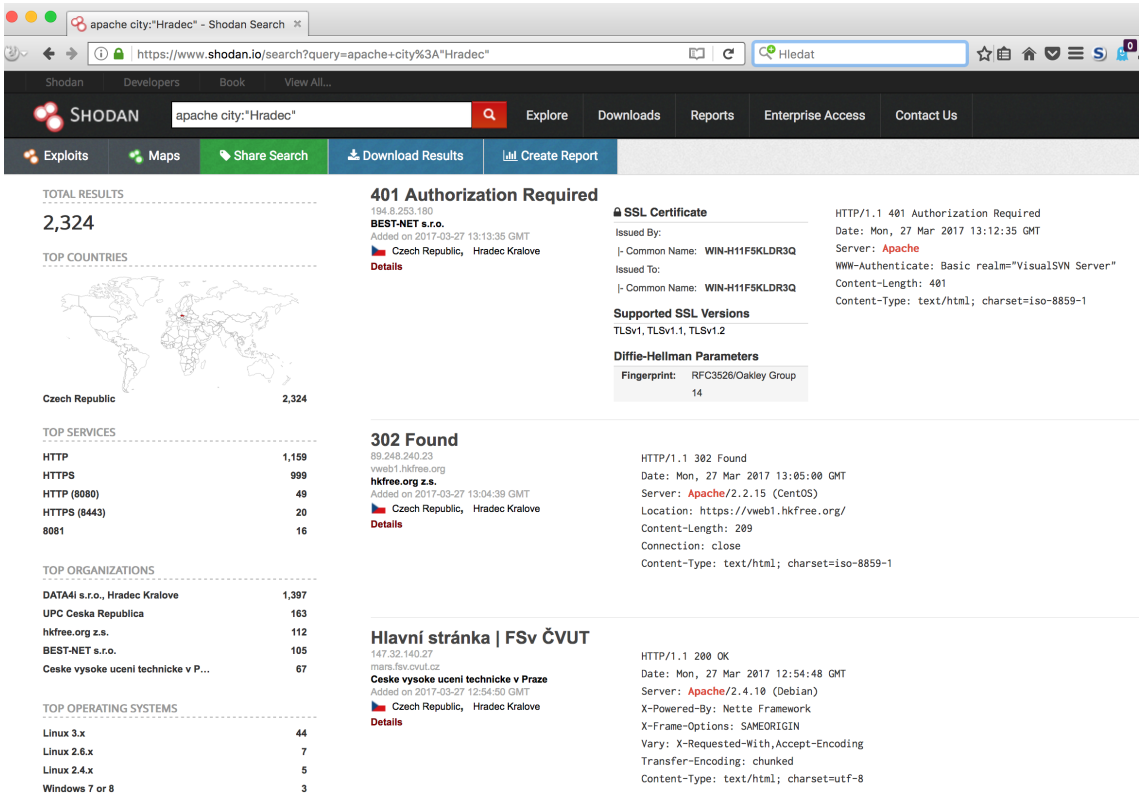
Na obrázku 3.3 je vidět graf, který Maltego vygeneruje. Jako vstup byla zadána doména a dále byly vybrány takzvané *stroje* (machine). *Strojem* (machine) je v Maltegu nazývána kategorie hledání tj. způsob jakým se informace mají hledat.

### 3.2.8 Shodan

Shodan je vyhledávací engine spuštěný v roce 2009 programátorem Johnem Matherlym, který je schopen vyhledávat specifická zařízení připojená k internetu. Shodan funguje na obdobném principu jako vyhledávače webových stránek s tím rozdílem, že nesbírá údaje z webových stránek, ale z takzvaných bannerů zařízení připojených k internetu. Jedná se o zařízení jako například webkamery, síťové prvky, tiskárny, skenery a další. Stejně jako v případě vyhledávání pomocí Google lze použít filtry ke zpřesnění vyhledávacích kritérií.

Shodan obsahuje podmnožinu komponent dělenou na:

1. **Exploits** – komponenta určená k vyhledání různých exploitů pro různé druhy operačních systémů, platform, aplikací atd.
2. **Maps** – je placená služba zobrazující výsledky zanesené do mapy
3. **Scanhubs** – je služba určená k hledání v sítích a podsítích. Podporuje analýzu za pomocí nástrojů Nmap a Masscan




Obr. 3.4: Výsledky hledání vyhledávače Shodan

Na obrázku 3.4 je zobrazen výsledek vyhledávání dotazu `apache city:"Hradec"`, který zobrazí servery Apache podle konkrétního města. Mezi další filtry, které lze při vyhledávání použít patří:

- `country:` najde zařízení podle země
- `geo:` najde zařízení podle geografických souřadnic
- `hostname:` najde zařízení podle hostname
- `net:` hledání v závislosti na IP nebo CIDR
- `os:` hledání podle operačního systému
- `port:` hledání podle otevřeného portu

Filtry lze libovolně kombinovat přidáním dalšího filtru odděleného mezerou. [26] Tímto způsobem lze například najít servery obsahující známou zranitelnost.

### 3.2.9 Pokročilé vyhledávání pomocí vyhledávače Google

Jednou z dalších pasivních technik, tedy technik, kdy cíl neví že se o něj útočník zajímá, je pokročilé vyhledávání pomocí vyhledávače Google. Vyhledávač Google poskytuje širokou paletu operátorů (příkazů, direktiv) umožňujících co největší zpřesnění kritérií vyhledávání tak, aby bylo nalezeno požadované. Příkazy se do vyhledávače zadávají ve formátu `prikaz:dotaz`. Jako příklad můžeme použít dotaz:

```
ext:xls site:fim.uhk.cz
```

Tento dotaz nám zobrazí všechny naindexované soubory s příponou `xls` v doméně `fim.uhk.cz`. Jestliže chceme vyhledávat v konkrétní cestě na webové stránce, můžeme použít direktivu `inurl`. Přehled direktiv s popisem možnosti použití lze vidět v tabulce 3.1.

Těmito rozšířenými možnostmi vyhledávače Google v souvislosti s bezpečnostními testy se zabývá disciplína Google hacking - neboli hackování Googlu. Jakousi biblí této disciplíny se stala publikace Google hacking for penetration testers [27], která vyšla ve dvou svazcích. Její autor, Johnny Long, je zároveň spojen s projektem Google Hacking Database[1], což je, jak už název napovídá, databáze dotazů, které pomocí vyhledávače Google odhalují nějakou konkrétní zranitelnost resp. zobrazí stránky s touto zranitelností. Samotná stránka zpřístupňující tuto databázi je členěna do několika kategorií podle typu zranitelnosti Jako příklad lze uvést například dotaz:

```
ext:sql intext:@email.cz intext:password
```

Dotaz hledá soubory s příponou `.sql`, které obsahují slovo `password` nebo text `@email.cz`. Tímto způsobem lze získat přístup ke špatně zabezpečenému souboru s citlivými údaji, který může být dále použit či zneužit.

[1] https://www.exploit-db.com/google-hacking-database/

| **Operator** | **Purpose** | **Mixes with Other Operators?** | **Can be used Alone?** | **Web** | **Images** | **Groups** | **News** |
|---|---|---|---|---|---|---|---|
| intitle | Search page Title | yes | yes | yes | yes | yes | yes |
| allintitle | Search page title | no | yes | yes | yes | yes | yes |
| inurl | Search URL | yes | yes | yes | yes | not really | like intitle |
| allinurl | Search URL | no | yes | yes | yes | yes | like intitle |
| filetype | specific files | yes | no | yes | yes | no | not really |
| intext | Search text of page only | yes | yes | yes | yes | yes | yes |
| allintext | Search text of page only | not really | yes | yes | yes | yes | yes |
| site | Search specific site | yes | yes | yes | yes | no | not really |
| link | Search for links to pages | no | yes | yes | no | no | not really |
| inanchor | Search link anchor text | yes | yes | yes | yes | not really | yes |
| numrange | Locate number | yes | yes | yes | no | no | not really |
| daterange | Search in date range | yes | no | yes | not really | not really | not really |
| author | Group author search | yes | yes | no | no | yes | not really |
| group | Group name search | not really | yes | no | no | yes | not really |
| insubject | Group subject search | yes | yes | like intitle | like intitle | yes | like intitle |
| msgid | Group msgid search | no | yes | not really | not really | yes | not really |

Tab. 3.1: Pokročilé operátory vyhledávače Google [28]

## 3.3 Bezpečnostní trenažéry

Vytvořit bezpečnou aplikaci bývá velmi často velice obtížný úkol. Stejně tak otestovat aplikaci z hlediska bezpečnosti tak, aby byla testy pokryta co možná nejširší škála zranitelných míst. Žádný penetrační tester se se svými znalostmi nenarodil. A jestliže se chce někdo tomuto oboru věnovat, vzniká potřeba ověřit si teoretické znalosti v praxi. V té chvíli se nabízejí tři možnosti. Testovat cizí webové aplikace bez povolení jejich vlastníka, testovat aplikace s povolením vlastníka a použít bezpečnostní trenažéry. Bez povolení se tester vystavuje riziku, že přijde do konfliktu se zákonem. S povolením toto riziko odpadá, ale výsledek je nejistý, protože není zřejmé jestli zde zranitelnost není, nebo jsme ji pouze nenašli. Třetí možnost, použití bezpečnostního trenažéru, přináší do začátku to nejlepší a nejjednodušší řešení. Testování neporušuje zákon, zranitelnosti jsou zdokumentovány a lze ověřit stav se zranitelností i bez ní.

Asi nejobsáhlejší řešení v této oblasti nabízí komunita OWASP se svým projektem OWASP Broken Web Application Project, který podrobněji představím v této kapitole.

### 3.3.1 OWASP Broken Web Application Project

Projekt OWASP Broken Web Applications Project[2]. je soubor zranitelných webových aplikací provozovaných na platformě Linux. Tento projekt je distribuován k volnému užití ve formě připraveného virtuálního stroje[3].

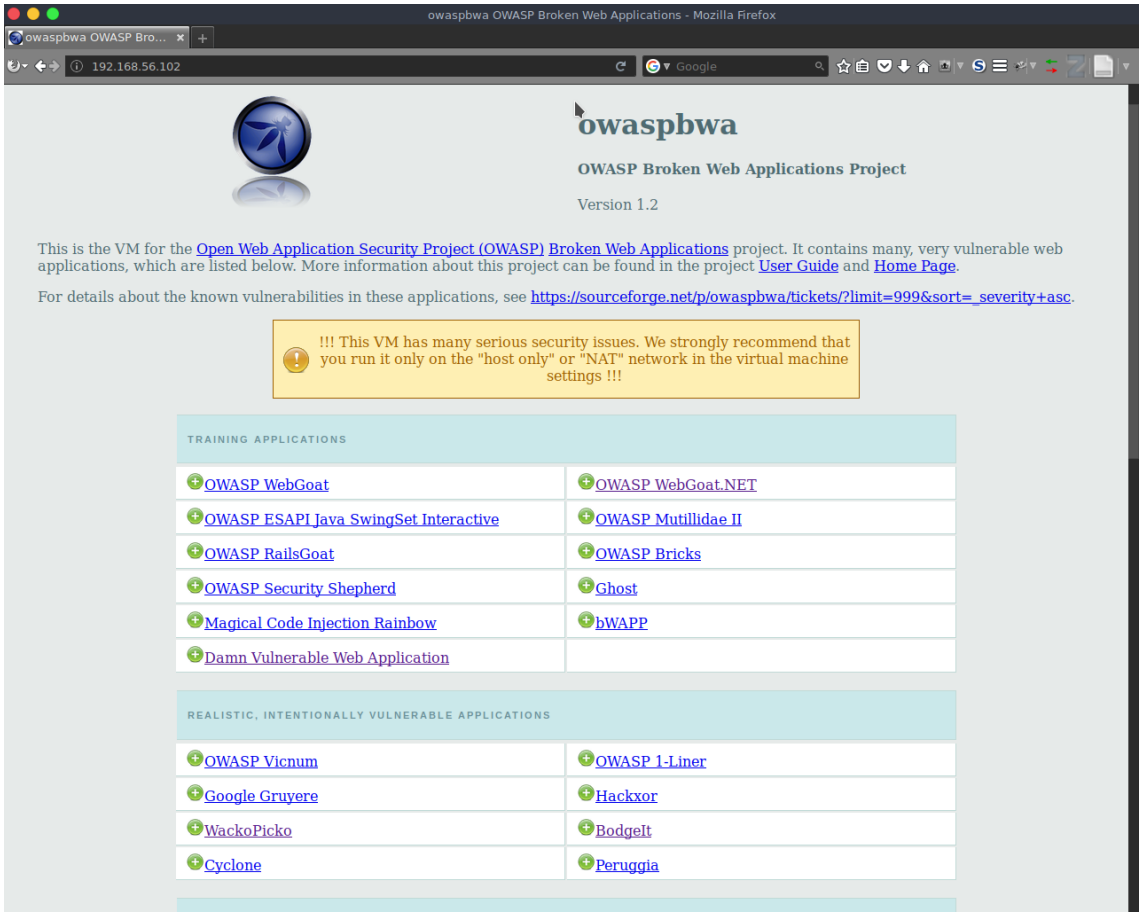
Projekt je určen pro:

- výuku webové bezpečnosti
- testování manuálních technik útoku
- testování automatizovaných nástrojů
- testování nástrojů pro analýzu zdrojových kódů
- sledování chování při útocích na webové aplikace
- testování aplikačních firewallů a podobných technologií

K běhu virtuálního stroje je zapotřebí lokální instalace virtualizačního prostředí, kterým může být například volně dostupný produkt VMware Player[4]. Po importu projektu do virtualizačního prostředí stačí projekt spustit a počkat na nastartování virtuálního stroje.

[2] `https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project`
[3] `https://sourceforge.net/projects/owaspbwa/files/`
[4] `http://www.vmware.com/products/player/playerpro-evaluation.html`

Webové rozhraní projektu je zobrazeno na obrázku 3.5. Nabídku aplikací určených k tréningu pak lze vyvolat po zadání IP adresy, která je v konfiguraci virtuálního stoje.




Obr. 3.5: OWASP Broken Web Application Project

Přístupová stránka nabízí aplikace naprogramované s pomocích různých programovacích jazyků, které využívají rozdílné technologie a frameworky. Pro přehlednost rozděluje aplikace do několika skupin:

- Tréningové aplikace
- Realistické, záměrně zranitelné aplikace
- Staré verze skutečných aplikací obsahující známou zranitelnost
- Aplikace určené pro využití specifických nástrojů
- Ukázky jednostránkových nebo velmi malých aplikací
- Aplikace AppSensor Demo komunity OWASP

Využití jednotlivých aplikací bude dále demonstrováno v kapitolách věnujících se nástrojům k testování.

## 3.4 Skenování a analýza

V předchozích kapitolách bylo zmíněno, že penetrační testy mohou být prováděny na základě různých znalostí o testovaném systému. V našem případě simulujeme skutečné chování aplikace v reálném prostředí a předpokládáme situaci druhu black-box, kdy útočník v počáteční fázi neví o systému žádné bližší informace. Ve fázi průzkumu a získávání informací jsme naše vědomosti o systému částečně rozšířili. Úkolem této fáze je však získání co možná největšího množství informací o samotné aplikaci a její struktuře. Zároveň je pro nás výhodné vytvoření funkční kopie aplikace k účelům offline průzkumu. Jelikož nejsme schopni vytvořit dynamickou webovou aplikaci, pomůžeme si nástroji odborně nazývanými jako Crawlery (prohledávače) a Spidery (pavouci). Tyto nástroje jsou schopné prozkoumat a uložit obsah webové aplikace ve statické podobě pomocí sledovaní odkazů na jednotlivých stránkách aplikace. Techniky a možnosti použití těchto nástrojů jsou předmětem této kapitoly.

### 3.4.1 Wget

GNU wget je utilita, která je součástí většiny Linuxových distribucí včetně těch, které jsou zmíněny v této práci. Wget je schopna rekurzivně stáhnou webovou stránku pro offline prohlížení a to i s transformací odkazů a stažení i jiných, než HTML souborů. Pro tuto funkcionalitu je nutné použití přepínačů, které budou demonstrovány na následujícím příkladu:

```
wget -r -P wackopicko_offline/ http://192.168.56.102/wackopicko/
```

Samotný příkaz `wget` stáhne pouze soubor `index.html`, proto je nutné příkaz parametrizovat pomocí přepínačů. Přepínačem `-r` říkáme, že chceme obsah stáhnou rekurzivně. Nástroj tedy projde všechny odkazy a uloží stránky, které na nich najde. Přepínač `-P` určuje adresář ke stažení obsahu. Další užitečné přepínače mohou být:

- `-l`: přepínač nasledovaný číslem, které vyjadřuje do jaké úrovně má nástroj sledovat odkazy
- `-k`: nastaví relativní cesty k odkazům, tak aby byly funkční
- `-p`: stáhne všechny obrázky včetně těch nacházejících se na jiných stránkách
- `-w`: přepínač následovaný číslem, které určuje, jak dlouho nástroj čeká mezi jednotlivými downloady. Lze využít, když se na serveru nachází mechanismus, který brání automatickému procházení obsahu

### 3.4.2 HTTrack

Obdobnou funkcionalitu jako nástroj `wget` nabízí nástroj HTTrack. Nástroj je ve výchozím nastavení dostupný v distribuci ParrotSec a umožňuje vytvoření funkční statické kopie webových stránek do lokálního úložiště. Nástroj lze pro vybranou URL jednoduše použít příkazem:

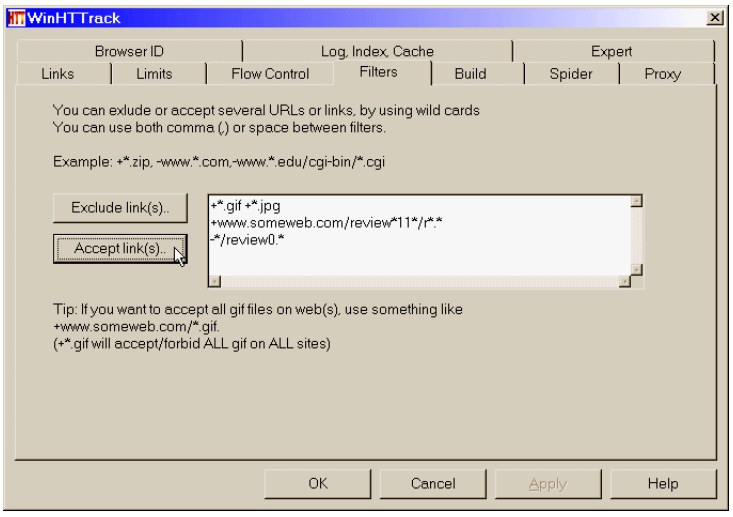
```
httrack http://192.168.56.102/wackopicko/
```

Po spuštění příkazu vytvoří nástroj do adresáře ve které se nacházíme:

- Složku s názvem serveru nebo adresou obsahující stažené soubory
- Soubor `cookies.txt`, který obsahuje informace o cookies, které byly použity při stahování obsahu stránek.
- Složku `hts-cache` obsahující seznamy stažených souborů.
- Soubor `hts-log.txt` obsahující informace o chybách a varováních, které nastaly během prohledávání.
- Soubor `index.html`, který je kopií originálního souboru stránek.

Nástroj lze také použít v kombinaci s přepínači, který nám umožňují přizpůsobit chování nástroje. Ty lze zobrazit při použití přepínače `-help`.

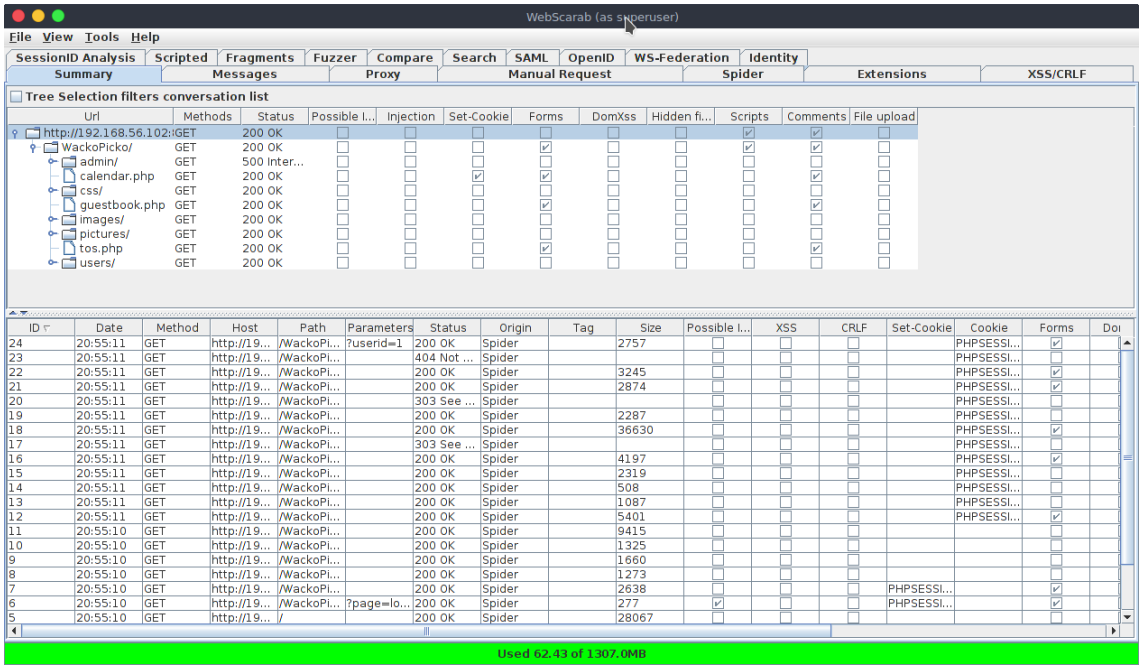
Původní verze nástroje je mířená na Linuxové operační systémy, ale v současné době je dostupná verze nástroje i pro operační systémy Windows. Pro tyto systémy je nástroj distribuován pod názvem WinHTTrack a disponuje i grafickým uživatelským rozhraním. Podoba této verze je vidět na obrázku 3.6




Obr. 3.6: GUI nástoje WinHTTrack

### 3.4.3 WebScarab

OWASP WebScarab Project je multiplatformní nástroj pro analýzu aplikací, které komunikují prostřednictvím HTTP a HTTPS protokolů. Nejčastěji se WebScarab používá jako záchytné proxy, čímž vývojářům a testerům umožňuje prozkoumávat komunikaci mezi prohlížečem a serverem. Je psán v Javě, umožňuje doplňovat, odstraňovat či modifikovat pluginy a i ve svém základním provedení se jedná o skvělý nástroj, který ocení především lidé, kteří se věnují bezpečnosti webových aplikací. [22]




Obr. 3.7: WebScarab

Kromě použití pro analýzu a editaci HTTP/HTTPS requestů umožňuje nástroj díky široké škále dostupných pluginů další použití například pro:

- analýzu WSDL
- jako spider nebo pro automatickou kontrolu souborů v aplikaci (například .bak, .zip aj.)
- pasivní analýzu - uživatel má možnost kontrolovat údaje v HTTP hlavičce a případně registrovat možná rizika typu CRLF Injection a XSS
- analýzu cookies - např. pro vizuální kontrolu nepredikovatelnosti SessionId
- sběr a analýza SessionID - z cookies anebo odpovídajícím regulárním výrazem
- automatizované nahrazování hodnot parametrů; vhodné pro odhalování rizik XSS, SQL Injection apod.

### 3.4.4 John the Ripper

John the Ripper je nástroj určený pro lámání hesel a dostupný pro linuxové systémy a Mac OS X. Je součástí představených bezpečnostních operačních systémů. Jedná se o pravděpodobně nejoblíbenější nástroj penetračních testerů, určený k lámání hesel. Disponuje množstvím užitečných vlastností, mezi které patří například schopnost rozpoznat většinu běžných šifrovacích a hashovacích algoritmů, nebo schopnost použití slovníků pro útok hrubou silou. Další užitečnou schopností je možnost generovat slovník s použitím vnitřních pravidel nástroje.

Následující příklad ukazuje vytvoření slovníku s využitím připravené slovní sady. Tuto sadu lze získat s pomocí nástroje CeWl způsobem popsaným v podkapitole 3.2.4 .

```
john --stdout --wordlist=dict.txt --rules > dict_tranformed.txt
```

K vysvětlení příkazu, parametr `stdout` vypíše použitá hesla, parametrem `wordlist` je přidán slovník a na něj pak aplikována transformační pravidla pomocí parametru `rules`. Pravidla jsou k nalezení ve výchozí složce `/etc/john/john.conf`, kde je také možná jejich libovolná modifikace. Výstup je pak přesměrován do souboru `dict_transformed.txt`. Výsledkem je pak vygenerování 260 hesel ze vstupu 8 hesel, jak je vidět na obrázku 3.8.

```
Josef-MacBook-Pro:PenTests josefcejka$ john --stdout --wordlist=dict.txt
Press 'q' or Ctrl-C to abort, almost any other key for status
test
123456789
password
heslo
hesloheslo
nejakeJmeno
jmeno123
heslo123
8p 0:00:00:00 100% 800.0p/s heslo123
Josef-MacBook-Pro:PenTests josefcejka$ john --stdout --wordlist=dict.txt --rule
s > dict_transform.txt
Press 'q' or Ctrl-C to abort, almost any other key for status
259p 0:00:00:00 100% 25900p/s Nejakejmening
Josef-MacBook-Pro:PenTests josefcejka$ wc -l dict_transform.txt
     259 dict_transform.txt
```

Obr. 3.8: John the Ripper – generování slovníku

Další možné použití nástroje, které by se mohlo řadit spíše do kapitoly exploitace, je například když jsou díky zranitelnosti SQL injection získána uživatelská data z databáze. Máme k dispozici údaje ve formátu `uzivatel:heslo`, kdy je pro ukrytí hesla použita hashovací funkce. Níže provedeným příkazem je provedeno využití slovníku ke generování hashovaných hesel a jejich porovnávání se vstupem, který je v souboru *md5.txt*. Parametrem `format` uvádíme použitou hashovací funkci.

```
john --wordlist=dict.txt --format=raw-md5 userheshes.txt
```

Výsledek, tedy uživatelé a odtajněná hesla, je vidět vidět na obrázku 3.9. Tento

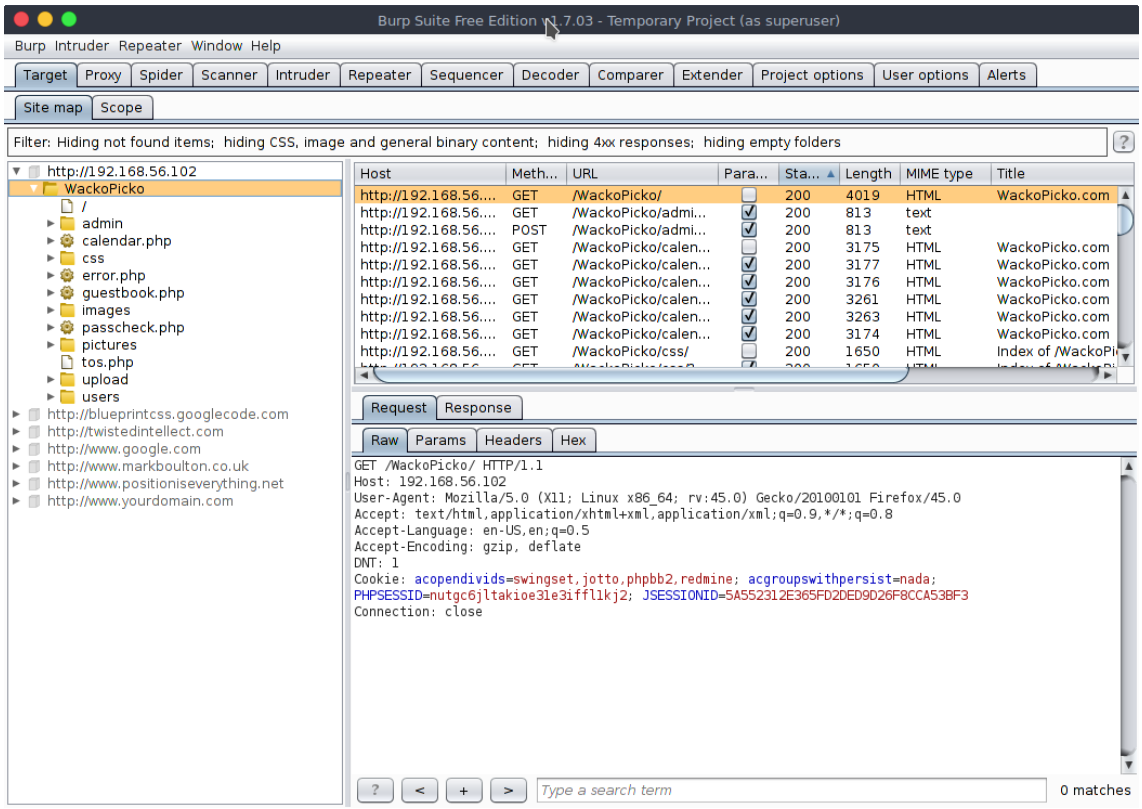
```
└─$ john --wordlist=dict.txt --format=raw-md5 md5.txt
Using default input encoding: UTF-8
Loaded 2 password hashes with no different salts (Raw-MD5 [MD5 128/128 AVX 4x3])
Press 'q' or Ctrl-C to abort, almost any other key for status
heslo321         (user2)
123heslo         (user)
2g 0:00:00:00 DONE (2017-04-08 13:24) 10.52g/s 10.52p/s 10.52c/s 21.05C/s 123heslo..heslo321
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Obr. 3.9: John the Ripper – použití slovníku při lámání hesel

způsob lámání hesel ukrytých pomocí hashovací funkce, v kombinaci s údaji z fáze získávání informací, kdy je použijeme k vytvoření slovníku, může být rychlou a nenáročnou cestou k získání přístupu k uživatelským účtům, která mají vytvořena nevhodná a slabá hesla. Další cestou je generování hashů v souladu s danými pravidly a jejich porovnávání se získanými daty. Vzhledem k výpočetní a časové náročnosti tohoto způsobu vzniklo třetí řešení, které představuje databáze hashů, takzvané *duhové tabulky (rainbow tables)*, které obsahující předgenerované hashe hesel. Jejich struktura je pak podřízena optimalizaci pro rychlé vyhledávání dle těchto hashů.

### 3.4.5 Burp Suite

Burp Suite je nástroj od společnosti PortSwigger [31]. Ve své bezplatné verzi poskytuje soubor nástrojů určených pro bezpečnostní testování webových aplikací. Obsahuje nástroje jako proxy server, spider, dekodér, fuzzer nebo také nástroj Sequence, který provádí analýzu nahodilosti generátoru sessionID na bitové a znakové úrovni. Jednou z jeho výhod je možnost rozšíření pomocí Burp Extender [32] API nebo BApp Store [33], který obsahu množství open-source pluginů.




Obr. 3.10: Burp Suite

Nástroj ve své komerční verzi obsahuje automatizovaný skener webových aplikací, který pokrývá všechny typy zranitelností z OWASP Top 10 [34]. Jedná se tedy o komplexní nástroj, který lze stejně dobře využít jak ve fázi skenování, tak ve fázi exploitace. Při použití je Burb Suite v prohlížeči nakonfigurován jako proxy na portu 8080. Po zadání vyšetřované adresy do prohlížeče se stránka zobrazí v záložce *Target* nástroje. Jednotlivé záložky jsou pojmenované podle funkce. Pro zjištění struktury aplikace pak vybereme požadovanou aplikaci v boxu *Site map* a po zmačknutí pravého tlačítka myší vybereme možnost *Spider this branch*. Po chvíli se zobrazí struktura aplikace zjištěná nástrojem.

## 3.5 Exploitace

V této části práce jsou popsány nástroje určené k přímému odhalování bezpečnostních zranitelností. První část se zabývá nástroji specializovanými k odhalovaní určeného typu zranitelností. Druhá část představuje komplexní nástroje jako jsou bezpečnostní frameworky a scannery.

### 3.5.1 OWASP Mantra

OWASP Mantra je bezpečnostní framework vyvíjený a poskytovaný komunitou OWASP. Jedná se o webový prohlížeč ve výchozí konfiguraci obohacený o širokou škálu bezpečnostních doplňků, rozšíření a nástrojů pro provádění bezpečnostních testů. Vyvíjen je od roku 2010, kdy byla zpřístupněna i první veřejná verze. V počátku byl framework přístupný pouze pod operačním systémem Windows, později však vznikl i pro ostatní operační systémy a jejich webové prohlížeče. V prvních verzích vycházel z webového prohlížeče Mozilla Firefox, v roce 2011 však byla představena verze vycházející z prohlížeče Chromium. Mantra byla přidána do výchozí konfigurace bezpečnostních distribucí BackTrack a Matriux. V současné době je framework dostupný ve výchozí konfiguraci distribucí Parrot Security a Kali Linux používaných v této práci.

Mezi funkcionality a doplňky, které Mantra obsahuje, patří množství bezpečnostních a proxy nástrojů, nástroje pro správu cookies a cache prohlížeče, FTP, SSH, REST a SQLite klienti nebo předdefinovaná sada záložek s odkazy na zdroje a portály, které se zabývají bezpečností. Výhodou je i dostupnost v přenosné verzi, kdy není nutná instalace. Nástroje jsou v Mantře podle účelu, ke kterému jsou určeny, rozděleny do několika kategorií na:

- nástroje pro získávání informací
- editory
- síťové utility
- různé
- audit aplikací
- enkodéry a dekodéry
- malwary
- anonymizace

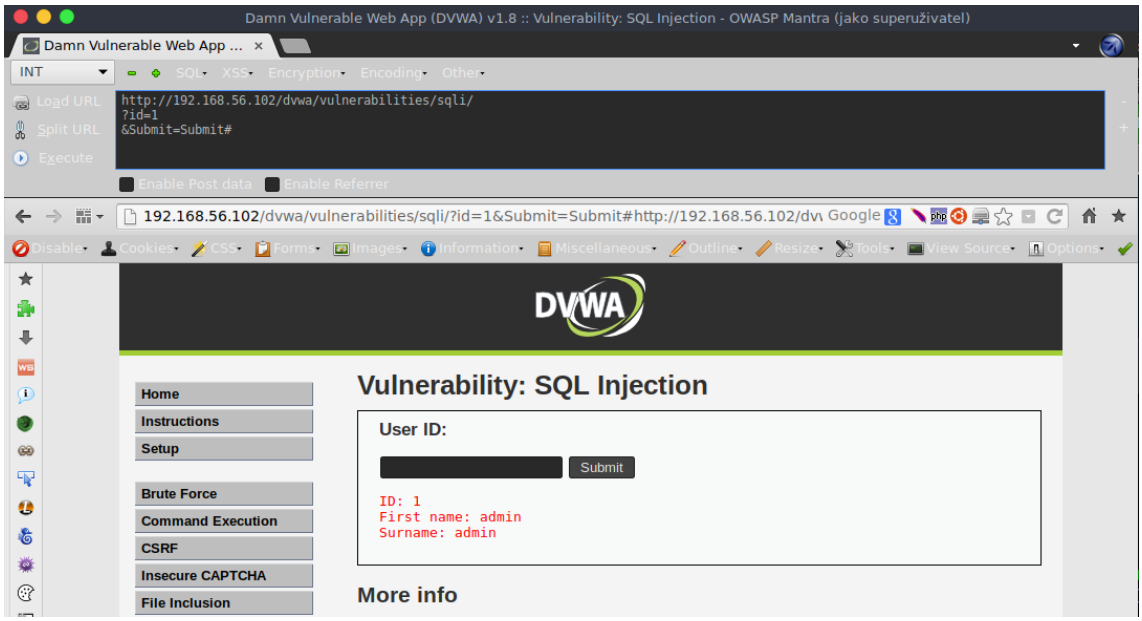
V této práci jsou představeny dva nástroje, které mohou najít uplatnění i mimo obor penetračního testování. Nástroje Hackbar a Temper Data jsou k dispozici ve frameworku Mantra, lze je však nainstalovat i samostatně jako doplňky prohlížeče.

## Hackbar

Hackbar je rozšíření webového prohlížeče Mozilla Firefox. Umožňuje rychlejší a efektivnější testování bezpečnostních zranitelností SQL injection a XSS. Použití tohoto rozšíření bude demonstrováno na aplikaci DVWA (Damn Vulnerable Web App), která je součástí souboru aplikací OWASP Broken Web Application Project.

Rozšíření má podobu adresního řádku obohaceného o možnosti formátování URL. To přináší usnadnění při testování výše zmíněných zranitelností, kdy se modifikuje příkaz nebo dotaz za účelem odeslání nedůvěryhodných dat do interpretu aplikace. V našem příkladu je ve zranitelné aplikaci použito vstupní pole, které při zadání a odeslání uživatelova ID vrací údaje o jménu a příjmení uživatele. Po odeslání uživatelova ID vidíme v adresní řádku URL požadavku. Tu můžeme jednoduše modifikovat následujícím způsobem k otestování SQL injection a XSS.

1. původní URL:
   `http://test.cz/?id=1&Submit=Submit#`
2. test SQL injection:
   `http://test.cz/?id=1'&Submit=Submit#`
3. test XSS:
   `http://test.cz/?id=1<script>alert(XSS)</script>&Submit=Submit#`

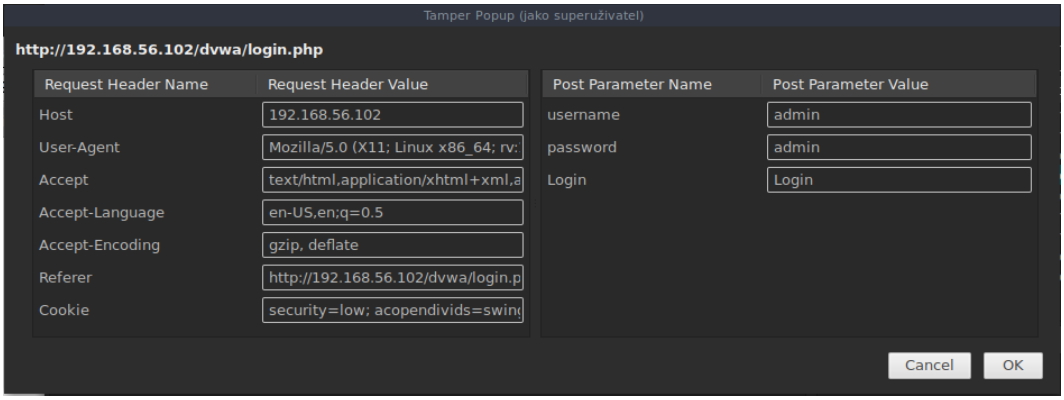



Obr. 3.11: Nástroj Hackbar

### Tamper Data

Druhým zástupcem užitečných rozšíření prohlížeče je doplněk Tamper Data. Jedná se o doplněk, který umožňuje zachytit a především modifikovat requesty, ještě před jejich odesláním na server. To je využitelné především jestliže má testovaná aplikace JavaScriptové validace, skryté formulářové prvky, nebo POST parametry, které nejdou modifikovat přímo v adresním řádku.

Použití demonstrujme opět na aplikaci DVWA, kdy budeme odchytávat request pro přihlášení uživatele do aplikace. Po aktivaci doplňku a odeslání přihlašovacího formuláře se zobrazí upozornění na sledování requestu, kdy po potvrzení odchycení requestu zobrazí Temper Data jeho obsah, který je vidět na na obrázku 3.12. Zde máme možnost odeslat požadavek s modifikovanými parametry nebo například i zamezit jeho odeslání.



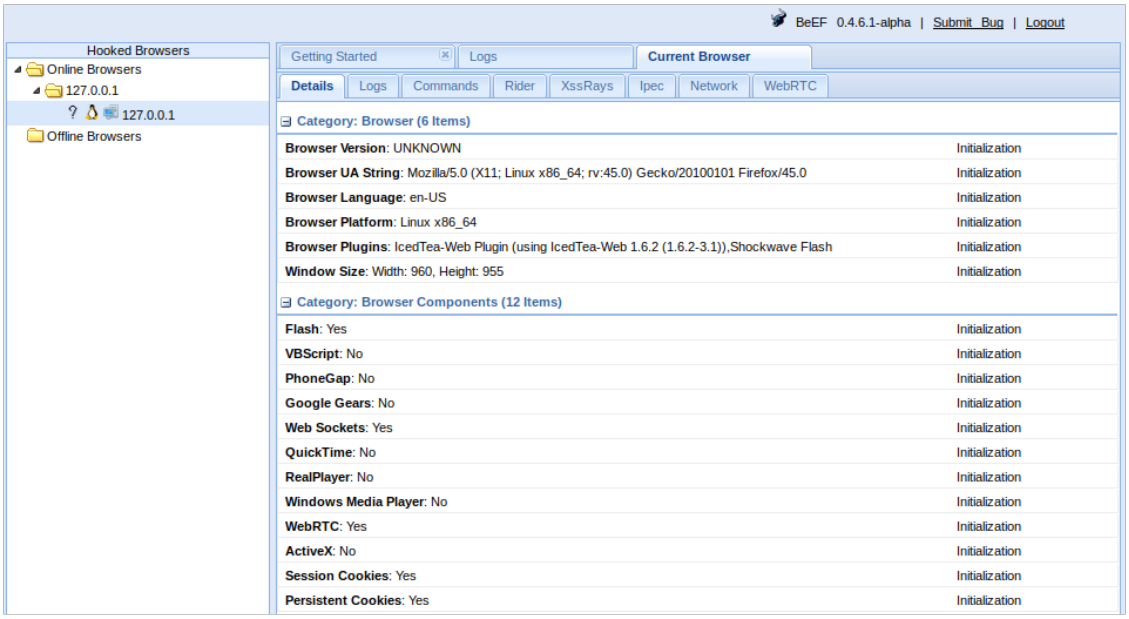

Obr. 3.12: Nástroj Temper Data

### 3.5.2 The BeEF

BeEF neboli The Browser Exploitation Framework je nástoj určený pro penetrační testování se zaměřením na webové prohlížeče. Ke svému fungování využívá zranitelnost Cross-Site Scripting XSS, díky které je docíleno, aby ve webovém prohlížeči oběti vykonal skript hook.js. To může v praxi umožnit zranitelná aplikace, ale i kupříkladu otevření neznámého odkazu v emailu. Samotný soubor hook.js obsahuje jak kompletní knihovnu jQuery, tak především vrátka pro testování známých útoků na konkrétní prohlížeče, zranitelnosti Flash Playeru, Java appletů, dále chyby v nejrůznějších programech aj.

Po spuštění serveru a navštívení lokální adresy `http://127.0.0.1:3000/ui/panel` lze po přihlášení výchozím loginem a heslem „beef“ vidět webové rozhraní viz. ob-

rázek 3.13. Toto základní prostředí umožňuje získání velkého množství informací o připojených počítačích. Pomocí nástroje lze pak například monitorovat aktivitu uživatele nebo získat jeho cookies. [36]




Obr. 3.13: Nástroj BeEF

### 3.5.3 SQLMap

SQLMap - Automatic SQL Injection and database takeover tool je nástroj přístupný přes příkazový řádek a dostupný ve výchozí konfiguraci obou zde uváděných linuxových distribucí. Je napsán v jazyce Python a určený k odhalovaní a exploitaci zranitelností typu SQL Injection. Nástroj je dostupný zdarma na `http://sqlmap.org/`. S tímto nástrojem je například možné přes nezabezpečený parametr získat veškeré informace o databázovém serveru, vytvořit uživatele (je-li to možné) nebo získat struktury tabulek celé databáze. SQLMap automatizuje odhalování děr typu SQL Injection. Ty se potom snaží zneužít a získat úplnou kontrolu nad databázemi a hostujícími servery.

Použití je demonstrováno na aplikaci OWASP Mutillidae II ze souboru aplikací OWASP Broken Web Application Project. Jedná se o soubor PHP skriptů implementujících zranitelnosti z OWASP Top 10.

V aplikaci je vybrána zranitelnost SQL Injection v implementaci umožňující extrahovat data z databáze. K získání informací o databázi s využitím známé zranitelnosti vede následující příkaz.

```
sqlmap -u "http://owasp.bwap/mutillidae/index.php?page=user-
info.php&username=user&password=password&user-info-php-submit-
button=View+Account+Details" -p username --current-user --current-db
```

Přepínačem `-u` kopírujeme URL jako hodnotu a přepínačem `-p` zaměří pozornost nástroje v k vyšetření zranitelnosti v parametru `username`. Zbytek příkazu požaduje informaci o aktuálním uživateli, pod kterým je aplikace přihlášena do databáze a o jménu databáze. Po výpisu sady testovacích dotazů do konzole je nástrojem představen výsledek který je vidět na obrázku 3.14.

```
---
[21:29:32] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL >= 5.0
[21:29:32] [INFO] fetching current user
current user:    'mutillidae@%'
[21:29:34] [INFO] fetching current database
current database:    'nowasp'
[21:29:36] [INFO] fetched data logged to text files under '/home/josephus/.sqlmap/output/192.168.56.102'

[*] shutting down at 21:29:36
```

Obr. 3.14: Výstup SQLMap příklad č. 1

Z tohoto výstupu byla získána informace o jménu databáze - `nowasp`, kterou v následujícím příkazu využijeme k získání informací o názvu tabulek databáze.

```
sqlmap -u "http://owasp.bwap/mutillidae/index.php?page=user-
info.php&username=test&password=test&user-info-php-submit-
button=View+Account+Details" -p username -D nowasp --tables
```

Během okamžiku dostaneme informaci o jménech databázových tabulek v podobě zobrazené na obrázku 3.15.

```
Database: nowasp
[12 tables]
+---------------------------+
| accounts                  |
| balloon_tips              |
| blogs_table               |
| captured_data             |
| credit_cards              |
| help_texts                |
| hitlog                    |
| level_1_help_include_files |
| page_help                 |
| page_hints                |
| pen_test_tools            |
| youtubevideos             |
+---------------------------+
```

Obr. 3.15: Výstup SQLMap příklad č.2

### 3.5.4 Metasploit framework

Metasploit framework neboli MSF je pravděpodobně nejuniverzálnější volně dostupný penetrační framework, který lze najít. V současné době je vyvíjen společností Rapid7. Framework začal vyvíjet bezpečnostní specialista a programátor H. D. Moore v roce 2003. Nejdříve byl framework vyvíjen v jazyce Perl, později se vývojářská komunita rozhodla k přepisu frameworku do jazyka Ruby.

Klíčovou předností frameworku je jeho modularita. Ta umožňuje to, že framework obsahuje rozličná nastavení a exploity vztahující se k různým druhům softwaru. Framework je díky své velikosti rozsáhlé téma a jeho zvládnutím se zabývají samostatné publikace. [37] V této práci bude představena pouze základní funkcionalita ve vztahu k testování webových aplikací. Obecně lze říci, že díky své modularitě je Metasploit framework vhodný k testování všech zranitelností zahrnutých v OWASP Top 10.

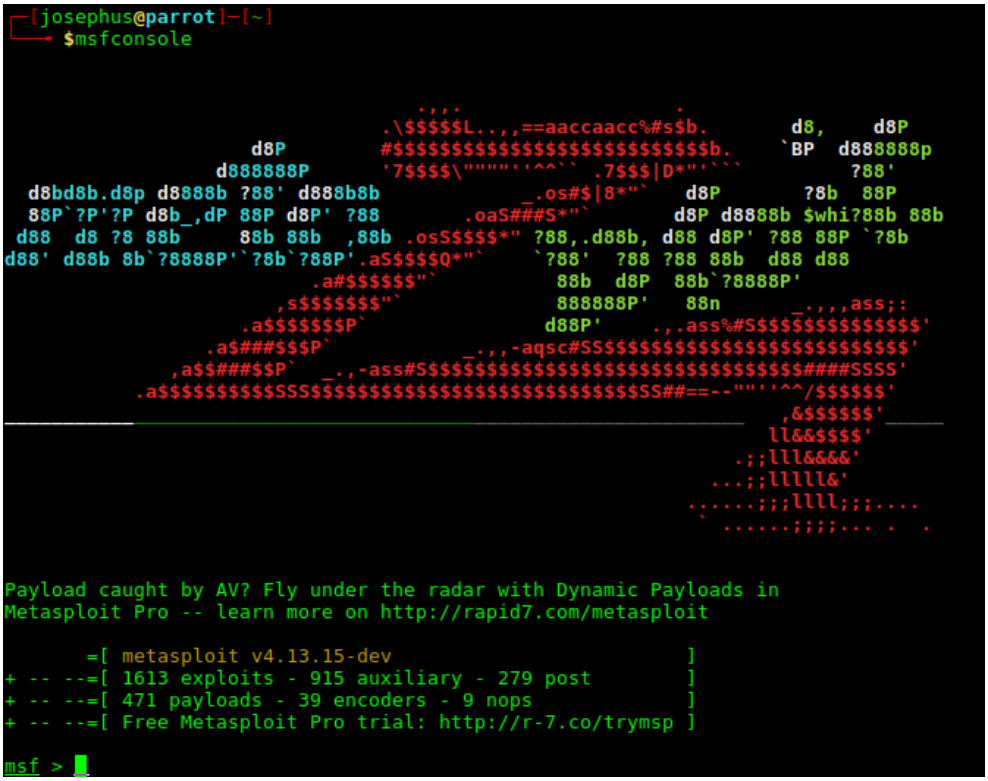
#### Moduly

- **Doplňkové moduly (auxiliary)** – jedná se o výchozí skripty frameworku, které provádějí nejrůznější typy penetračních testů a scanů. Hlavním záměrem těchto skriptů je poskytnou penetračnímu testerovi sadu testů, které mu umožní efektivně otestovat vybraný cíl. Pro příklad lze uvést modul s názvem `mysql_enum`, který poskytuje základní úroveň získání informací o vyšetřovaném MySQL databázovém serveru.
- **Moduly exploitů (exploit)** – tyto moduly obsahují různé skripty, které obsahují kód umožňující využití známé zranitelnosti. Výsledkem provedení těchto skriptů je většinou přístup testera do testovaného systému. Skripty pokrývají široké spektrum zranitelností a to od známých zranitelností softwaru, webových serverů po zranitelnosti operačních systémů jako Windows nebo Android. Pro příklad lze uvést modul `ms08_067_ netapi`, který využívá známé zranitelnosti ve stanicích s operačním systémem Microsoft Windows.
- **Moduly enkodérů (encoder)** – nebo také kódovací moduly představují užitečnou sadu modulů, které mají za cíl upravit případné exploity a payloady do takové podoby, aby nebyly snadno detekovatelné antivirovými programy a IPS/IDPS systémy. Efektivní použití těchto modulů vyžaduje určitou předchozí zkušenost a obeznámenost testera s danou problematikou
- **Moduly payloadů (payload)** – jak název napovídá, tyto moduly obsahují payloady. Jedná se o taková data, která odkazují na konkrétní část škodlivého softwaru, jenž vykonává škodlivou či nebezpečnou činnost. Škodlivým softwarem jsou myšleny trojské koně, viry či červi. [38] Jsou vykonány poté, co se dostanou do systému za pomocí exploitů.

- **Ostatní moduly (other)** – jedná o moduly, které se vymykají ostatním kategoriím. Jmenovitě se jedná o modul Nops a post exploitační modul.

### Msfconsole

Je interaktivní konzole Metasploit frameworku. Je využívána k výběru a provádění jednotlivých modulů a dalších konzolových operací, které jsou spojeny s prováděním testů pomocí frameworku. Konzoli lze spustit z příkazového řádku systému po zadání příkazu `msfconsole`, kdy je zobrazena úvodní obrazovka konzole. Jedna z možných podob je viditelná na obrázku 3.16.

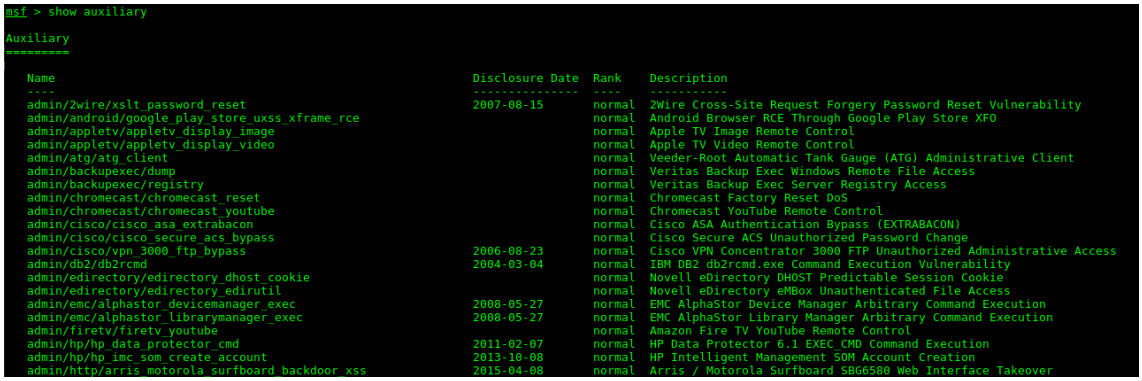



Obr. 3.16: Msfconsole

Konzole umožňuje prohlížet jednotlivé moduly podle jejich názvu v souladu s předchozím popisem (anglické názvy v závorkách) pomocí příkazu `show` a názvu modulu. V případě použití příkazu:

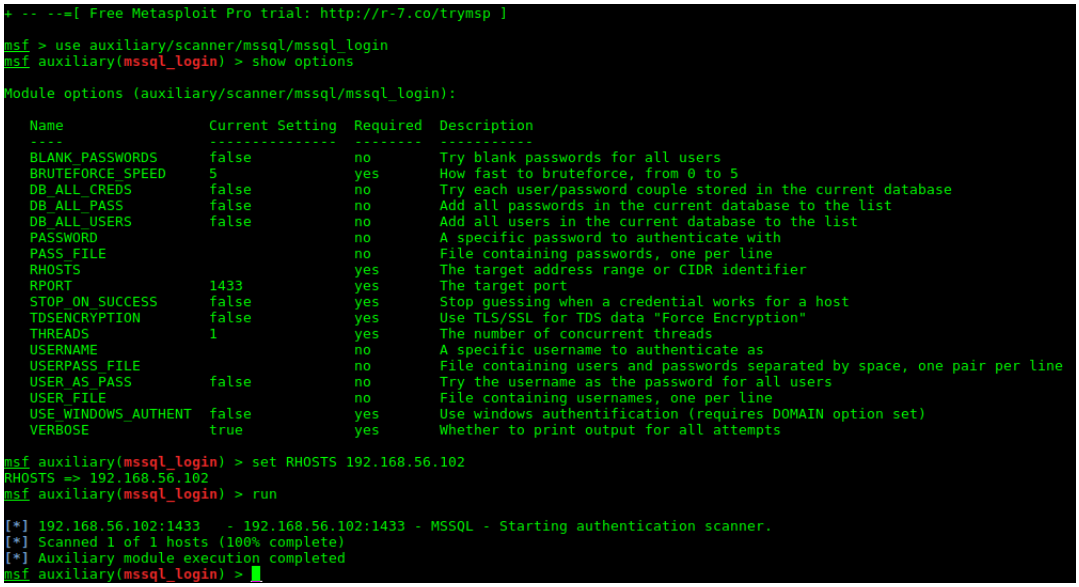
```
show auxiliary
```

dostaneme vyčerpávající seznam modulů, jehož část je znázorněná na obrázku 3.17. Moduly jsou dále rozděleny do podkategorií dle způsobu nebo fáze útoku.




Obr. 3.17: Msfconsole – přehled modulů

Lze zde najít kategorie jako `scanner, fuzzers, sqli, spoof`. Samotné použití některého modulu je provedeno příkazem `use`, který jen následován cestou k modulu, která končí jeho názvem. Po zapnutí modulu lze ke zjištění konfigurace modulu použít příkaz `show options` a pro nastavení parametrů pak příkaz `set`, doplněný mezerou názvem modifikované proměnné a mezerou následovanou hodnotou, která bude do proměnné vložena. Spuštění modulu se pak provede příkazem `run`. Celý proces je ilustrován na obrázku 3.18, kde jsou vypsána nastavení, modifikována testovaná URL a posléze otestována.




Obr. 3.18: Msfconsole – exekuce modulu

### Metasploit WMAP

WMAP otvírá závěrečnou část této kapitoly, která je věnována automatickým scannerům. Jedná se o testovací skript uvnitř Metasploit frameworku, který byl vytvořen jako odnož nástroje SQLMap. Využívá soubor modulů frameworku k provedení komplexní testovací sady na vybraném cíli.

Nástoj je spustitelný z msfconsole příkazem `load wmap`, po němž se zobrazí úvodní logo a informační obrazovka o úspěšném načtení nástroje. Ke svému fungování využívá nástroj PostgreSQL databázi. Pro správný běh nástroje je proto nutná její inicializace příkazem `msfdb init` s příslušnými právy. V případě úspěšného spuštění nástroje je pak možné vyvolat pomocné menu příkazem `help`.

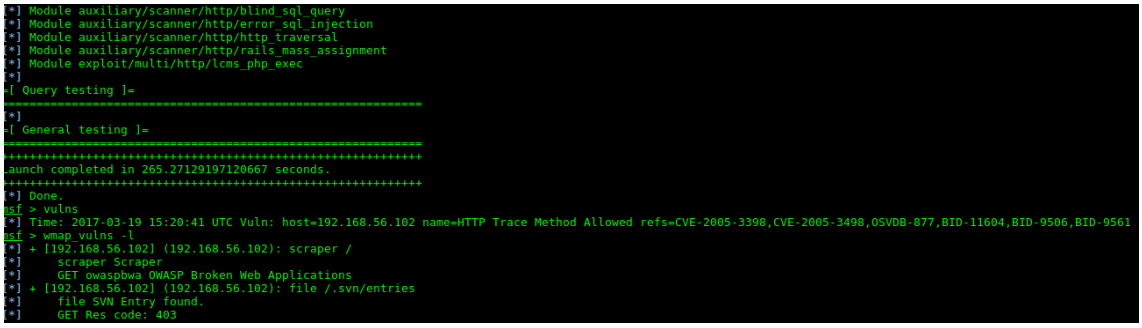
Pro samotné testovaní je nutné přidání testovacího cíle. To je možné příkazem:

```
wmap_sites -a protocol://host:port
```

Po přidání URL v požadovaném formátu (např.: `http://127.0.0.1:8080`). Pro kontrolu můžeme u příkazu použít přepínač `-l` pro zobrazení seznamu přidaných URL. Před během samotných testů je nutné přidat adresu do seznamu cílů. To je provedeno následujícím příkazem:

```
wmap_targets -t protocol://host:port
```

Kde můžeme použít formát `protocol://host:port` nebo za přepínač zadat číslo, pod kterým byla adresa zaregistrována do seznamu adres. Kontrolu cílů můžeme provést opět přepínačem `-l`. Kontrolu modulů, kterými se budou provádět testy, provedeme příkazem `wmap_run -t`. Následné spuštění testů je provedeno příkazem `wmap_run -e`. Výsledky provedených testů zobrazíme s pomocí příkazů `vulns` a `wmap_vulns`. Část výstupu je zobrazena na obrázku 3.19




Obr. 3.19: Automatický scanner WMAP

### 3.5.5 Nikto

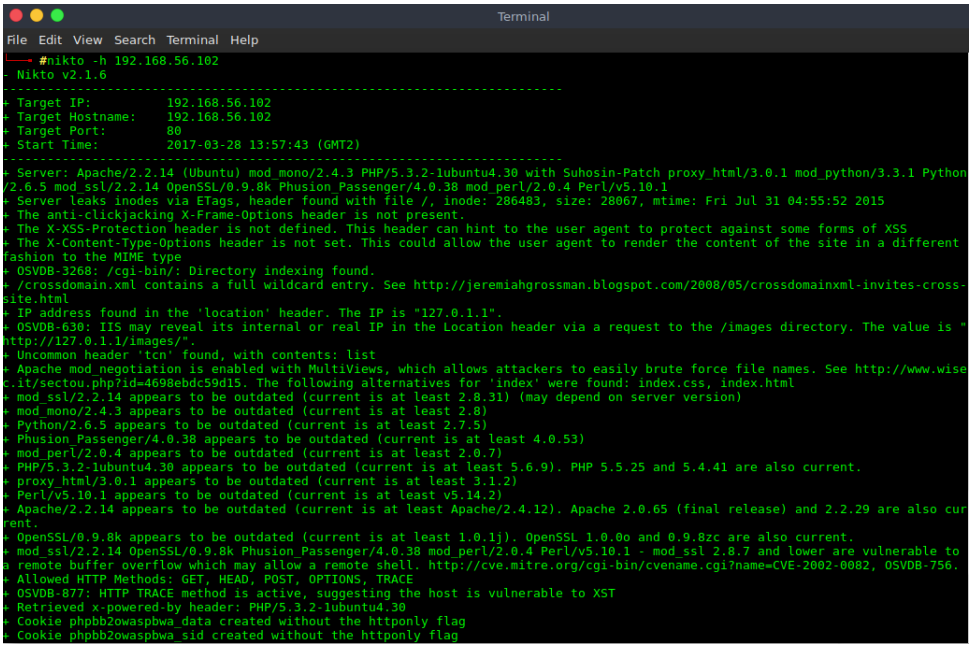
Nikto Web Scanner je název bezpečnostního nástroje, který zjišťuje, zda se na webovém serveru nevyskytují nebezpečné CGI skripty, zastaralý software a jiné problémy. Zároveň provádí obecné i specializované testy.

Podle údajů na domovské stránce projektu je schopný odhalit 6400 potenciálně nebezpečných souborů/CGI skriptů, 1200 potenciálně nebezpečných neaktualizovaných serverů na 270 specifických chyb spjatých s některými verzemi serverů. Scanner je schopný identifikovat instalované servery a zjistit případné konfigurační chyby serverů. Nikto je stále vyvíjený produkt, tudíž je udržován aktuální ve vztahu k nově objeveným zranitelnostem. Najít ho lze ve výchozím nastavení distribucí ParrotSec i Kali. Více informací lze najít na stránkách projektu[5].

Použití můžeme demonstrovat zadáním příkazu:

```
nikto -h protocol://host:port
```

Ve výchozím nastavení je použit port 80. Na obrázku 3.20 je vidět výsledek skenu IP adresy 192.168.56.102, kde je lokálně spuštěn soubor aplikací OWASP BWA.




Obr. 3.20: Automatický scanner Nikto

Výsledkem je obsáhlý report s popisem nalezených zranitelností a případně URL adresami s odkazem na dokumentaci zranitelnosti.

[5] https://cirt.net/Nikto2

### 3.5.6 Wapiti

Wapiti je automatický skener vyvinutý Nicolasem Surribasem. Poslední aktualizace skeneru proběhla v roce 2013 a je šířený pod licencí GNU GPL. Skener disponuje standardní funkcionalitou, ale na rozdíl od jiných nástrojů nedisponuje grafickým uživatelským rozhraním. Skener podporuje operační systémy Linux, Windows a OS X. [35] Nástroj je napsán v programovací jazyce Python. Použití je demonstrováno na příkazu:

```
$ wapiti http://mojestranka.cz -n 10 -b folder -u -v 1 -f html
        -o /tmp/report
```

Použití nastavení v příkazu:

- `-n`: limit vyšetřovaných adres (10) k zamezení nekonečných smyček
- `-b`: rámec hledání, vymezuje na sledování odkazů ze zadané domény
- `-u`: nastavení barvy zobrazení výstupu
- `-v`: nastavení úrovně výpisu – 1 pro vše
- `-g`: typ výstupu ve kterém bude report – html
- `-o`: adresář pro výstup

Wapiti rozšiřuje značné množství modulů, které jsou schopné odhalit různé zranitelnosti injektací (CR/LF, XSS, SQL), zálohy skriptů nacházejících se na serveru, zranitelnosti na vykonání příkazů, souborové zranitelnosti nebo umějí například využít databázi nástroje Nikto k zjištění potenciálně nebezpečných souborů. Ačkoliv Wapiti není v současné době aktualizován, stále se jedná o nástroj, který je schopen odhalit velké množství zranitelností čemuž napomáhá i to, že od roku 2013 se dle OWASP situace v oblasti bezpečnostních hrozeb, myšleno s ohledem na jejich výskyt v aplikacích, nijak zásadně nezměnila.

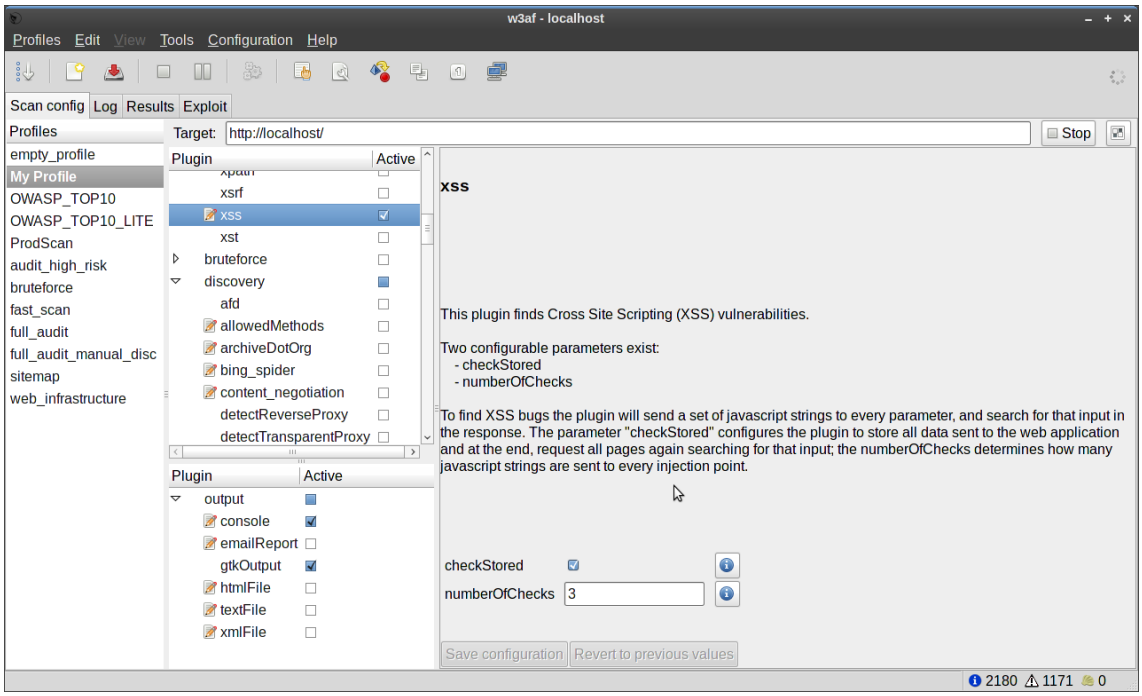
### 3.5.7 w3af

w3af neboli Web application attack and audit framework je otevřený bezpečnostní skener webových aplikací. Tento software nabízí skener zranitelností a exploitační nástroje pro webové aplikace. Získává informace o bezpečnostních zranitelnostech a poskytuje rady pro testování průniku.

Tento multiplatformní nástroj je k dispozici ve všech populárních operačních systémech jako Microsoft Windows, Linux, Mac OS X, FreeBSD a OpenBSD. Je napsaný v programovacím jazyce Python.

w3af dokáže identifikovat mnoho zranitelností webových aplikací a používá k tomu více než 130 pluginů. Po identifikaci zranitelností jako jsou například (Blind) SQL injection, vložení cizího (vzdáleného) souboru (v PHP), Cross-site scripting (XSS), nebo nezabezpečené nahrávání souborů, můžou být využity k získání různého druhu a úrovně přístupu ke vzdálenému systému. [40] Podle údajů uvedených na stránkách projektu[6] je scanner schopný identifikovat více jak 200 zranitelností.

Nástroj disponuje grafickým uživatelský rozhraním, které je znázorněno na obrázku 3.21. Zároveň je možné nástroj provozovat v konzolovém režimu spuštěním skriptu `w3af_console`.




Obr. 3.21: Automatický scanner w3af

Obě dvě uživatelská prostředí nabízejí stejné možnosti konfigurace a je pouze na osobních preferencích testera, kterou variantu zvolí. Nástroj sám o sobě je koncipován jako soubor pluginů, které mají své specializace k odhalovaní různých bezpečnostních slabin. Pluginy jsou sdružovány do profilů. Ve výchozím nastavení je dostupných několik profilů koncipovaných k různým účelů. Je zde profil pro *rychlý scan* nebo také profil, který prověřuje aplikaci na všechny zranitelnosti s OWASP Top 10. Pro širší možnosti konfigurace je k nástroji dostupný manuál, který se zabývá jak použitím nástroje z konzole, tak v jeho grafické podobě[7].

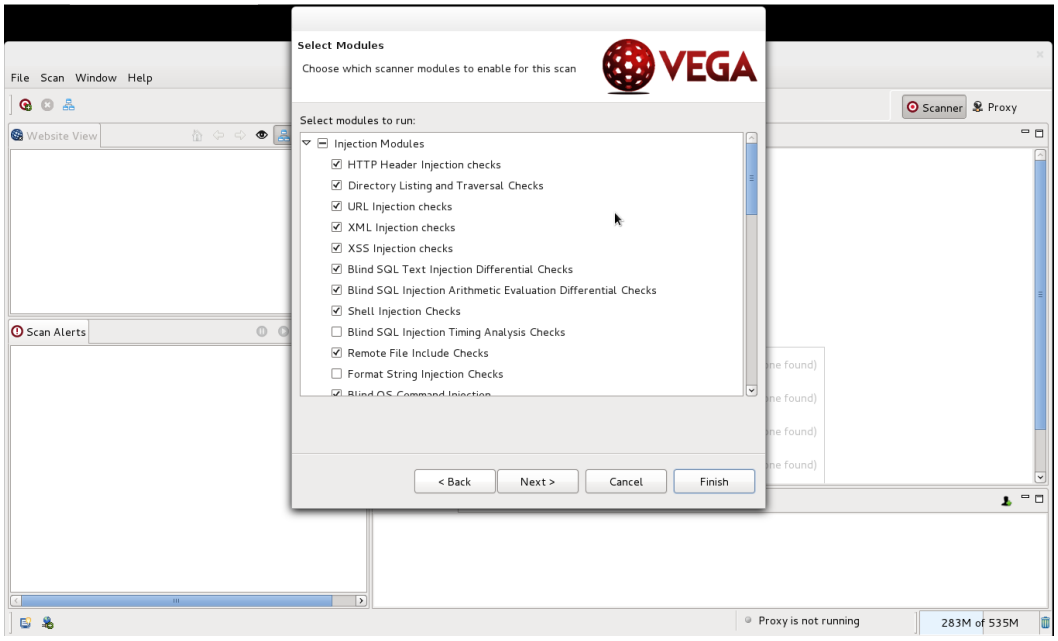
[6] `http://w3af.org/`
[7] `https://media.readthedocs.org/pdf/w3af/latest/w3af.pdf`

### 3.5.8 Vega scanner

Vega je volně dostupný open-source automatický webový scanner a zároveň platforma pro testování bezpečnosti webových aplikací. Vega je schopná odhalit zranitelnosti typu SQL Injection, Cross-Site Scripting (XSS), vystavování citlivých dat a další zranitelnosti z OWASP Top 10. Mezi další funkcionality patří možnost ověřit nastavení zabezpečení TLS/SSL a identifikace potenciálních problémů v tomto nastavení.

Je napsána v programovacím jazyce Java a díky tomu je dostupná pro Linux, OS X i operační systémy Windows. Disponuje také grafickým uživatelským rozhraním viz. obrázek 3.22.

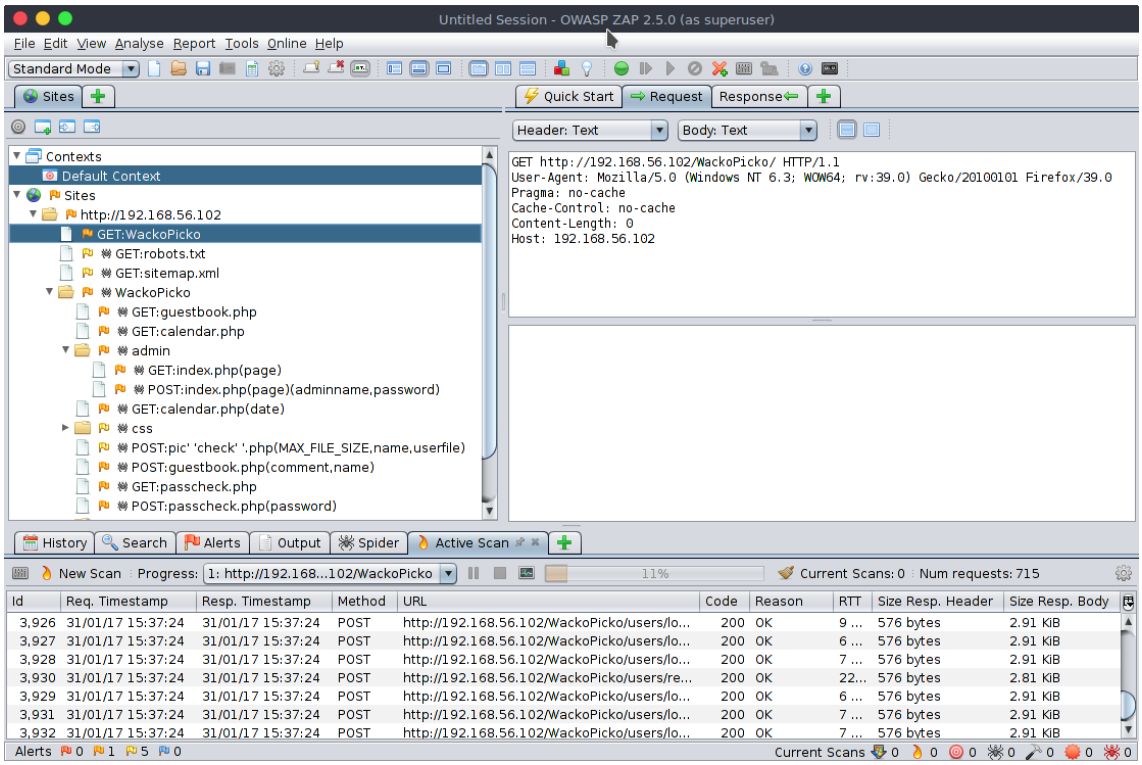
Scanner je modulární, kdy každý modul je určen k testování příslušné zranitelnosti. Moduly lze jednoduše vybírat pomocí checkboxů při konfiguraci nového scanu - auditu zadané adresy, který lze inicializovat červenou ikonou v levém horním rohu nástroje. Nástroj lze také využít jako proxy mezi aplikací a prohlížečem. To umožňuje monitorovat a posléze vyšetřit komunikaci proběhlou mezi prohlížečem a aplikací. Nástroj lze využít i jako proxy scanner, když může být nakonfigurován tak, aby vykonával testovací moduly během uživatelova pohybu testovanou aplikací. Tento přístup je ukázkou semiautomatizového typu testování, kdy je díky uživatelovi znalosti aplikace možné docílit maximálního pokrytí testy – testování se také nazývá user-driven.




Obr. 3.22: Automatický scanner Vega

### 3.5.9 OWASP ZAP

OWASP Zed Attack Proxy Project (ZAP) je projekt, na kterém se podílí množství odborníků z celého světa. Patří mezi nejpoužívanější open-source aplikace určené pro penetrační testování. [29] Do tohoto nástroje je integrováno množství nástrojů jako skener, fuzzer, spider, AJAX spider, proxy server, forced browser, nástroj pro vytváření reportů a další. Mezi jeho další výhody patří i jeho lehká integrace skrze REST API a jeho rozšiřitelnost pomocí Zap Extensions [30], což umožňuje použití pluginů nebo vytváření vlastních. Je tak vhodný pro testování všech zranitelností, které se nacházejí v OWASP Top 10. ZAP nabízí obdobné možnosti jako předchozí




Obr. 3.23: OWASP ZAP

scannery. Díky použitelnosti jako proxy jej lze využít i ve fázi skenování. Jedná se o aktivně vyvíjený projekt s rozsáhlou a podrobnou dokumentací.

Výsledky penetračních testů provedených nástrojem jsou dostupné ve formě reportu ve formátech *xml* nebo *html*. Jejich export je možný z *Hlavní menu →Report →Generate HTML/XML Report*. Ze záložky *Report* je možné exportovat i další výsledkové přehledy. Obsahem vygenerovaných reportů je přehled zjištěných zranitelností a hodnocení jejich závažnosti v souladu s technikou OWASP Risk Rating Methodology [12], která je podrobněji popsána v následující podkapitole.

## OWASP Risk Rating Methodology

Jedná se o metodologii vyvinutou organizací OWASP k účelu zhodnocení rizik zranitelností. Dle této metodiky lze míru rizika obecně stanovit na základě vzorce:

$$\text{RIZIKO} = \text{PRAVDĚPODOBNOST} \cdot \text{DOPAD} \text{ [12]}$$

V první fázi se identifikuje riziko. Získávají se informace o zranitelnosti, útoku, útočníkovi a dopadech testů. Následně jsou provedeny odhady pravděpodobnosti, dopadu a nakonec určení závažnosti rizika.

### Odhad pravděpodobnosti

Při identifikaci bezpečnostního rizika je zjišťována pravděpodobnost zneužití. Ta je hodnocena na stupnici 0 - 9, kde 0 představuje žádnou závažnost a 9 maximální závažnost. Hodnotí se ve dvou kategoriích v následujících krocích:

1. Pravděpodobnost útoku ze strany útočníka:
    - úroveň dovedností
    - motiv
    - příležitost
    - velikost skupiny útočníků
2. Pravděpodobnost ze strany zranitelnosti:
    - náročnost zjištění zranitelnosti
    - náročnost využití zranitelnosti
    - povědomí o zranitelnosti
    - detekce průniku

Výpočet průměrného hodnocení se pak provede součtem hodnocení v jednotlivých krocích vyděleným počtem kroků (8).

### Odhad dopadu

Následně se stanovuje jaké technické a business dopady by mohly být způsobeny. Opět se hodnotí na škále 0 - 9 a zohledňují se:

1. Technické faktory dopadu:
    - ztráta důvěrných dat
    - ztráta integrity
    - ztráta dostupnosti služeb
    - ztráta odpovědnosti

2. Business faktory:
    - finanční škody
    - poškození dobrého jména
    - poškození obchodních dohod
    - narušení ochrany soukromí

Průměrné hodnocení je provedeno stejně jako v případě předchozího kroku.

### Závažnost rizika

Zhodnocené faktory jsou pak podle bodového ohodnocení klasifikovány do úrovní závažnosti dle tabulky 3.2. Tyto úrovně lze stanovit jako výsledné pro *pravděpodob-*

| Průměr | Úroveň |
|---|---|
| 6 až 9 | Vysoká |
| 3 až <6 | Běžné |
| 0 až <3 | Vzácné |

Tab. 3.2: Úrovně rizika

*nost* a *dopad*, ale je také možné použít dílčí faktory dle potřeby. Jejich zprůměrováním dostaneme výslednou úroveň rizika. To je provedeno i v případě hodnocení úrovní rizik v generovaných reportech, kdy jsou zhodnoceny pouze známé faktory vztahující se ke zranitelnostem. Ty jsou vidět v tabulce 3.3.

| Vektor útoku | Rozšíření slabiny | Zranitelnost slabiny | Technický dopad |
|---|---|---|---|
| Snadný | Rozsáhlé | Snadná | Vážný |
| Průměrný | Běžné | Průměrná | Střední |
| Obtížný | Vzácné | Obtížná | Malý |

Tab. 3.3: Klasifikace závažnosti zranitelnosti

Obě tabulky dodržují i barevné zatřídění, které je použito v reportech nástroje OWASP ZAP. Podrobnější popis i s příklady bodového hodnocení lze najít v dokumentaci metodologie. [12] Metodologie není vyhrazena pouze pro nástroj OWASP ZAP, ale je aplikovatelná na hodnocení výsledků jakýkoliv bezpečnostních testů.

# 4 VYUŽITÍ NÁSTROJE OWASP ZAP PŘI PRŮBĚŽNÉ INTEGRACI

Tato kapitola se zabývá procesem průběžné integrace a automatických testů, jež je v dnešní době běžnou součástí vývoje nových aplikací. Osvětluje základní pojmy a představuje nástroje, jejichž použití je demonstrováno na praktickém příkladu v závěru kapitoly.

## 4.1 Správa zdrojového kódu

Při vývoji rozsáhlých projektů v týmu vývojářů vzniká potřeba správy zdrojových kódů. V rámci této správy je nutné, aby všichni členové týmu měli okamžitý přístup k aktuální verzi kódu, aby každý vývojář měl možnost nový kód zpřístupnit ostatním. Zároveň je potřeba oddělit stabilní verze od vývojových. Mít možnost sledovat historii vývoje a zamezit ztrátám kódu nebo jeho získání neoprávněnými osobami. Z těchto a dalších důvodů vznikly systémy pro správu verzí kódu (version control systems - VCS), které si kladou za cíl tyto potřeby řešit.

V dnešní době se na trhu nachází řada nástrojů, které tyto problémy řeší. Společnou mají schopnost ukládat zdrojové soubory do projektů a jejich metadata do tzv. repozitářů. Z hlediska práce s těmito repozitáři lze nástoje rozdělit do dvou hlavních skupin: [43]

- **Centralizované VCS** – používá se pouze jeden centrální repozitář. Ukládání kódu do systému znamená automatické sdílení s ostatními uživateli. Centrální repozitář je umístěn na server a uživatelé pracují pouze s lokálními kopiemi. Během práce musí uživatelé komunikovat se serverem, a jestliže je nedostupný nelze provádět žádné operace.
- **Distribuované VCS (DVCS)** – používá se centrální repozitář, který slouží k synchronizaci a integraci práce mezi uživateli. Každý uživatel má lokální kopii repozitáře se kterou pracuje a synchronizace vzniká jako požadavek. Oproti předchozímu řešení zde mohou při synchronizaci vznikat různé konflikty, kdy vývojáři editují stejnou část kódu. Ty se musí řešit. Jedná se však o řešení, které implementuje převážná část moderních verzovacích nástrojů.

### 4.1.1 Nástroje pro správu zdrojových kódů

Systémy pro centrální správu verzí zdrojových kódů jsou:

- **CVS** – je multiplatformní systém sloužící ke správě verzí projektu umožňující konzistentní a distribuovanou práci více lidí na jednom projektu současně. Kvůli mnohým problémům CVS (například nemožnosti záznamu přesunu souboru, nemožnosti verzování symbolických odkazů, nedostatečné podpoře UTF, neatomickým komitům, velmi problematickému větvení nebo špatné podpoře binárních souborů) vznikla řada následovníků a náhrad. V dnešní době již není vyvíjen a jeho použití v praxi je minimální.
- **Apache Subversion (SVN)** – multiplatformní systém, který z CVS vychází a nahrazuje ho. Zachovává styl a obdobný způsob práce jako nástroj CVS, ale odstraňuje jeho hlavní nedostatky (kupříkladu zmíněný záznam přesunu souborů). Použití systému v praxi ustupuje, stále se s ním však lze setkat především u starších projektů, které jsou udržovány, ale již na nich neprobíhá další vývoj.
- **Mercurial** – komerční systém vyvinutý firmou Perforce Software. Nástroj vznikal současně s nástojem Git v roce 2005. Svými vlastnostmi a funkcemi je s ním téměř totožný, avšak na rozdíl od Gitu je navržen jako monolitický software se striktněji předdefinovanými funkcemi a postupy použití. Mezi výhody nástroje je často zmiňována strmá křivka učení. Předmětem kritiky bývá práce s větvemi a malá podpora na straně hostingu.
- **Git** – open source distribuovaný systém vytvořeným L. Torvaldsem původně vyvíjen pro verzování linuxového jádra. V současnosti se jedná o pravděpodobně nejpoužívanější verzovací nástroj. To s sebou přináší výhody v podobě množství zdarma dostupných hostingů repozitářů, či široké podpory ze strany dalších vývojových nástrojů. Mezi základní principy patří rychlost, jednoduchý design, silná podpora pro nelineární vývoj (libovolné množství paralelních větví) a schopnost zvládat rozsáhlé projekty. Z těchto důvodů byl tento verzovací nástroj použit i v praktické části této práce.

### 4.1.2 Git – hostování repozitářů

Při použití verzovacího nástroje Git je třeba vyřešit umístění origin repozitáře. Tedy umístění hlavního repozitáře, ze kterého a do kterého se synchronizují verzované soubory projektu. Po tomto úkolu je nutné vyřešit otázky, jako jsou způsob komunikace s klienty, správu a řízení přístupu uživatelů a další úkoly spojené s bezpečností. Na internetu lze najít širokou paletu služeb a produktů, které řeší tyto otázky a nabízejí i doplňkové služby. Při výběru řešení je třeba dále brát na zřetel, zda chci vytvořit soukromý nebo veřejný repozitář, možnosti uchovávání dat na serveru, finanční náročnost řešení atd.

Při verzování projektu se nabízejí dvě hlavní cesty:

#### Hosting jako služba

Toto je nejrychlejší a nejjednodušší řešení, které bylo použito i v této práci. Stačí najít poskytovatele této služby, vytvořit účet, nakonfigurovat prostředí, založit projekt a vše je připraveno k použití. Existuje mnoho poskytovatelů, mezi nejznámější patří *Github*, *Bitbucket, Gitlab, SourceForge*, ale existuje i mnoho dalších. Šíře poskytovaných služeb je rozdílná u placených a neplacených účtů. Další otázkou je, jestli uživatel chce mít svá data na cizím serveru.

#### Vlastní hosting

Další možností je zajištění vlastního hostingu provozovaného na vlastním serveru. Výhoda tohoto řešení je kontrola nad vlastními daty. Připravené prostředí k instalaci nabízí například Gitlab, ale lze najít i další. Instalaci lze pak provést dle dokumentace vybraného produktu.

## 4.2 Průběžná integrace

Kontinuální nebo také průběžná integrace je souhrn vývojářských nástrojů, metodik a postupů používaných při vývoji softwaru v týmech. Při dodržování principů průběžné integrace se snaží vývojář postupně integrovat svoji práci s prací ostatních vývojářů v týmu. Slouží mimo jiné k urychlení nalezení nedostatků a chyb u softwarových projektů ve fázi vývoje. Pro spojení těchto metod a nástrojů se používají Integrační Servery. Každé přidání nebo změna kódu je sestavena automatickou kompilací (build), kdy jsou zároveň spuštěny kontrolní testy a případné chyby tak mohou být identifikovány a dohledány v co nejkratším čase. V různých zdrojích lze najít i alternativní názvy jako postupná integrace. V anglickém jazyce se ustálil název Continuous Integration (CI). [41]

Mezi hlavní výhody zavedení a používání průběžné integrace patří:

- rychlé nalezení chyb ve zdrojovém kódu
- automatická kontrola kódu
- přehled všech členů týmu o stavu buildu
- přehledné verzování jednotlivých verzí a rychlý přístup k verzím
- ušetření času při kompilaci a vydání nové verze
- ušetření testovacích kapacit při využití automatického testování [42]

### 4.2.1 Sestavení aplikace

Hlavním procesem z hlediska funkčnosti aplikace je její sestavení. Převod zdrojových kódů do funkční aplikace je mnohdy složitý proces, jehož součástí je kompilace zdrojových kódů, přesun souborů, nahrávání schémat do databáze a další. Celý tento proces je možné automatizovat. Díky tomu se eliminují zbytečné chyby, které by mohly být zaneseny člověkem a podstatně se proces zrychlí. V současnosti jsou dostupné mnohé nástroje k tomuto určené. Jednotlivé nástroje jsou určeny pro různé programovací jazyky. Mezi nejznámější lze zmínit nástroj *make* používaný v *nixových systémech, nástroje *Ant, Maven, Gradle* určené pro jazyk Java nebo nástroje *Grunt a Gulp*, které jsou určené k sestavení front-endů.

### 4.2.2 Testování aplikace

Provádění testů je jednou z technik, které pomáhají zajistit kvalitu aplikace a její udržení během budoucích úprav. Jedná se o opakovanou, rutinní činnost a proto je v dnešní době vyvíjena snaha tuto činnost maximálně automatizovat. Existují různé druhy testů a k nim různé druhy nástrojů a frameworků, které jsou k nim vyvíjeny. Mezi základní testy, které jsou nejčastěji prováděny patří: [44]

- **Unit testy** – jedná se o testy základních jednotek kódu na úrovni funkcí a metod.
- **Integrační testy** – jedná se o testy větších jednotek kódu a jejich integrace. Příkladem může být testování připojení k databázi, náhodné generátory nebo připojení k síti.
- **Regresní testy** – využívají při opětovném testování funkcí a vlastností aplikace. Jejich smyslem je ověření, že provedené změny či implementace nových vlastností v aplikaci neměly žádný vliv na stávající funkce a vlastnosti. Tedy především na oblasti, které zůstaly v programovém kódu nezměněny.
- **UI testy** – testuje se grafické uživatelské rozhraní. Jedná se o testování různých zobrazení, přechodů mezi obrazovkami atd. V dnešní době se i tyto testy velice dobře automatizují a k tomu existuje množství nástrojů.

Mezi další testy mohou patřit kupříkladu zátěžové testy nebo **testy bezpečnostní**. V praxi jsou bezpečnostní testy mnohdy opomíjeny, čemuž napovídají i časté bezpečnostní nedostatky aplikací a webových stránek. Právě jimi se zabývá tato práce a na příkladu, který je uveden v závěru této části, demonstruje možnost jejich začlenění do procesu CI.

### 4.2.3 Nástroje pro průběžnou integraci

Postupná integrace (CI) je dnes velmi používanou technikou. Díky tomu je na trhu dostupná celá řada nástrojů pro její zajištění. Dostupná jsou řešení open source, ale také placené. Nástroje podporují široké spektrum programovací jazyků mezi které patří například *Java, C#, PHP, Javascript*. Jsou většinou ovládány přes webové grafické rozhraní, případně pomocí konfiguračních souborů. Mezi často používané patří placený produkt Microsoftu Team Foundnation Server (TFS), open source řešení Gitlab CI nebo nástroj Jenkins. Poslední zmíněný byl použit v této práci a proto je níže popsán podrobněji.

#### Jenkins

Jedná se pravděpodobně o nejpoužívanější open source řešení pro CI, psané v jazyce Java. Je vhodný pro týmy různých velikostí a je možné jej použít pro projekty psané v různých jazycích jako např. *Java, C#, Ruby, Python, Groovy, Grails, PHP, C/C++* a další. Mezi hlavní výhody tohoto prostředí patří jednoduchost použití, široká podpora programovacích jazyků a velká uživatelská základna, která sebou přináší kvalitní dokumentaci. Jenkins nabízí široké možnosti rozšíření a úprav chování, což zprostředkovává řada dostupných pluginů. V neposlední řadě zde existuje možnost napsat si plugin vlastní.

Konfiguraci lze provádět z webového rozhraní, kde je také dostupný repozitář pluginů. Nástroj je distribuován pouze se základní sadou pluginů, proto je v počáteční fázi konfigurace nutná doinstalace pluginů v závislosti na potřebách projektu.

##### Struktura projektu

Práce s nástrojem Jenkins probíhá s využitím následujících komponent:

- **Jenkins projekt** – v rámci projektu se konfiguruje nastavení Jenkins serveru. Jako součást instance serveru je možné mít více projektů, lze tedy spravovat průběžnou integraci více aplikací a vývojových projektů nebo lze mít například Jenkins projekty pro různá integrační prostředí. V nástroji existuje několik druhů projektů. Jimi jsou:
  - **Freestyle projekt** – je základní typ projektu s kompletní možností konfigurace určený pro potřeby řízení průběžné integrace.
  - **External job** – tento projekt je určený k použití Jenkinse jako Dashboardu, kdy je proces buildu řízen jiným systémem, který předá do Jenkinse pouze data k zobrazení.

  - **Multi-configuration projekt** – je určen pro projekty s více konfiguracemi prostředí a podmínek. Nabízí stejnou funkcionalitu jako Freestyle projekt rozšířenou o takzvanou konfigurační matici. Tu si lze představit jako n-rozměrnou matici, kde každá dimenze udává proměnnou. Velikost této dimenze udává počet hodnot, kterých bude proměnná nabývat, hodnoty v daném rozměru pak konkrétní hodnoty proměnné. Jenkins spustí buildy pro všechny kombinace proměnných. Buildy mohou být spouštěny paralelně nebo sériově dle potřeb projektu. Typické užití je např. spuštění stejného projektu na různých slave uzlech, s jinou verzí překladače či knihoven nebo různými proměnnými pro build skript. [46]
- **Pohledy (View)** – projekty lze pro přehlednost slučovat do vyšších jednotek, kterými jsou pohledy.
- **Build proces** – jedná se o hlavní část Jenkins projektu. Je zde definováno propojení s VCS, nastavení testovacího prostředí, způsob sestavení aplikace, spuštění, vyhodnocení a zobrazení testů nebo spouštění dalších pomocných nástrojů a napojení na externí systémy. Build proces jako celek se dělí na několik dílčích kroků, kterými jsou:
  1. **Pre-build** – jedná se o nepovinnou fázi, kterou lze doinstalovat pomocí pluginu. Většinou v ní probíhá dodatečné konfigurace projektu, která však může být uskutečněna i ve fázi následující.
  2. **Build** – je hlavní fáze, jejíž součástí je sestavení, překlad, testování, ale i vyhodnocení a analýzy projektu. Nejdůležitější fází je spuštění build skriptu, který bývá dělen na několik dalších fází. S pomocí pluginů lze doinstalovat podporu široké škály nejpoužívanějších nástrojů, mezi které patří *Ant, Maven, Gradle, Rake, Visual Build*, ale i další. Pro ukázkový scénář prezentovaný v této práci byl zvolen nástroj *Maven*.
  3. **Post-build** – tato fáze slouží jako doplňující fáze buildu, kdy jsou interpretovány a zobrazeny jeho výsledky. Dále je zde možné nadefinovat například akce, které navazují na úspěšné provedení build fáze. Může být proveden merge větví ve verzovacím systému nebo například nasazení aplikace na aplikační server.

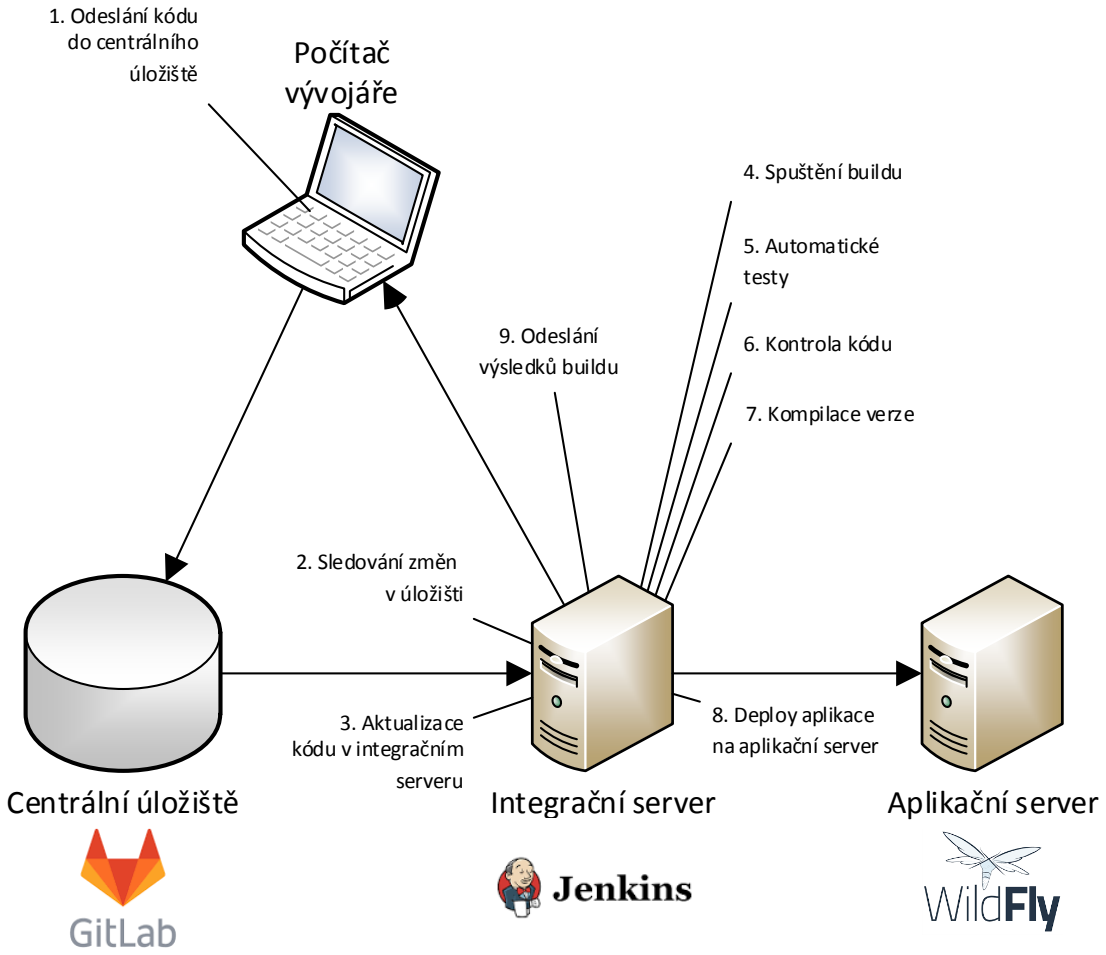
### Další funkce nástroje Jenkins

Mezi další další důležité funkcionality, které má význam zmínit, patří distribuovaný build. Ten se v praxi často používá pro multi-configuration projekty, kdy je potřeba distribuovat zátěž a zajistit různá prostředí. Pak také *Rest API*, kterým nástroj disponuje. S jeho využitím je možné například spouštět buildy, získávat jejich výsledky nebo také na něj zasílat data z jiných systémů.

## 4.3 Průběžná integrace a bezpečnostní testy příklad použití

V následující kapitole je v jednotlivých krocích popsána konfigurace projektu, který pak může být dále provozován v souladu s principy průběžné integrace. Součástí této konfigurace je také ukázka využití nástroje OWASP ZAP k provádění automatizovaných bezpečnostních testů aplikace. Ačkoliv je využita pouze demonstrativní ukázka aplikace na které není prováděn aktivní vývoj, je projekt nakonfigurován tak, jako kdyby zde vývoj probíhal.

Použité nástroje a jednotlivé kroky procesu průběžné integrace jsou přehledně znázorněny na obrázku 4.1



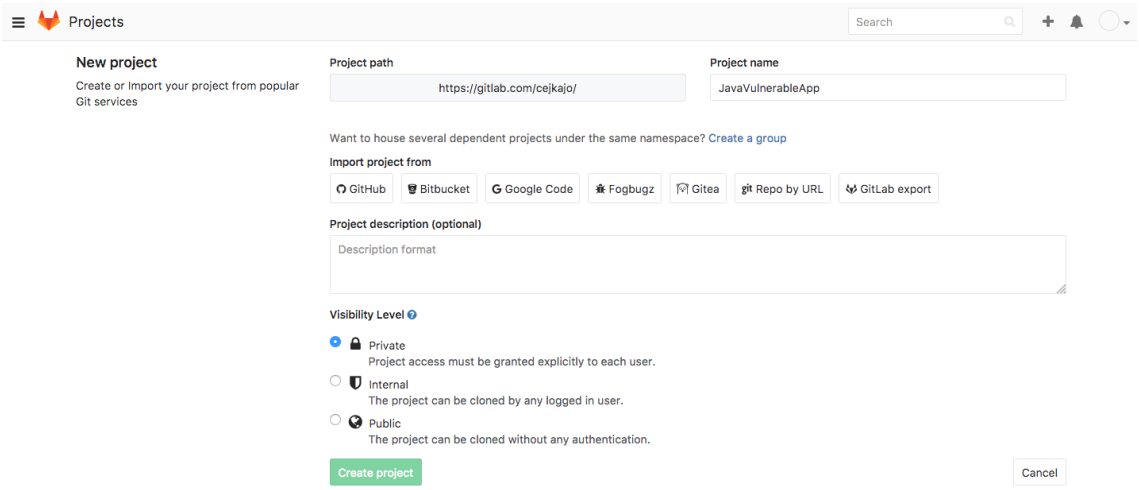

Obr. 4.1: Životní cyklus průběžné integrace

Jednotlivé nástroje jsou, až na hosting repozitáře, provozovány lokálně. V případě provozu jednotlivých nástrojů na oddělených serverech je však konfigurace obdobná,

pouze s rozdílným nastavením příslušných cest. Kapitola se nezabývá popisem instalace jednotlivých nástrojů. Způsobů jak nainstalovat nástroje do různých systémů je mnoho. Všechny způsoby jsou dle autorova názoru dobře zdokumentovány na webových stránkách jednotlivých nástrojů.

## Konfigurace Gitlab hostingu

Jak bylo zmíněno v kapitolách výše, je Gitlab[1] jednou z možností hostingu Git repozitářů. V základním nastavení je hosting provozován zdarma. Po registraci pomocí emailu lze přistoupit k vytvoření projektu. Na obrázku 4.2 je zobrazena obrazovka se základním nastavením. Pro vytvoření projektu je nutné vyplnit název projektu a úroveň ze které je projekt přístupný, kdy lze vybrat z možností přístupnosti jakýmkoliv uživatelem, pouze přihlášeným nebo řízení přístupu pro konkrétní uživatele.



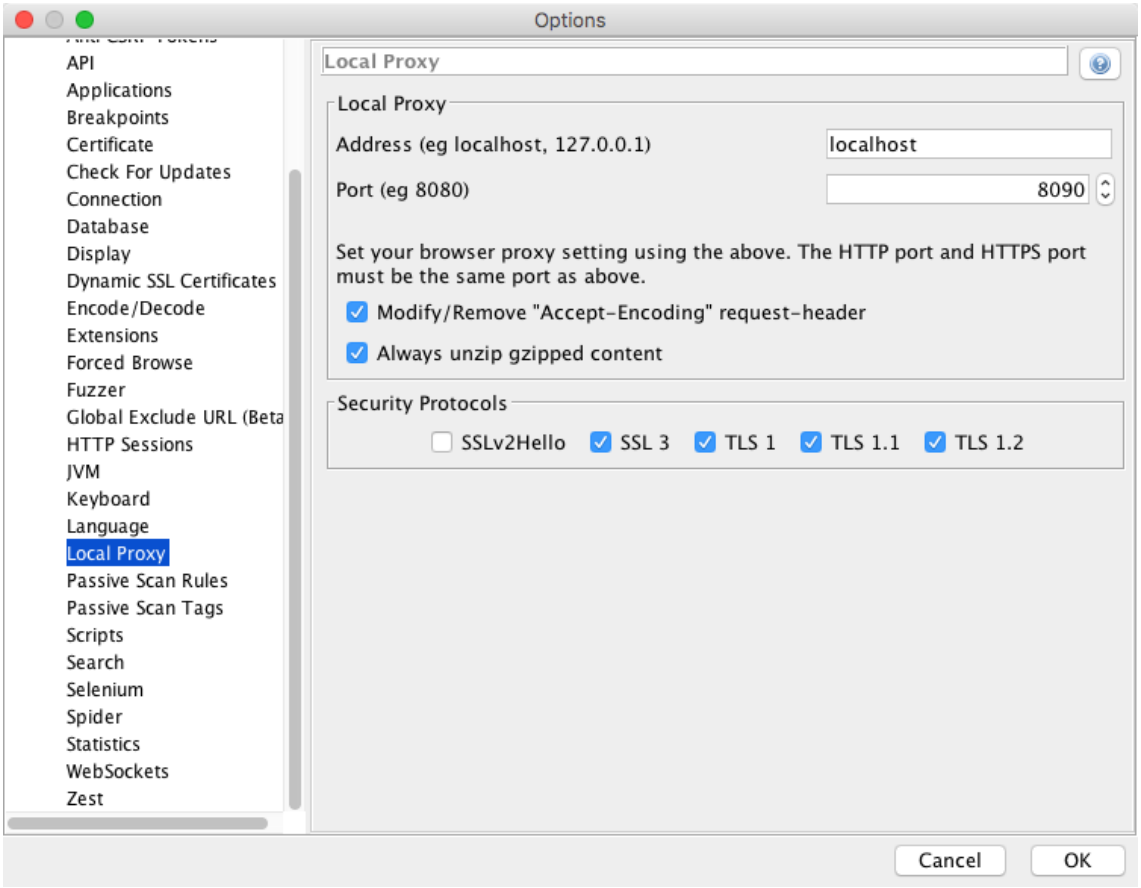

Obr. 4.2: Založení repozitáře v Gitlabu

Po vyplnění a potvrzení obrazovky je uživatel přesměrován na obrazovku, kde je jednoduše popsán způsob vytvoření lokálního repozitáře pomocí příkazové řádky. To bylo v tomto případě provedeno a následně byly do repozitáře přidány zdrojové soubory aplikace a synchronizovány do úložiště v Gitlabu. Zároveň byla k větvi *master* přidána vývojová větev *develop* v souladu s metodikou GitFlow [45]. Toto nastavení je pro účely práce dostačující a dál již nebude měněno.

[1] `https://about.gitlab.com/`

## Konfigurace nástoje OWASP ZAP

Pro správný průběh testů je na straně nástroje potřebná konfigurace. Prvním krokem je konfigurace proxy nástroje. Ta je ve výchozím stavu provozována na adrese `http://localhost:8080`, která je výchozí i pro přístup do grafického rozhraní nástroje Jenkins. Změnu lze provést v *Preferences →Local Proxy*, jak je znázorněno na obrázku 4.3. Na tuto adresu je pak nutné nasměrovat připojení k internetu výchozího prohlížeče. Pro Mozilla Firefox je dostupné na cestě *Předvolby →Rozšířené →Síť →Konfigurovat připojení aplikace Firefox k internetu →Ruční konfigurace proxy serverů*. Druhým závěrečným krokem v konfiguraci nástroje je založení nové session.




Obr. 4.3: ZAP změna default proxy

To je provedeno z hlavního menu *File →New Session*, kdy se zobrazí okno, na které je v tomto případě vhodné vybrat možnost *Yes, I want to persist this session but I want to specify the name location*. Tato volba umožní změnit výchozí místo pro ukládání session souborů. Jedná se o soubory ve formátu `*.session*`. Následně je nutné jejich uložení do workspace projektu na integračním serveru Jenkins, tak aby byly dále přístupné při konfiguraci Jenkins serveru.

## Konfigurace nástroje Jenkins[2]

Po přihlášení do Jenkinse je nutná instalace doplňujících pluginů, které jsou potřeba pro spuštění nástroje ZAP. Konfigurace je přístupná z *Úvodní obrazovky →Administrace →Spravovat pluginy →Dostupné*. Zde jsou vybrány a nainstalovány následující pluginy:

- EnvInject Plugin
- Summary Display Plugin
- HTML Publisher Plugin
- Zap Plugin

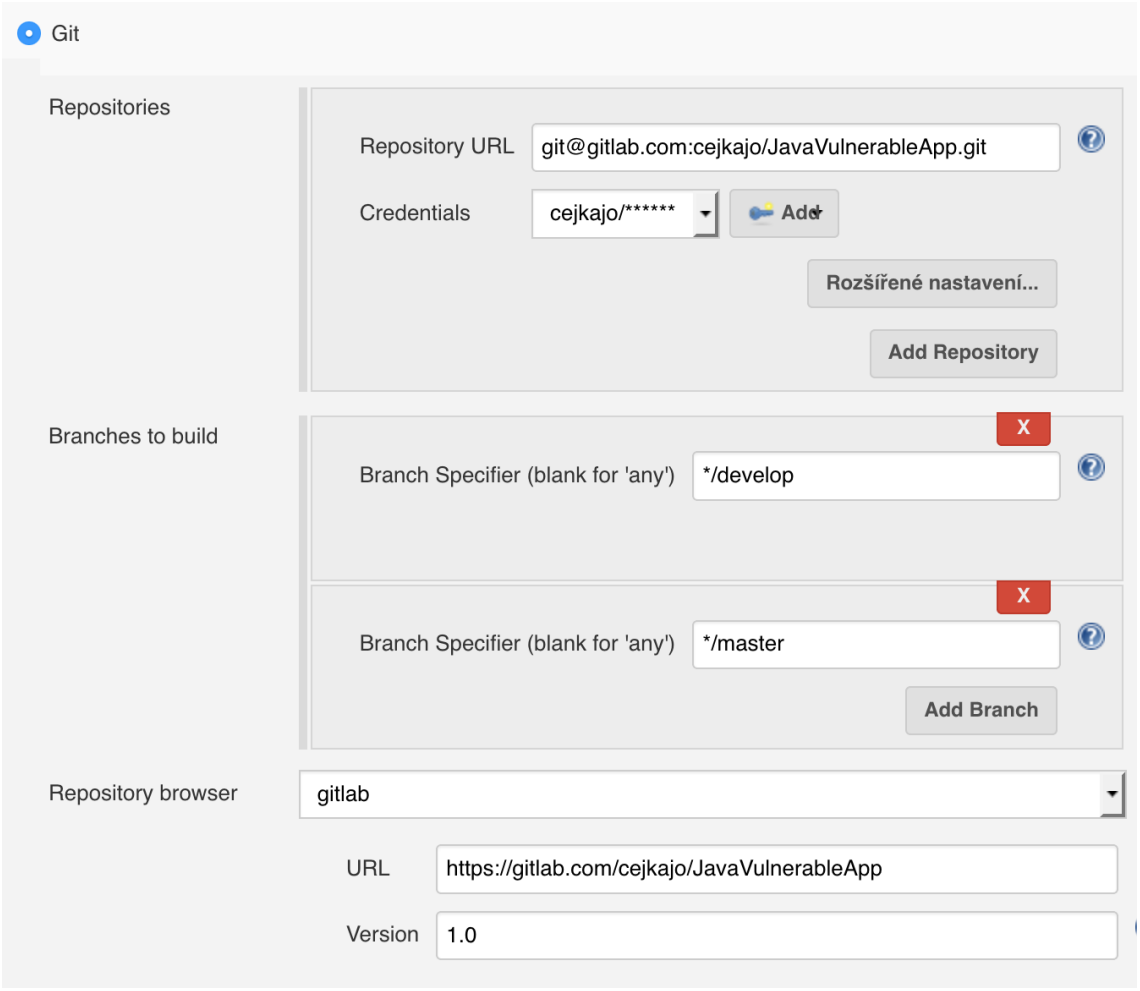
K samotnému buildu jsou potřebné i další pluginy, například *Git plugin* nebo *Maven plugin*, ty jsou však přítomny ve výchozí konfiguraci a není nutné je doinstalovávat.

Dalším krokem je založení projektu. To je provedeno z *Úvodní obrazovky →Nové →Freestyle project* s napsáním jména v horní části obrazovky. Tím je založen projekt a může být přistoupeno k jeho konfiguraci.

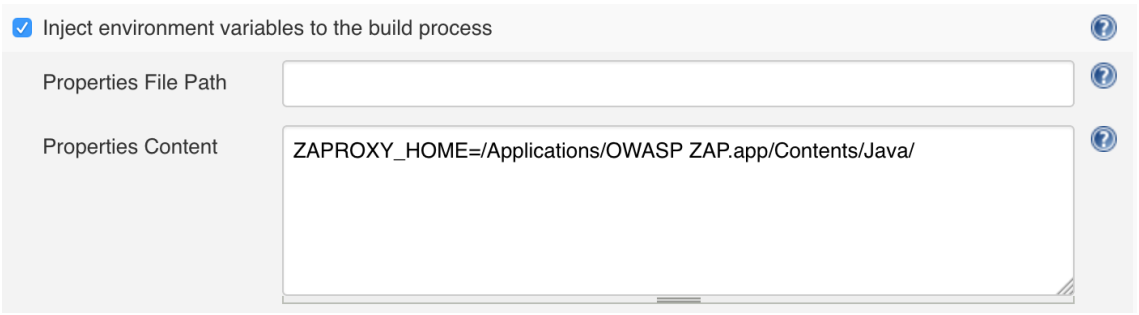
Po zobrazení detailu projektu kliknutím na jeho název je v levé části obrazovky dostupné menu projektu. Zde k jeho konfiguraci vybereme možnost *Nastavit*. Nastavení je rozděleno na jednotlivé části - *General, Source Code Management, Build Trigger, Build Enviroment, Build, Post-build Actions*. Důležitá nastavení v jednotlivých částech jsou popsána níže:

- ***General*** – zde je vyplněno pouze jméno aplikace, další je ponecháno beze změny
- ***Source Code Management*** – v této části jsou nastaveny údaje o hostingu repozitáře a přístupové údaje. Konkrétní nastavení ukázkového projektu je znázorněno na obrázku 4.4
- ***Build Triggers*** – zde je řízeno spuštění buildu. V našem případě je zaškrtnuto *Build when a change is pushed to GitLab.* Znamené to, že build je spuštěn při změně v repozitáři. Zároveň je možné vyvolat build ručně z postranního menu.
- ***Build Enviroment*** – v této části je možné nastavit různé proměnné prostředí případně spustit skripty. Pro správný běh nástroje OWASP ZAP je nutné zaškrtnou volbu *Inject environment variables to the build process* a zde pak do pole *Properties Content* doplnit proměnnou `ZAPROXY_HOME`, která míří do domovského adresáře nástroje ZAP. To je učiněno dle obrázku 4.5.

[2] `https://jenkins.io/`

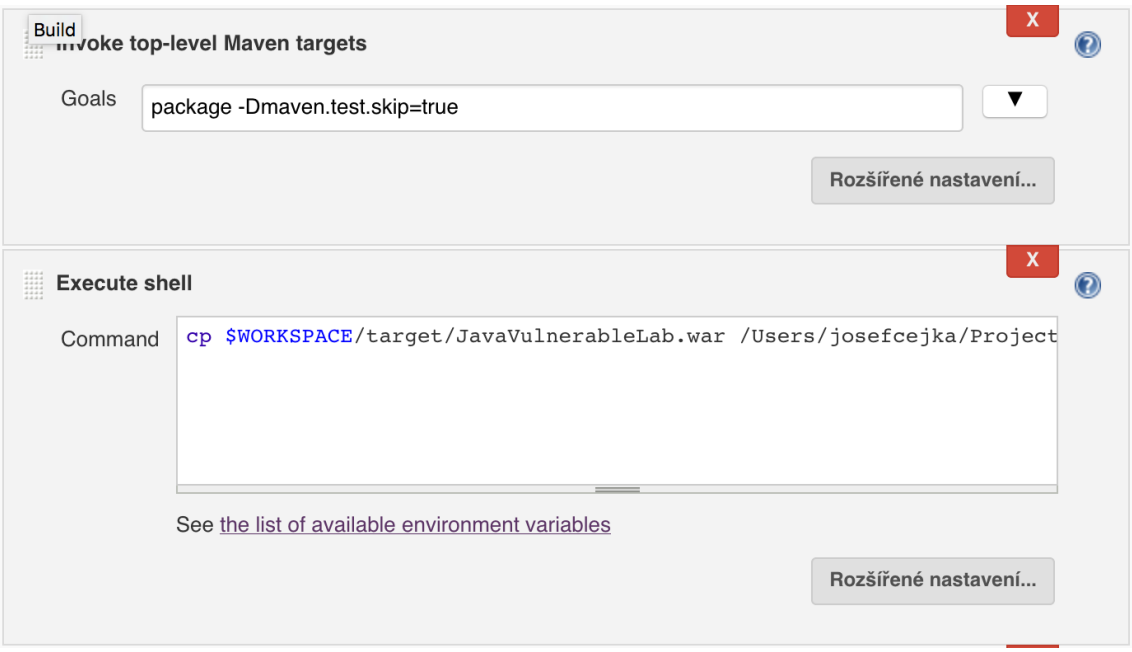



Obr. 4.4: Jenkins nastavení projektu – Source Code Management



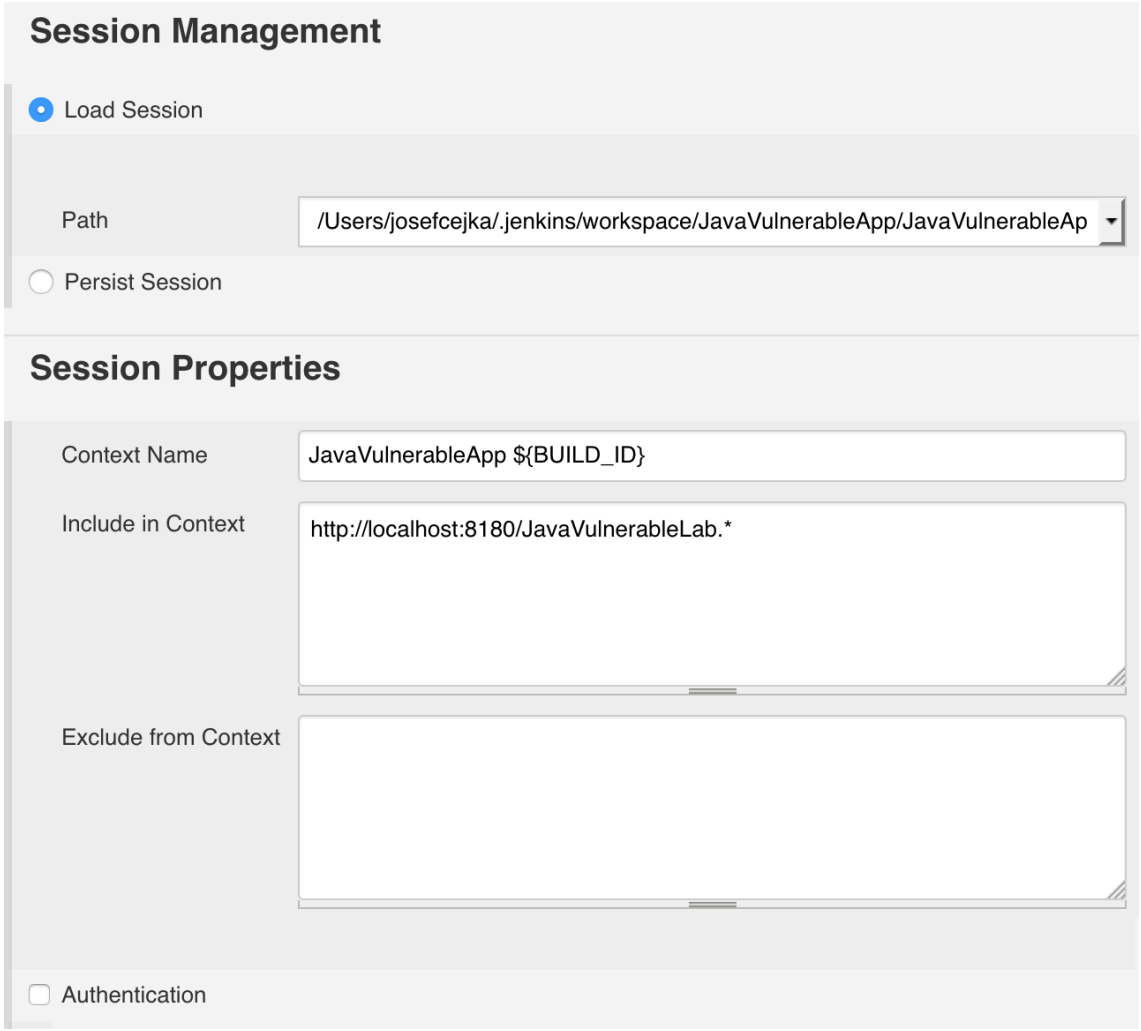

Obr. 4.5: Jenkins nastavení projektu – Build Enviroment

- ***Build*** – Ve výchozím nastavení se v sekci nachází pouze tlačítko s popiskem *Add build step*. Jedná se o rolovací menu, které nabízí možnost přidání buildovacího kroku. V tomto případě je využita možnost *Invoke top-level Maven targets*, *Execute shell* a *Execute ZAP*. V prvním případě sestavíme aplikaci buildovacím nástrojem *Maven*, druhý je použití k umístění sestavené aplikace na běžící server. Nastavení je vidět na obrázku 4.6.




Obr. 4.6: Jenkins nastavení projektu – Build steps – Maven – Shell

Dále dochází ke konfiguraci kroku, kdy budou spuštěny bezpečnostní testy. To je provedeno s pomocí *Zap pluginu*, díky kterému lze jako další build step vybrat *Execute ZAP*. Zde je potřeba nakonfigurovat několik částí. Proxy, na které poslouchá nástroj OWASP ZAP, můžeme měnit v nastavení samotného nástroje a JDK, je ponechána možnost *InheritFromJob*. V části *Installation Method* je zaškrtnuta možnost *System Installed*, kde je do políčka doplněna proměnná `ZAPROXY_HOME`. V části *ZAP Settings* je pak potřeba vyplnit cestu do nastavení nástroje ZAP.
Jestliže nebylo provedeno při konfiguraci ZAPu, je třeba překopírovat session soubory ZAPu do *workspace* Jenkins projektu a v sekci *Session Management* k nim vybrat cestu. V sekci *Session Properties* je pak třeba nakonfigurovat kontext testování. Zde je možné napsat cesty, které jsou testem sledovány nebo lze popsat ty, které jsou z testů vynechány. Konfigurace této části je zobrazena na obrázku 4.7.




Obr. 4.7: Jenkins nastavení projektu – Build steps – ZAP

Poslední dvě části jsou věnovány způsobu testování. V části *Attack Mode* je vybrán *Starting point* – tedy výchozí bod ze kterého je započato testování – zvolena vstupní stránka aplikace. Dále je možné vybrat mód testovaní. Zde jsou k dispozici volby *Spider Scan, AJAX Spider, Active Scan* s možnostmi konfigurace. Následuje část *Finalize Run*, která nabízí generování reportů ve formátech *xml* a *html* s možnostmi cest uložení. V našem případě je vybrát *html* report, který je po testu uložen do *workspace* adresáře.

- ***Post-build Actions*** – je necháno beze změny. Lze využít například k další práci se reporty.

Tímto je dokončena konfigurace projektu. Build aplikace se spustí při změnách v repozitáři nebo ho lze vyvolat ručně. Jako výsledek úspěšného buildu je vytvořen `html` report, který je možné dále vyhodnocovat

## Vyhodnocení bezpečnostních testů

Jako výsledek proběhlých bezpečnostní testů je vytvořen obsáhlý report, jehož část je vidět na obrázku 4.8. Tento report je ve výchozím nastavení k nalezení ve složce *report*, nacházející se v pracovním adresáři Jenkins projektu. Nalezené bezpečnostní nedostatky jsou v reportu rozděleny do několika úrovní rizika dle závažnosti - *High, Medium, Low, Informational*. Úrovně vycházejí z hodnocení metodikou OWASP Risk Rating Methodology, která je popsaná v kapitole 3.5.9. Je zde také zpřehledněn počet jejich výskytů. Níže v reportu je pak u zjištěných zranitelností popis o jakou zranitelnost se jedná, kde v aplikaci se vyskytuje, jaké jsou způsoby jejího řešení a případné odkazy, které jsou nasměrovány na stránky obsahující další informace o dané zranitelnosti. To vše je prezentováno v tabulkové formě, kdy je pro přehlednost závažnost zranitelnosti odlišena barvou záhlaví tabulky.

**ZAP Scanning Report**

**Summary of Alerts**

| Risk Level | Number of Alerts |
|---|---|
| High | 5 |
| Medium | 4 |
| Low | 5 |
| Informational | 0 |

**Alert Detail**

| High (Medium) | Path Traversal |
|---|---|
| Description | The Path Traversal attack technique allows an attacker access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in such a way that the web site will execute or reveal the contents of arbitrary files anywhere on the web server. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal.<br><br>Most web sites restrict user access to a specific portion of the file-system, typically called the "web document root" or "CGI root" directory. These directories contain the files intended for user access and the executable necessary to drive web application functionality. To access files or execute commands anywhere on the file-system, Path Traversal attacks will utilize the ability of special-characters sequences.<br><br>The most basic Path Traversal attack uses the "../" special-character sequence to alter the resource location requested in the URL. Although most popular web servers will prevent this technique from escaping the web document root, alternate encodings of the "../" sequence may help bypass the security filters. These method variations include valid and invalid Unicode-encoding ("..%u2216" or "..%c0%af") of the forward slash character, backslash characters ("..\") on Windows-based servers, URL encoded characters "%2e%2e%2f"), and double URL encoding ("..%255c") of the backslash character.<br><br>Even if the web server properly restricts Path Traversal attempts in the URL path, a web application itself may still be vulnerable due to improper handling of user-supplied input. This is a common problem of web applications that use template mechanisms or load static text from files. In variations of the attack, the original URL parameter value is substituted with the file name of one of the web application's dynamic scripts. Consequently, the results can reveal source code because the file is interpreted as text instead of an executable script. These techniques often employ additional special characters such as the dot (".") to reveal the listing of the current working directory, or "%00" NULL characters in order to bypass rudimentary file extension checks. |

Obr. 4.8: Report nástroje OWASP ZAP

Report je velice přehledný a obsahuje všechny informace, které mohou napomoci k řešení problému. V závislosti na tomto výstupu je dále možné během *Post-build* akce založit například Bug v nástroji pro evidenci chyb a zaslat upozornění o tomto problému.

# 5 ZÁVĚR

Tato diplomová práce seznamuje čtenáře s problematikou internetové bezpečnosti a možnostmi jejího testování. Zaměřuje se na její praktické aspekty a snaží se s využitím volně dostupných nástrojů čtenáři přiblížit způsob, jakým může být testováno zabezpečení webových aplikací.

V první části práce je představena základní terminologie vztahující se k internetové bezpečnosti a představena organizace Open Web Application Security Project (OWASP), jež má v dnešní době významný podíl na propagaci tvorby bezpečných aplikací a metodik jejich testování. Dále je představen žebříček nejkritičtějších zranitelností sestavený touto organizaci – OWASP Top 10.

Po této úvodní části jsou představeny linuxové distribuce Kali a Parrot Security, určené k provádění penetračních testů. Z těchto distribucí jsou vybrány testovací nástroje a u nich je demonstrováno použití v jednotlivých fázích průzkumu.

Pro průzkumnou fázi byly představeny nástroje určené k získávání informací. Mezi ně patří Fiece, dig nebo SubBrute pro získání seznamu subdomén. Po nich následují komplexnější nástroje jako CeWl, DirBuster, WhatWeb a Maltego určené k získání konkrétnějších informací o vybraném cíli. Nástroje se liší způsoby i zdroji, ze kterých jsou informace získávány. V závěru této části jsou představeny vyhledávače Shodan a Google. U nich jsou uvedeny možnosti pokročilého vyhledávání a jejich využití pro provádění penetračních testů.

K účelům provádění dalších testů jsou zmíněny možnosti využití bezpečnostních trenažérů ze kterých byly v práci využity především trenažéry, které jsou součástí projektu OWASP Broken Web Application Project. Na těch je demonstrováno skenovaní aplikací pomocí nástrojů Wget, BurbSuite, či HTTrack a jejich analýza pomocí nástrojů WebScarab nebo John the Ripper. V navazující fázi exploitace jsou pak v jejím úvodu představeny nástroje specializované k odhalovaní určitého typu zranitelnosti. Mezi ně patří prohlížeč OWASP Mantra, The BeEF nebo SQLMap. Druhá část je věnována komplexnějším nástrojům jako jsou Metasploit framework a automatické scannery. Mezi představené scannery patří Nikto, Wapiti, w3af nebo Vega scanner. Kapitolu uzavírá nástroj OWASP ZAP, kde je kromě jeho schopností i podrobněji popsána metodologie OWASP Risk Rating Methodology, používaná při hodnocení závažnosti zranitelností v reportech, které jsou výsledkem testování tímto nástrojem

Poslední část práce se věnuje jedné z technik agilního vývoje známé pod označením Continous Integration, neboli průběžná integrace. V této části je kladen důraz na objasnění principů této metodiky tak, aby byly zřejmé její přínosy při použití

ve vývoji aplikací. Práci uzavírá praktický příklad průběžné integrace s použitím integračního serveru Jenkins a bezpečnostního nástroje OWASP ZAP pro provádění automatizovaných bezpečnostních testů.

Ačkoliv je práce cílena spíše na čtenáře kteří se zabývají vývojem aplikací nebo jejich testováním, kladla si za cíl srozumitelnou formou nastínit význam a možnosti bezpečnostních testů i méně pokročilým čtenářům.

Při tvorbě práce byly nástroje vybírány tak, aby pokryly co největší možnou škálu zranitelností. Během používání nástrojů jsem vycházel z volně dostupných zdrojů a s většinou nástrojů, které jsou v práci uvedeny , jsem se seznamoval při její tvorbě. Zarážející věcí byla pro mě jednoduchost, bezproblémový provoz, účelnost a stabilita těchto nástrojů, které jsou navíc dostupné na jednom místě v uváděných linuxových distribucích. Použití nástrojů v této práci je představeno v kontextu zvýšení bezpečnosti aplikací, avšak je potřeba si uvědomit, že stejně tak mohou být tyto nástroje zneužity pro trestnou činnost. Toto by dle mého názoru mělo být motivací při jejich použití ve vývojové procesu webových aplikací.

# 6 LITERATURA


[1] GOODMAN, Marc. *Future crimes: everything is connected, everyone is vulnerable and what we can do about it*. New York: Doubleday, c2015. ISBN 978-0-385-53900-5.

[2] ANON., 2016. *Webová aplikace* [online]. [vid.2017-01-02]. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=Webov%C3%A1_aplikace&oldid=14427111>.

[3] ANON., 2017. *CSIRT.CZ*[online]. [vid.2017-03-25]. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=CSIRT.CZ&oldid=14766340>

[4] ANON., nedatováno. *What Are Web Applications?* [online] . [vid.2017-01-16]. Dostupné z: <http://www.acunetix.com/websitesecurity/web-applications/>.

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

[6] ANON., nedatováno. *Terminologický slovník - krizové řízení a plánování obrany státu - Ministerstvo vnitra České republiky* [online] [vid.2017-01-03]. Dostupné z: <http://www.mvcr.cz/clanek/terminologicky-slovnik-krizove-rizeni-a-planovani-obrany-statu.aspx>.

[7] ČERMÁK, Miroslav, 2008. *CIA: Důvěrnost-Integrita-Dostupnost.* CleverAndSmart [online]. [vid.2017-01-30]. Dostupné z: <hhttp://www.cleverandsmart.cz/duvernost-integrita-dostupnost/>.

[8] ČERMÁK, Miroslav, 2016. *Přečtěte si, co je to vektor útoku, zranitelnost, exploit a payload*. CleverAndSmart [online]. [vid.2017-01-03]. Dostupné z: <http://www.cleverandsmart.cz/prectete-si-co-je-to-vektor-utoku-zranitelnost-exploit-a-payload/>.

[9] KROPÁČOVÁ, Andrea, nedatováno. *CERT/CSIRT týmy a jejich role. Root.cz* [online] [vid.2017-03-25]. Dostupné z: <https://www.root.cz/clanky/cert-csirt-tymy-a-jejich-role/>.

[10] POŽÁR, Josef, nedatováno. *Vybrané hrozby informační bezpečnosti organizace*[online] [vid.2017-01-03]. Dostupné z: <http://www.cybersecurity.cz/data/Pozar2.pdf>.

[11] ANON., nedatováno. *Threat Modeling* [online] [vid.2017-04-15]. Dostupné z: <https://msdn.microsoft.com/en-us/library/aa302419.aspx#c03618429_011>.

[12] OWASP Foundation. *OWASP Risk Rating Methodology - OWASP* [online] [vid.2017-03-09]. Dostupné z: <https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology>.

[13] ANON., nedatováno. *Výkladový slovník kybernetické bezpečnosti* [online] [vid.2017-01-05]. Dostupné z: <https://www.govcert.cz/download/aktuality/container-nodeid-548/slovnikv231nbuwebcolor.pdf>.

[14] ANON., nedatováno. *Trestní zákoník – 40/2009 Sb. | Zákony.centrum.cz* [online] [vid.2017-01-05]. Dostupné z: <http://zakony.centrum.cz/trestni-zakonik/>.

[15] ZÁVODSKÝ, Petr, nedatováno. *Úctyhodný Open Web Application Security Project (OWASP).* Root.cz [online] [vid.2017-03-26]. Dostupné z: <https://www.root.cz/clanky/uctyhodny-open-web-application-security-project-owasp/>

[16] OWASP Foundation. *OWASP Top 10 - 2013* [online] [vid.2017-03-09]. Dostupné z: <https://www.owasp.org/index.php/Top_10_2013-Top_10>.

[17] ANON., nedatováno. *OSSTMM 3* [online] [vid.2017-01-06]. Dostupné z: <http://www.isecom.org/mirror/OSSTMM.3.pdf>

[18] ANON., nedatováno. *Technical guide to information security testing and assessment*[online] [vid.2017-01-10]. Dostupné z: <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf>

[19] SUTTON, Michael, GREENE, Adam a Pedram AMINI. *Fuzzing: Brute Force Vulne- rability Discovery Upper Saddle River*, NJ: Addison-Wesley, 2007. ISBN 978-0-321- 44611-4.

[20] OWASP Foundation. *Testing Guide 4.0* [online]. c 2014 [cit. 2017-01-27]. Dostupné z: <https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf>.

[21] OWASP Foundation. *OWASP Broken Web Applications Project - OWASP* [online] [vid.2017-01-08]. Dostupné z: <https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project>.

[22] ZÁVODSKÝ, Petr, nedatováno. *Sexy proxy chrobák WebScarab analyzuje aplikace*. Root.cz [online] [vid.2017-01-15]. Dostupné z: <https://www.root.cz/clanky/sexy-proxy-chrobak-webscarab-analyzuje-aplikace/>.

[23] ANON., nedatováno. *Gray box testing - Wikipedia* [online] [vid.2017-03-10]. Dostupné z: <https://en.wikipedia.org/wiki/Gray_box_testing>

[24] ANON., nedatováno. *Black-box testing - Wikipedia* [online] [vid.2017-03-10]. Dostupné z: <https://en.wikipedia.org/wiki/Black-box_testing>

[25] ANON., nedatováno. *White-box testing - Wikipedia* [online] [vid.2017-03-10]. Dostupné z: <https://en.wikipedia.org/wiki/White-box_testing>

[26] ANON., nedatováno. *A Shodan Tutorial and Primer* [online] [vid.2017-03-27]. Dostupné z: <https://danielmiessler.com/study/shodan/>

[27] LONG, J. *Google hacking for penetration testers.* Burlington, MA: Syngress Pub 2008. ISBN 0080484263.

[28] ANON., nedatováno. *Google hacking - Wikipedia* [online] [vid.2017-04-16]. Dostupné z: <https://en.wikipedia.org/wiki/Google_hacking>

[29] OWASP Foundation. *OWASP Zed Attack Proxy Project*. [online] [vid.2017-01-25]. Dostupné z: <https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project>.

[30] OWASP Foundation. *ZAPpingTheTop10 - OWASP*.[online] [vid.2017-01-25]. Dostupné z: <https://www.owasp.org/index.php/ZAPpingTheTop10>.

[31] PORTSWIGGER WEB SECURITY. *Burp Suite*. [online] [vid.2017-01-31]. Dostupné z: <https://portswigger.net/burp/>.

[32] PortSwigger Web Security. *Burp Extender*. [online] [vid.2017-01-31]. Dostupné z: <https://portswigger.net/burp/extender>.

[33] PORTSWIGGER WEB SECURITY. *BApp Store*. [online] [vid.2017-01-31]. Dostupné z: <https://portswigger.net/bappstore/>.

[34] PORTSWIGGER WEB SECURITY. *Using Burp to Test for the OWASP Top Ten. PortSwigger Web Security* [online] [vid.2017-01-31]. Dostupné z: <https://support.portswigger.net/customer/portal/articles/1969845-using-burp-to-test-for-the-owasp-top-ten>.

[35] ANON., nedatováno. *Automated Audit using WAPITI - OWASP* [online] [vid.2017-03-28]. Dostupné z: <https://www.owasp.org/index.php/Automated_Audit_using_WAPITI>.q

[36] ČÍŽEK, Jakub, nedatováno. *Proměnili jsme prohlížeč v zombie a zapojili počítač do malého botnetu. Živě.cz* [online] [vid.2017-03-11]. Dostupné z: <http://www.zive.cz/clanky/promenili-jsme-prohlizec-v-zombie-a-zapojili-pocitac-do-maleho-botnetu/sc-3-a-182111/default.aspx>.

[37] JASWAL, N. *Mastering Metasploit - Second Edition*, Packt Publishing, Limited. 2016. ISBN 9781786463166.

[38] ANON., nedatováno. *Co je to Payload? - IT Slovník* [online] [vid.2017-03-12]. Dostupné z: <http://it-slovnik.cz/pojem/payload>

[39] ANON., nedatováno. *WMAP Web Scanner* [online] [vid.2017-03-19]. Dostupné z: <https://www.offensive-security.com/metasploit-unleashed/wmap-web-scanner/>

[40] ANON., 2014. *W3af* [online]. [vid.2017-03-19]. Dostupné z: <https://en.wikipedia.org/wiki/W3af>

[41] FOWLER, Martin, 2006. *Continuous Integration* [online] [vid.2017-04-05]. Dostupné z: <https://www.martinfowler.com/articles/continuousIntegration.html>

[42] ANON., nedatováno. *Průběžná integrace – Wikipedie* [online] [vid.2017-04-05]. Dostupné z: <https://cs.wikipedia.org/wiki/Pr%C5%AFb%C4%9B%C5%BEn%C3%A1_integrace>

[43] ANON., nedatováno. *What is Version Control: Centralized vs. DVCS - Atlassian Blogs* [online] [vid.2017-04-09]. Dostupné z: <https://www.atlassian.com/blog/2012/02/version-control-centralized-dvcs>

[44] ANON., nedatováno. *Software testing - Wikipedia* [online] [vid.2017-04-09]. Dostupné z: <https://en.wikipedia.org/wiki/Software_testing>

[45] DWARKANI, Bharat , 2017. *GitFlow: The Easy Release Management Workflow* [online] [vid.2017-04-09]. Dostupné z: <https://blog.axosoft.com/2017/01/31/gitflow/>

[46] SMART, J. F.: Jenkins: *The Definitive Guide*. USA: O'Reilly Media, první vydání, 2011, 405 s., iSBN 978-1-449-30535-2.

# 7 SEZNAM ZKRATEK

| | |
|---|---|
| ITU | International Telecommunication Union |
| CSIRT | Computer Security Incident Response Team |
| OWASP | Open Web Application Security Project |
| ASP | Active Server Pages |
| XSS | Cross-Site Scripting |
| CSRF | Cross-Site Request Forgery |
| CMS | Content Management System – systém pro správu obsahu |
| OSSTMM | Open Source Security Testing Methodology Manual |
| DVWA | Damn Vulnerable Web App |
| MSF | Metasploit framework |
| IPS | Intrusion Prevention Systems |
| IDPS | Intrusion Detection and Prevention Systems |
| ZAP | Zed Attack Proxy |
| CI | Continuous Integration |
| VCS | Version Control Systems |
| DVCS | Distributed Version Control System |
| XML | eXtensible Markup Language |
| HTML | HyperText Markup Language |

# 8 PŘÍLOHY

1) **javaVulnerableApp.xml** – export konfigurace ukázkového projektu použitého v této práci
2) **JENKINS_ZAP_VULNERABILITY_REPORT.html** – report vygenerovaný nástrojem OWASP ZAP po provedení bezpečnostních testů ukázkové aplikace

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2015/2016

Studijní program: Aplikovaná informatika
Forma: Kombinovaná
Obor/komb.: Aplikovaná informatika (ai2-k)

**Podklad pro zadání DIPLOMOVÉ práce studenta**

| PŘEDKLÁDÁ: | ADRESA | OSOBNÍ ČÍSLO |
|---|---|---|
| Ing. Čejka Josef | Albertova 891/4, Hradec Králové - Pražské Předměstí | I1500069 |

**TÉMA ČESKY:**

Nástroje pro penetrační testování webových aplikací a jejich praktické využití

**TÉMA ANGLICKY:**

Penetration testing tools and their usage

**VEDOUCÍ PRÁCE:**

Ing. Pavel Kříž, Ph.D. - KIKM

**ZÁSADY PRO VYPRACOVÁNÍ:**

Práce se zabývá tématem ověřování bezpečnosti webových aplikací pomocí výsledků penetračních testů. Cílem práce je zmapovat současné bezplatně dostupné nástroje pro provádění penetračních testů. Následně u vybraných nástrojů demonstrovat možnosti jejich praktického využití k odhalování nejběžněji se vyskytujících bezpečnostních rizik.

Osnova:
1.Úvod do problematiky zabezpečení webových aplikací
2.Testování webových aplikací a nejběžnější bezpečnostní nedostatky
3.Přehled a stručný popis běžně užívaných nástrojů pro penetrační testy
4.Praktická ukázka využití vybraných nástrojů pro provádění penetračních testů

**SEZNAM DOPORUČENÉ LITERATURY:**

DOSTÁLEK, Libor, Marta VOHNOUTOVÁ a Miroslav KNOTEK. Velký průvodce infrastrukturou PKI a technologií elektronického podpisu. 2., aktualiz. vyd. Brno: Computer Press, 2009.
Ansari, Juned Ahmed. Web Penetration Testing with Kali Linux: Build Your Defense Against Web Attacks with Kali Linus 2.0. second ed. Community Experience Distilled. Birmingham: Packt Publishing, 2015.
Dieterle, Daniel W. Intermediate Security Testing with Kali Linux 2. CreateSpace Independent Publishing Platform, 2015.
Daswani, Neil, Christoph Kern, and Anita Kesavan. Foundations of Security: What Every Programmer Needs to Know. The Expert's Voice in Security. Berkeley, CA: Apress, 2007.
Mueller, John. Security for Web Developers. Sebastopol, CA: O, 2015.
HOWARD, Michael a David LEBLANC. Bezpečný kód: [techniky a strategie tvorby bezpečných webových aplikací]. Vyd. 1. Brno: Computer Press, 2008.

**Podpis studenta:** ..... **Datum:** 14.10.16

**Podpis vedoucího práce:** ..... **Datum:** 14.10.16