VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO PENETRAČNÍ TESTOVÁNÍ WEBOVÝCH APLIKACÍ

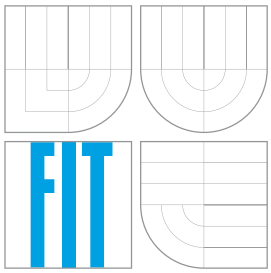
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL DOBEŠ

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO PENETRAČNÍ TESTOVÁNÍ WEBOVÝCH APLIKACÍ

THE TOOL FOR PENETRATION TESTS OF WEB APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL DOBEŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MAROŠ BARABAS

BRNO 2015

Abstrakt

Tato práce se zabývá problematikou penetračního testování webových aplikací, se zaměřením na zranitelnosti Cross-Site Scripting (XSS) a SQL Injection (SQLI). Popisuje technologie webových aplikací, nejběžnější zranitelnosti dle OWASP Top 10 a motivaci pro penetrační testování. Uvádí principy, dopady a doporučení pro nápravu zranitelností Cross-Site Scripting a SQL Injection. V rámci praktické části práce byl implementován nástroj pro podporu penetračního testování. Tento nástroj je rozšiřitelný prostřednictvím modulů. Byly implementovány moduly pro detekci zranitelností Cross-Site Scripting a SQL Injection. Vytvořený nástroj byl porovnán s existujícími nástroji, mimo jiné s komerčním nástrojem Burp Suite.

Abstract

The thesis discusses the issues of penetration testing of web applications, focusing on the Cross-Site Scripting (XSS) and SQL Injection (SQLI) vulnerabilities. The technology behind web applications is described and motivation for penetration testing is given. The thesis then presents the most common vulnerabilities according to OWASP Top 10. It lists the principles, impact and remediation recommendations for the Cross-Site Scripting and SQL Injection vulnerabilities. A penetration testing tool has been developed as a part of this thesis. The tool is extendable via modules. Modules for detection of Cross-Site Scripting and SQL Injection vulnerabilities have been developed. The tool has been compared to existing tools, including the commercial tool Burp Suite.

Klíčová slova

Bezpečnost, webové aplikace, Cross-Site Scripting, XSS, SQL Injection, SQLI, penetrační testování, zranitelnost, OWASP

Keywords

Security, web applications, Cross-Site Scripting, XSS, SQL Injection, SQLI, penetration testing, vulnerability, OWASP

Citace

Michal Dobeš: Nástroj pro penetrační testování webových aplikací, bakalářská práce, Brno, FIT VUT v Brně, 2015

Nástroj pro penetrační testování webových aplikací

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Maroše Barabase. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Dobeš
14. května 2015

Poděkování

Děkuji svému vedoucímu, Ing. Maroši Barabasovi, za čas věnovaný vedení mé práce, za ochotu při konzultacích a za jeho konstruktivní připomínky.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá problematikou penetračního testování webových aplikací. Penetrační testování je metoda, při níž se tester snaží prolomit zabezpečení testované aplikace a napodobuje tak chování reálného útočníka. Díky tomu metoda testuje odolnost aplikace vůči těm útokům, kterým bude s nejvyšší pravděpodobností vystavena při reálném provozu.

V poslední době zaznamenávají webové aplikace obrovský růst uživatelské základny. S tím souvisí i značný nárůst počtu útoků na tyto aplikace a vznik nových kategorií bezpečnostních útoků. Překvapivě mnoho webových aplikací je zranitelných – podle průzkumu společnosti Symantec z roku 2013 [23] bylo 77 % prověřených webových aplikací zranitelných, z toho 16 % obsahovalo kritické zranitelnosti. Tyto zranitelnosti mohou mít vážné následky – v praxi často dochází ke krádeži osobních údajů, včetně čísel kreditních karet. Může dojít i k ovládnutí serveru, na kterém aplikace běží[2]. Napadený web může být také zneužit k šíření malware, například v rámci tzv. *watering hole*[1] útoku.

Penetrační testování je forma bezpečnostního auditu, při kterém se auditor snaží testovanou aplikaci napadnout z pozice reálného útočníka. Výsledkem je zpráva o nalezených zranitelnostech, která vlastníkovi webové aplikace pomůže opravit bezpečnostní slabiny. Penetrační testování tak spadá pod pojem *ethical hacking* [16].

Jedná se však o zdlouhavý proces, z čehož vyplývá potřeba jej alespoň částečně automatizovat. Byla tak vytvořena řada nástrojů pro penetrační testování nejen webových aplikací. K nejznámějším patří Metasploit Framework společnosti Rapid7 či Burp Suite firmy Portswigger web security.

[1] Termín *watering hole* označuje útok, při kterém je zdroj navštěvovaný velkým množstvím klientů zneužit k šíření útoku na tyto klienty. Vhodným výběrem zdroje může útočník poměrně přesně zasáhnout určitou cílovou skupinu. Termín je odvozen z chování predátorů lovících zvířata, která se shromažďují u zdroje vody.

1.1 Cíl práce

Cílem této práce je přiblížit problematiku penetračního testování a vyhledávání webových zranitelností, včetně principů nejběžnějších zranitelností. Zvláštní důraz bude kladen na zranitelnosti Cross-Site Scripting a SQL Injection.

V rámci práce bude vyvinut rozšiřitelný framework, který usnadní penetrační testování. Následně budou pro tento framework vytvořeny moduly pro detekci zranitelností Cross-Site Scripting a SQL Injection. Tyto moduly budou sloužit pro demonstraci funkčnosti řešení. Vytvořený nástroj bude porovnán s již existujícími řešeními a budou diskutovány možnosti jeho rozšíření.

1.2 Struktura práce

V následující kapitole je diskutován současný stav. Jsou přiblíženy technologie webových aplikací a nejčastější zranitelnosti těchto aplikací. Zranitelnosti SQL Injection a Cross-Site Scripting jsou přiblíženy detailněji. V závěru kapitoly jsou uvedeny existující nástroje pro penetrační testování.

Třetí kapitola se věnuje návrhu nástroje pro penetrační testování. Uvádí požadavky, které jsou na tento nástroj kladeny, a rozděluje jeho funkcionalitu do jednotlivých komponent. Jako součást návrhu uvádí i postup práce s nástrojem, grafické uživatelské rozhraní a API, které budou využívat moduly tohoto nástroje. Závěr kapitoly se věnuje návrhu detekčních modulů, včetně použitých metod detekce zranitelností.

Čtvrtá kapitola popisuje implementační detaily navrženého nástroje, včetně detekčních modulů. Pátá kapitola se zabývá testováním vytvořeného nástroje a jeho porovnáním s existujícími řešeními, včetně komerčního nástroje Burp Suite. Práci uzavírá šestá kapitola, která shrnuje dosažené výsledky a uvádí podněty pro další vývoj.

Kapitola 2

Analýza a současný stav

Jelikož je cílem této práce vyvinout nástroj pro vyhledávání webových zranitelností, je třeba nejprve vysvětlit jejich principy, vznik a možné dopady. Webové zranitelnosti vznikají programátorskou chybou, nejčastěji neošetřením vstupních dat. Pokud aplikace přebírá data od uživatele, mohou tato data obsahovat takovou sekvenci, která způsobí, že se aplikace zachová jiným způsobem, než programátor zamýšlel. Tato odchylka od zamýšleného chování představuje *zranitelnost*. Vstupní data je tedy třeba kontrolovat a zajistit, aby data s nevhodnými sekvencemi byla buďto odmítnuta, nebo upravena tak, aby aplikaci neuškodila. Jinak řečeno, vstupní proměnné, které přicházejí z vnějšího prostředí, je třeba ošetřit.

Vstup aplikace, který může být tímto způsobem zneužit, označíme jako *zranitelný vstup*. Cílem bezpečnostního auditu je identifikovat co nejvíce zranitelných vstupů, aby následně mohly být opraveny. Tím dojde ke zmenšení množiny zranitelných vstupů – jinými slovy, aplikace se stane bezpečnější.

Při identifikaci zranitelných vstupů může v zásadě dojít ke dvěma druhům nepřesností. Prvním je chyba *false positive*, kdy je vstupní parametr označen jako zranitelný, přestože pro útok nemůže být použit a zranitelný tedy není. Druhým druhem je chyba *false negative*, při které vstup, který je zranitelný, není označen jako zranitelný. V praxi se snažíme omezit počet chyb obou typů, jako přijatelnější však považujeme chyby typu false positive.

Zranitelné vstupy lze hledat i jinými způsoby, než penetračním testováním. V praxi se například používá revize zdrojového kódu aplikace, ať už manuální či automatizovaná. Penetrační testování však simuluje reálný útok na aplikaci a umožňuje odhalit i zranitelnosti, které by byly při revizi kódu obtížně zjistitelné. Některé komerční penetrační nástroje kombinují automatickou revizi kódu s penetračním testováním.

2.1 Technologie webových aplikací

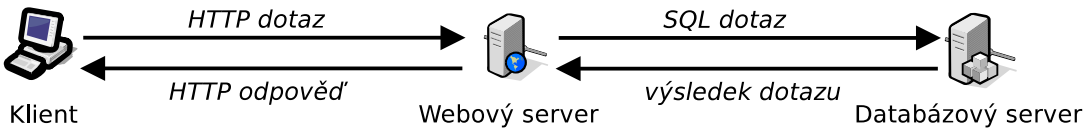
Aby bylo možno vysvětlit principy webových zranitelností, je třeba nejdříve přiblížit technologie, na kterých jsou tyto aplikace postaveny. Typické prostředí, ve kterém webová aplikace pracuje, je znázorněno na obrázku 2.1. Tvoří jej:

1. *Klient* aplikace, obvykle webový prohlížeč, běžící na uživatelově zařízení. Zde jsou vykonávány části aplikace označované jako tzv. *dynamický obsah*. Jedná se například o Java Script či Adobe Flash. Klient obvykle aplikaci poskytuje nová data a zobrazuje výstupní data, která mu aplikace předává.

2. *Webový server*, který zpracovává dotazy klienta a odesílá na ně příslušné odpovědi. Ve své podstatě je bezstavový. Informace o stavu tedy musí být udržována na straně

klienta a předávána serveru s každým dotazem. Obvykle se k tomuto účelu používají tzv. *cookies*. Cookie, která označuje klientovu relaci, je známa pod pojmem *session cookie*.

3. *Databázový server*, který slouží k uchovávání dat, například uživatelského obsahu. U menších aplikací často běží na stejném stroji jako webový server.

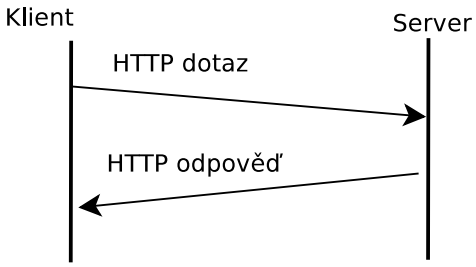


Obrázek 2.1: Zjednodušené schéma webové aplikace.

Pokud klient předá serveru data s nevhodnou sekvencí, může dojít ke zneužití webové zranitelnosti na straně serveru. Jelikož klient se serverem komunikuje prostřednictvím protokolu HTTP, bude tento protokol nyní přiblížen.

2.1.1 Protokol HTTP

Protokol HTTP (Hypertext Transfer Protocol) je klient-server protokol, který pracuje nad protokolem TCP. Komunikace mezi klientem a serverem je znázorněna na obrázku 2.2 a sestává se z dotazů klienta (*HTTP Request*) a odpovědí serveru (*HTTP Response*). Jak dotaz, tak odpověď mohou obsahovat tzv. hlavičky (*HTTP Headers*), ve kterých jsou typicky uloženy informace o spojení a o odesílající straně, včetně cookies. TCP spojení mezi klientem a serverem může mezi jednotlivými dotazy zůstat otevřené, či se může ustavovat pro každý dotaz znovu. To je dáno hodnotou HTTP hlavičky `Connection`.



Obrázek 2.2: Schéma HTTP komunikace

Protokol HTTP je popsán v RFC 7230 [8], v RFC 7231 [9] a dalších. Ty specifikují syntaxi a význam jeho zpráv a nahrazují původní RFC 2616 [7]. Formát HTTP zpráv je detailně popsán v [8] v sekci 3. Podle této sekce vypadá zpráva HTTP takto:

```
HTTP-message   = start-line
                 *( header-field CRLF )
                 CRLF
                 [ message-body ]
```

HTTP dotaz od klienta pak po nahrazení `start-line` vypadá takto ([8, sekce. 3.1.1]):

```
HTTP-request    = method SP request-target SP HTTP-version CRLF
                  *( header-field CRLF )
                  CRLF
                  [ message-body ]
```

HTTP hlavičku specifikuje [8] v sekci 3.2[1]:

```
header-field    = field-name ":" OWS field-value OWS
```

Přenášená data

Přenášená data jsou v RFC označována jako `[ message-body ]`. Jejich interpretace je dána hodnotou hlavičky `Transfer-Encoding`. Jejich délka je dána hlavičkou `Content-Length`. V případě HTTP zprávy typu odpověď představují přenášená data HTML stránku, která má být zobrazena příjemci.

V případě zprávy typu dotaz je často potřeba předat webové aplikaci určité parametry. Parametry lze v dotazu uvést třemi způsoby:

- Jako součást URL cílové stránky. Tento způsob se označuje jako metoda GET. Jednotlivé parametry ve tvaru `parametr=hodnota` jsou od sebe odděleny znakem `&` a do URL jsou vloženy za znak otazníku. Libovolný HTTP dotaz může obsahovat parametry typu GET.
- Jako tělo HTTP dotazu. Tento způsob je označován jako metoda POST. Parametry jsou vloženy do části dotazu označené `message-body`. Jsou kódované jedním z definovaných serializačních formátů. Parametry typu POST může obsahovat pouze HTTP dotaz, jehož metoda je POST.
- Jako hodnotu cookie. Cookies jsou posílány jako součást HTTP hlaviček. Nastavuje je webová aplikace příkazem webovému prohlížeči. Webový prohlížeč je pak aplikaci při každém následujícím dotazu odešle. Obvykle jsou cookies používány k uchování stavu komunikace s klientem.

HTTPS

Aby byl přenášený obsah, například hesla, skryt před útočníky, lze použít protokol HTTPS (HTTP over TLS)[21]. V HTTPS nejdříve dochází k ustavení šifrovaného spojení SSL (Secure Socket Layer), přes které jsou poté odesílány dotazy a odpovědi HTTP.

Nyní již má čtenář dostatečný přehled o technologii webových aplikací. Mohou tedy být přiblíženy jednotlivé zranitelnosti, které těmto aplikacím hrozí.

[1] pozn.: OWS značí Optional WhiteSpace – nepovinnou sekvenci bílých znaků.

2.2 OWASP Top 10

OWASP (Open Web Application Security Project) je organizace, jejímž cílem je informovat o bezpečnostních zranitelnostech webových aplikací, jejich dopadech a obraně proti nim. Sestavuje žebříček deseti nejběžnějších webových zranitelností, známý jako OWASP Top 10[25]. V posledním vydání tohoto žebříčku figurují následující zranitelnosti:

1. **Injection** K útoku Injection může dojít, pokud jsou interpretu aplikace předána nedůvěryhodná data. Útočník má tedy kontrolu nad částí vykonávaného kódu a může tak ovlivnit běh aplikace. Za určitých okolností může dojít ke kompletnímu ovládnutí systému, kde webová aplikace běží[2]. Zranitelností Injection se hlouběji zabývá [22, str. 238]. O SQL Injection, specifickém případu Injection, pojednává sekce 2.2.2.

2. **Broken Authentication and Session Management** Chybná autentizace a správa relace může mít za následek útok na uživatelské účty. Příčinou bývá špatné schéma správy relací a autentizace uživatelů, například nevhodný způsob obnovení zapomenutého hesla.

3. **Cross-Site Scripting (XSS)** K útoku Cross-Site Scripting může dojít, pokud webová aplikace vloží nedůvěryhodná data do webové stránky. Následkem může být útok na klienty aplikace, například neoprávněný přístup k jejich datům. Zranitelnost XSS je hlouběji rozvedena v sekci 2.2.1.

4. **Insecure Direct Object References** Pokud aplikace přistupuje k datům na základě předaného identifikátoru a nekontroluje při tom přístupová práva, může útočník získat data, ke kterým by neměl mít přístup.

5. **Security Misconfiguration** Nevhodná konfigurace webové aplikace, či prostředí, ve kterém běží, může otevřít cestu útočníkům. Příkladem je ponechání výchozího hesla pro správce či ponechání aplikace v ladicím režimu, kdy jsou uživatelům zobrazovány ladicí zprávy.

6. **Sensitive Data Exposure** Citlivá data, například hesla, by se na serveru neměla vyskytovat v čisté podobě. Pokud se v čisté podobě vyskytují a získá je útočník, dochází k expozici citlivých dat. Citlivé údaje by měly být šifrovány, či v případě hesel hashovány. Pro útočníky pak bude citelně složitější tyto údaje zneužít.

7. **Missing Function Level Access Control** Uživatelské rozhraní webové aplikace může být vyvinuto tak, že zobrazuje pouze ty funkce aplikace, ke kterým má uživatel přístup. I tak ale může uživatel zavolat jinou funkci aplikace tím, že zadá URL této funkce do adresového řádku svého prohlížeče. Pokud pak aplikace na serveru neprovede ověření, že má uživatel právo danou funkci spustit, může uživatel spustit funkci, kterou nemá právo spustit.

8. **Cross-Site Request Forgery (CSRF)** Tento útok předpokládá, že je k webové aplikaci přihlášen uživatel. Informace o tom, že je přihlášen, je spjata s hodnotou *session cookie* uloženou v jeho prohlížeči. Když pak tento uživatel navštíví jinou webovou stránku, do které útočník vložil například obrázek, uživatelův prohlížeč otevře URL uvedenou u obrázku. Tato URL může volat funkce webové aplikace, ke které je uživatel přihlášen, jménem uživatele. Zabránit této zranitelnosti je možné například tím, že pro danou funkci aplikace je při každém načtení webové stránky generována náhodná hodnota, která musí být odeslána, aby byla funkce spuštěna.

9. **Using Components with Known Vulnerabilities** Webové aplikace nejsou obvykle psány od základu a využívají tak řady již existujících komponent. Tyto komponenty však mohou obsahovat zranitelnosti. Tým vyvíjející webovou aplikaci by tedy měl sledovat, zda v některé z používaných komponent nebyla objevena zranitelnost. Taková zranitelnost může být veřejně publikována, například v databázi CVE[24]. Pokud tým neaktualizuje komponentu na bezpečnou verzi, mohou útočníci zranitelnost zneužít.

10. **Unvalidated Redirects and Forwards** Webové aplikace mohou využívat přesměrování, některé z nich cíl přesměrování uvádějí jako parametr stránky. Pokud aplikace neověří, že je hodnota tohoto parametru bezpečná, může útočník v parametru uvést vlastní hodnotu a přesměrovat tak uživatele na libovolnou stránku.

Z uvedených zranitelností se budu blíže věnovat zranitelnostem Cross-Site Scripting a Injection, konkrétně zranitelnosti SQL Injection. V případě zájmu se může čtenář o dalších zranitelnostech dozvědět například ve výukové platformě WebGoat[14].

2.2.1 Zranitelnost Cross-Site Scripting (XSS)

Zranitelnost Cross-Site Scripting (XSS) spočívá v tom, že útočník získá kontrolu nad HTML obsahem, který je doručen uživateli webové aplikace. Může tak do zdrojového kódu webové stránky, která se uživateli zobrazí, umístit vlastní kód Java Scriptu. Tento kód je pak spuštěn jako součást aplikace, se kterou uživatel komunikuje. Útočník tedy může různými způsoby zneužít uživatelův účet, ať už k šíření spamu či malware nebo ke krádeži osobních informací[3]. Útoky XSS jsou tedy mířeny proti klientům webové aplikace. I přesto lze jejich prostřednictvím získat kompletní kontrolu nad webovou aplikací[22, str. 377].

V poslední době bylo zranitelností XSS zasaženo veliké množství stránek. Jako příklady uvádím následující tři incidenty:

- Weather.com je celosvětově populární server, který každý měsíc navštíví více než miliarda uživatelů. Přesto však neuplatňoval ani základní obranu proti XSS [19].
- Amazon.com obsahoval závažnou zranitelnost typu XSS, která umožňovala zneužití zákaznických kreditních karet [20].
- Wordpress je platforma využívaná mnoha webovými aplikacemi jako systém pro správu obsahu. Obsahovala však několik zranitelností XSS a útočníci tak mohli získat administrátorský přístup k dané aplikaci [18].

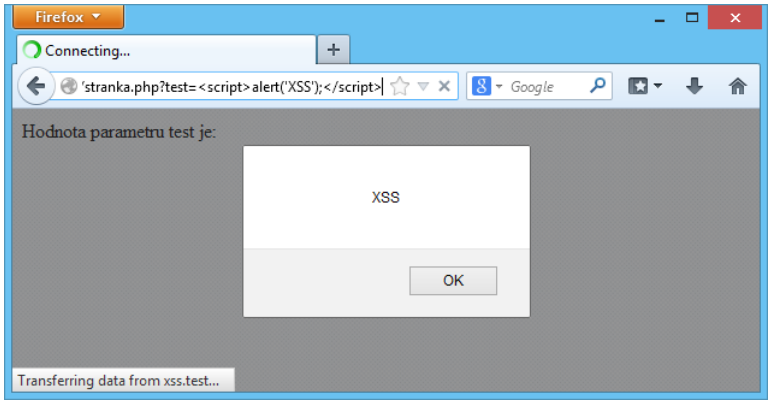
Ukázka zranitelnosti XSS

Obrázek 2.3 ukazuje kód webové stránky v jazyce PHP, který v sobě obsahuje zranitelnost XSS. Pokud této stránce předáme jako hodnotu proměnné `test` kód v Java Scriptu, dojde k jeho spuštění ve webovém prohlížeči. Tato situace je zobrazena na obrázku 2.4. Jako hodnota proměnné `test` byl použit řetězec `<script>alert('XSS');</script>`, což lze vidět v adresovém řádku.

Je třeba poznamenat, že vložený skript může obsahovat libovolný příkaz a může tak být použit například ke krádeži osobních informací. Útočník tak může získat hesla uložená v prohlížeči [11, str. 220] nebo vytvořit kód, který napadá další uživatele[11, str. 386].

```php
echo "<HTML><body>";
echo "Hodnota parametru test je: " . $_GET['test'];
echo "</body></HTML>";
```

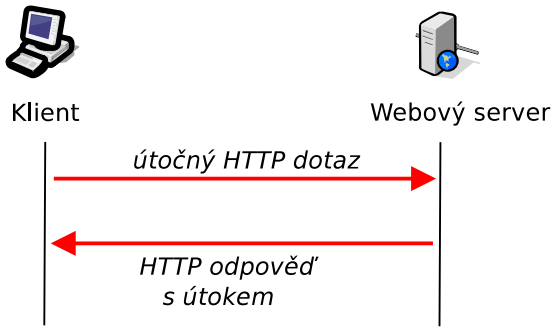
Obrázek 2.3: Ukázka kódu se zranitelností Cross-Site Scripting.



Obrázek 2.4: Úspěšný útok Cross-Site Scripting.

Typy XSS

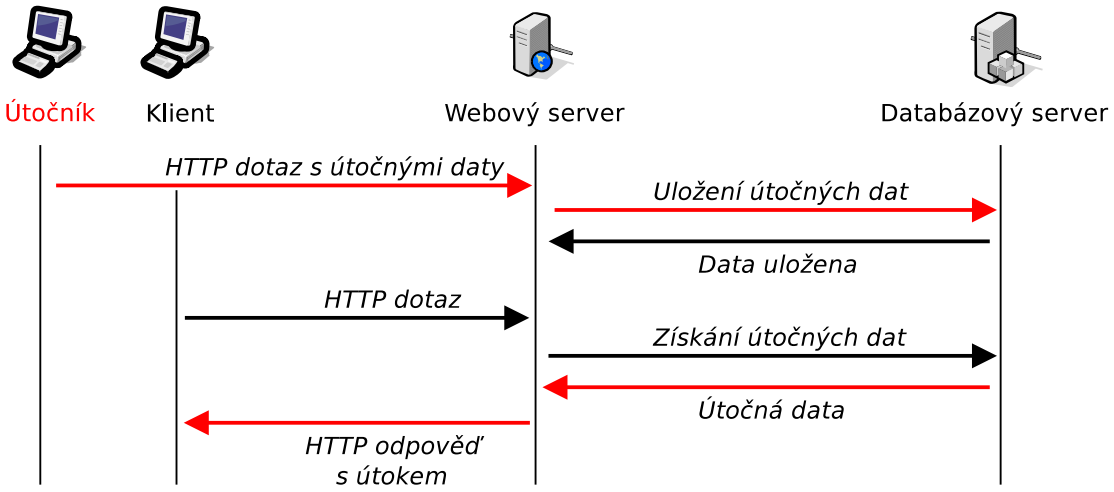
Z hlediska persistence rozlišujeme dva typy zranitelnosti XSS, které jsou v praxi často označovány pojmy *Reflected XSS (RXSS)* a *Stored XSS*. Reflected XSS je charakterizováno tím, že útočná data nesetrvávají na serveru. Musí tak být obsažena v HTTP dotazu. V praxi útok často probíhá tak, že oběť klikne na nastražený odkaz, který útočná data obsahuje v hodnotě jedné z proměnných. Reflected XSS je znázorněno na obrázku 2.5.



Obrázek 2.5: Schéma útoku Reflected XSS.

U Stored XSS naopak útočná data zůstávají uložena na serveru, typicky v databázi. Toto je znázorněno na obrázku 2.6. Stored XSS tak má potenciál ovlivnit každého návštěvníka serveru, na rozdíl od Reflected XSS.

V souvislosti s Cross-Site Scripting se také mluví o jeho jiném typu, tzv. *DOM-based* XSS[22, str. 386]. U tohoto typu útoku jsou útočná data předána Java Script kódu webové aplikace a nemusí při tom být ani odeslána serveru. Jelikož detekce *DOM-based* XSS přesahuje rámec této práce, nebude tento typ dále diskutován.



Obrázek 2.6: Schéma útoku Stored XSS.

Detekce XSS

Přístup k detekci XSS je mimo jiné popsán v patentu společnosti Microsoft[10][2]. Spočívá v tom, že jsou do testované aplikace vkládány jako hodnoty parametrů tzv. *tracery*. Jedná se o řetězce, které mají nízkou pravděpodobnost výskytu ve webové stránce. Když se pak v některé ze stránek testované aplikace vyskytne *tracer*, znamená to, že hodnota daného parametru bylo vložena do webové stránky. Bylo tedy nalezeno spojení mezi parametrem a obsahem stránky. Podle umístění traceru v DOM modelu stránky lze určit, zda může dojít k útoku XSS a jakým způsobem by takový útok měl být veden. Na základě toho je zvolena sada útoků, které jsou vloženy do hodnoty parametru. Analýzou obsahu ovlivněné stránky pak lze určit, zda byl daný útok úspěšný.

Obrana proti útokům XSS

OWASP uvádí jako obranu proti XSS následující doporučení[15]:

- Nevkládat nedůvěryhodná data přímo do skriptů, HTML komentářů a kaskádových stylů a nepoužívat je ani jako názvy atributů či elementů.
- Při vkládání nedůvěryhodných dat provést tzv. *escaping*, tedy nahrazení potenciálně nebezpečných sekvencí znaků za jejich bezpečnou reprezentaci. Metodu pro escaping je třeba zvolit podle toho, v jakém kontextu budou do stránky tato data umístěna (mohou být například vkládána do hodnoty atributu či do textového řetězce v Java Scriptu.)

Lze také použít tzv. *aplikační firewall* (*application firewall*), který v příchozích HTTP dotazech vyhledává útoky. Při zjištění útoku jsou přijata příslušná opatření – dotaz může být například zahozen. Jedním z opensource řešení je ModSecurity[3]. Více informací o aplikačních firewallech je uvedeno v [1, str. 426].

[2] Tento patent byl stažen v roce 2012.
[3] http://modsecurity.org/

2.2.2 Zranitelnost SQL Injection (SQLI)

Zranitelnost SQL Injection (SQLI) spočívá v tom, že útočník získá kontrolu nad SQL dotazem, který webový server posílá databázovému serveru. Může tak dotaz upravit tak, aby z databáze exfiltroval data či aby výsledkem tohoto dotazu ovlivnil běh aplikace ve svůj prospěch. V nejhorším případě lze získat plnou kontrolu nad systémem, na kterém databázový server běží[2]. Narozdíl od XSS jsou útoky na SQLI mířeny proti webovému a databázovému serveru, ne proti klientům. Mezi médii zaznamenané případy zneužití zranitelnosti SQL Injection patří následující události:

- Společnosti WorldView Limited, která zprostředkovává rezervace hotelů, byly ukradeny informace o více než třech tisících kreditních karet[13]. Za tento únik byla společnost pokutována.
- Drupal je, podobně jako výše zmíněný Wordpress, systém pro správu obsahu. Zranitelnost SQL Injection, která byla v Drupalu objevena, umožňovala vykonávání libovolného kódu nad databázovým systémem[17]. Tato zranitelnost byla zveřejněna na internetu a během pouhých sedmi hodin začala být ve velkém rozsahu zneužívána. Tvůrci platformy Drupal doporučili svým uživatelům považovat systém za napadený, pokud na něm nebyly do sedmi hodin po ohlášení zranitelnosti aplikovány bezpečnostní záplaty.

Detekce SQLI a Exfiltrace dat

Data lze z databáze exfiltrovat různými způsoby. První možností je, že webová aplikace zahrne výsledek SQL dotazu v HTTP odpovědi. Data z databáze jsou tak zobrazena jako součást webové stránky. Pokud však webová aplikace nezahrnuje výsledek SQL dotazu do odpovědi, lze do SQL dotazu přidat instrukci, která vykonání tohoto dotazu za určité podmínky zpomalí, a měřit, jak se změní doba odezvy. Tento přístup se označuje jako *Blind SQL Injection*. Souhrnně se pro metody, které používají netradiční kanály pro extrakci informací, používá název *side channel* útok.

Těmto možnostem odpovídají i příslušné způsoby detekce zranitelnosti SQLI. V prvním případě lze do SQL dotazu vložit znak, který při vykonání dotazu způsobí chybu[22, str. 245]. Informace o této chybě pak bude obsažena v HTTP odpovědi. Ve druhém případě lze do dotazu vložit instrukci, která vykonání dotazu zpomalí. Pokud pak pozorujeme zvýšené zpoždění, jedná se pravděpodobně o *blind SQL Injection*.

Ukázka zranitelnosti SQL Injection

Na obrázku 2.7 je uveden kód webové stránky v jazyce PHP.[4] Má zajistit bezpečné přihlášení uživatele, obsahuje však zranitelnost SQL Injection. Hodnoty parametrů `jmeno` a `heslo` nejsou nijak kontrolovány a jsou přímo vloženy do SQL dotazu.

Upravením SQL dotazu se útočník může přihlásit jako libovolný uživatel. Tento útok je zobrazen na obrázku 2.9, kde webová stránka provedla přihlášení uživatele `admin`. Místo hesla byl do SQL dotazu vložen řetězec „`' OR jmeno = 'admin`“. Aby byla zachována syntaktická správnost SQL dotazu, bylo třeba hodnotu parametru `heslo` začít apostrofem a odebrat ukončující apostrof.[5] Výsledný SQL dotaz je zobrazen na obrázku 2.8.

[4] Kód je značně zjednodušený. Mimo jiné by se uživatelská hesla nikdy neměla v databázi objevit, měly by být uloženy pouze jejich hashe.

[5] V praxi se také používá ukončení znakem komentáře.

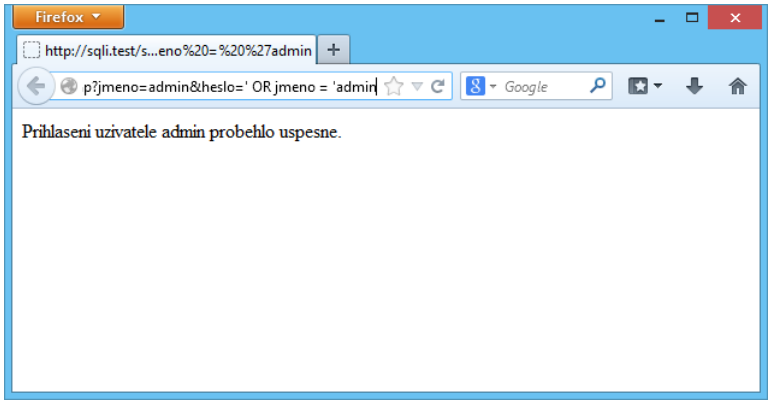
```php
echo "<HTML><body>";
// Kód pro připojení databáze byl vynechán.
$vysledek = mysql_query("SELECT * FROM uzivatele WHERE "
                ."jmeno = '" . $_GET['jmeno'] . "' AND "
                ."heslo = '" . $_GET['heslo'] . "';");

if ($vysledek && $row=mysql_fetch_array($vysledek)) {
        echo "Prihlaseni uzivatele ".$row['jmeno']." probehlo uspesne.";
}
else {
        echo "Spatne prihlasovaci udaje!";
}
echo "</body></HTML>";
```

Obrázek 2.7: Ukázka kódu obsahujícího zranitelnost SQLI.

```
SELECT * FROM uzivatele WHERE
       jmeno = 'admin' AND
       heslo = '' OR jmeno = 'admin';
```

Obrázek 2.8: Výsledný SQL dotaz. Část tvořená proměnnou `heslo` je zarámována.

Tento dotaz vyhledal záznam uživatele `admin` bez toho, aby ověřil heslo. Webová aplikace výsledek dotazu vyhodnotila jako úspěšné přihlášení. Situace je zobrazena na obrázku 2.9.



Obrázek 2.9: Úspěšný útok SQL Injection.

Obrana proti útokům SQLI

Obrana proti SQL Injection spočívá, stejně jako u Cross-Site Scripting, ve vhodném filtrování vstupních proměnných. Vhodnou obranou je také použití parametrizovaných SQL dotazů. Tyto dotazy jsou již předpřipravené a vložením útočného řetězce nedojde ke změně jejich významu. Stejně jako u zranitelnosti XSS lze také použít aplikační firewall.

Starší verze[6] webového serveru Apache se tento problém pokoušely řešit za programátora. V továrním nastavení obsahovaly mechanismus *magic quotes*, který před potenciálně

[6] Před verzí 5.3.0

nebezpečné znaky umisťoval zpětné lomítko. To způsobilo, že tyto znaky ztratily řídicí význam. Tento mechanismus však bylo možno obejít použitím *unicode* reprezentace znaků[6]. Byly popsány vybrané zranitelnosti webových aplikací. Nyní budou přiblížena existující řešení, která jsou schopna webové zranitelnosti vyhledávat.

2.3 Existující nástroje

Existuje mnoho nástrojů pro penetrační testování. Mezi opensource řešení patří například OWASP ZAP či w3af. Mezi špičková komerční řešení patří Nessus, Acunetix WVS, HP WebInspect či Burp Suite.

2.3.1 OWASP ZAP

Nástroj OWASP ZAP (Zed Attack Proxy)[7] poskytuje proxy, přes kterou se tester připojí k testované aplikaci. Podporuje i HTTPS spojení. ZAP sleduje komunikaci a získává informace o adresářové struktuře aplikace a o parametrech jednotlivých stránek. Nástroj také obsahuje *crawler*, který stránky webové aplikace rekurzivně prochází za testera. Je však třeba obezřetnosti, neboť tento přístup v předem neznámém pořadí spustí dostupné funkce webové aplikace. Je doporučeno tuto metodu kombinovat s manuálním průchodem testovanou aplikací[22, str. 65].

Následně lze spustit audit kliknutím na tlačítko *Attack*. ZAP webovou aplikaci prověří a nalezené nedostatky zobrazí v záložce *Alerts*. V náhledu HTTP komunikace je projev daného nedostatku vyznačen.

ZAP může být rozšířen pomocí *Add-Ons*, které do něj lze doinstalovat. Podporuje také ruční vytváření vlastních HTTP dotazů a nabízí nástroje pro převod textu do různých kódování. ZAP je implementován v jazyce Java, díky čemuž je podporován na několika platformách.

2.3.2 w3af

Nástroj w3af[8] nabízí široký výběr detekčních *pluginů*. Jejich výběrem tester určí, které zranitelnosti mají být vyhledávány. Pluginy jsou psány ve skriptovacím jazyce Python. Po jejich výběru je proveden audit webové aplikace. Ve výsledcích auditu jsou vyznačeny části HTTP komunikace, na základě kterých byla zjištěna zranitelnost. Je poskytována možnost manuální tvorby HTTP dotazů. Nástroj w3af je podporován pouze na operačním systému Linux, systém Windows není oficiálně podporován.

2.3.3 Nessus

Nástroj Nessus[9] neslouží pouze k vyhledávání webových zranitelností, ale vyhledává i chyby v zabezpečení systémů a síťové infrastruktury. Vyhledává také viry. Pro ovládání nabízí webové rozhraní. Audit probíhá automaticky po zvolení parametrů. Narozdíl od ostatních nástrojů nenabízí Nessus proxy, přes kterou by tester komunikoval s testovanou aplikací. Nástroj také neobsahuje možnost vytvořit vlastní HTTP dotaz ani nástroje pro kódování textu, jak tomu je u nástroje ZAP.

[7] Dostupný na https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.
[8] Dostupný na http://w3af.org/.
[9] Dostupný na http://www.tenable.com/products/nessus-vulnerability-scanner.

2.3.4 Acunetix WVS

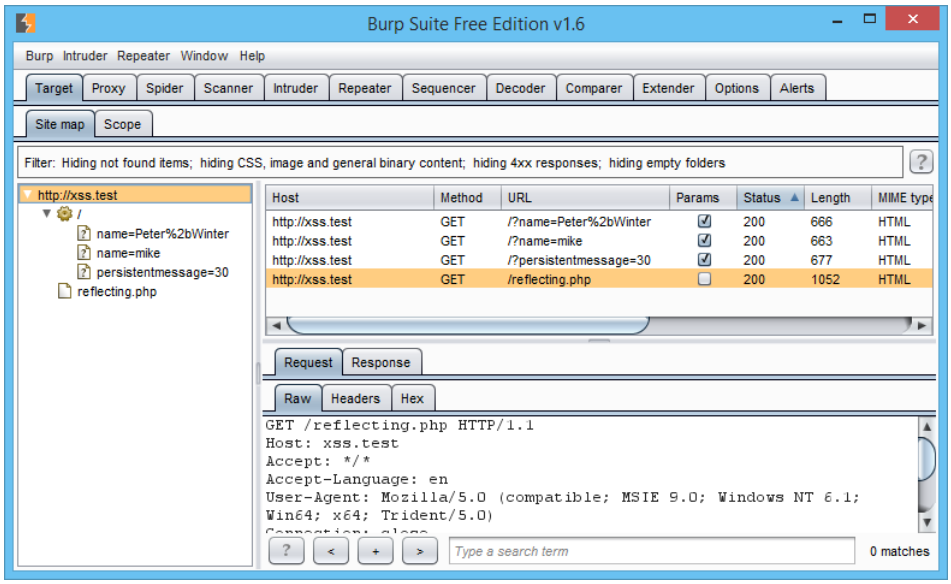
Nástroj Acunetix WVS (Web Vulnerability Scanner)[10] také nabízí proxy, přes kterou se tester připojí k testované aplikaci. Lze nahrát přihlašovací sekvenci k webové aplikaci, aby Acunetix mohl testovat i části aplikace, pro které je nutné přihlášení. Stejně jako OWASP ZAP poskytuje crawler, který prochází adresářovou strukturu testované aplikace. Následně lze spustit audit, který se pokusí odhalit zranitelnosti. Acunetix poskytuje přehledovou obrazovku, na které ve formě grafu znázorňuje počet nalezených zranitelností dle jejich závažnosti. Poskytuje rovněž možnost ručně odeslat vlastní HTTP dotaz. U nalezených zranitelností lze zobrazit HTTP komunikaci, není v ní však vyznačeno, které části odhalují danou zranitelnost.

2.3.5 HP WebInspect

Nástroj WebInspect[11] namísto proxy nabízí integrovaný webový prohlížeč. Umožňuje nahrát přihlašovací sekvenci a zaznamenat stránky, které mají být testovány. Podle zvoleného módu je proveden buď průchod aplikací (crawl), audit, či obojí. Lze nastavit různou úroveň důkladnosti pro průchod adresářovou strukturou aplikace. Nástroj poskytuje přehledovou obrazovku s grafem, podobně jako Acunetix. U nalezených zranitelností lze zobrazit HTTP komunikaci s vyznačenými projevy zranitelností.

2.3.6 Burp Suite

S nástrojem Burp Suite[12] budu své řešení porovnávat, proto jej nyní přiblížím. Aplikace Burp Suite je zachycena na obrázku 2.10. Obsahuje několik komponent, které jsou reprezentovány záložkami v horní části okna:



Obrázek 2.10: Aplikace Burp Suite s náhledem adresářové struktury testované aplikace.

[10] Dostupný na http://www.acunetix.com/vulnerability-scanner/.
[11] Dostupný na http://www8.hp.com/us/en/software-solutions/webinspect-dynamic-analysis-dast/index.html.
[12] Dostupný na http://portswigger.net/burp/.

- Target: Poskytuje náhled na adresářovou strukturu testované aplikace. Také obsahuje definici cíle testu (*scope*), kde lze určit, které webové stránky mají být testovány a které ne.
- Proxy: Zachytává uživatelské HTTP dotazy a ukládá jejich historii. Jednotlivé dotazy umožňuje pozměnit a odeslat.
- Spider: Automaticky prochází testovanou aplikaci a mapuje tak její adresářovou strukturu.
- Scanner: Vyhledává zranitelnosti v testované aplikaci.
- Intruder: Umožňuje tvořit vlastní útoky tím, že na vyznačená místa HTTP dotazu vkládá postupně řetězce (*payloads*) ze sady řetězců.
- Repeater: Umožňuje měnit zachycené HTTP dotazy a odesílat vlastní dotazy.
- Extender: Poskytuje možnost rozšířit funkcionalitu nástroje o další moduly.

Burp Suite dále nabízí nástroje, které uživateli usnadní tvorbu a provádění vlastních útoků. Umožňují například zakódovat útočná data mnoha různými kódováními.

Aplikací Burp Suite končí souhrn existujících nástrojů. Následující kapitola na základě tohoto souhrnu specifikuje požadavky na nový nástroj pro penetrační testování a popisuje jeho návrh.

Kapitola 3

Návrh

Tato kapitola objasňuje návrh frameworku, který bude částečně automatizovat proces penetračního testování. Nejdříve jsou uvedeny požadavky, které musí úspěšné řešení splňovat. Následuje vysokoúrovňový návrh a návrh uživatelského rozhraní. Závěr kapitoly se věnuje návrhu modulů pro detekci zranitelností Cross-Site Scripting a SQL Injection.

3.1 Specifikace požadavků

Framework se bude skládat z několika komponent a bude dále rozšiřitelný prostřednictvím modulů. Bude poskytovat vhodné abstrakce a zprostředkovávat přístup k testované aplikaci. Moduly budou zajišťovat detekci zranitelností. Mohou také poskytovat libovolná rozšíření k již existující funkcionalitě frameworku. Pro komunikaci s uživatelem bude framework obsahovat grafické uživatelské rozhraní (GUI). Framework také musí umožnit modulům, aby poskytly vlastní GUI.

Framework tedy musí poskytnout API, pomocí kterého budou moduly využívat jeho služeb. Součástí tohoto API budou abstrakce z oblasti protokolu HTTP, abstrakce zranitelností a další užitečné abstrakce.

Podobně jako v existujících řešeních bude framework sledovat HTTP komunikaci mezi testerem a testovanou aplikací. Toto bude zajištěno tak, že se tester připojí k testované aplikaci skrz proxy, kterou tento framework bude poskytovat. Jelikož mnoho webových aplikací vyžaduje komunikaci pomocí protokolu HTTPS, musí být proxy schopna zachytávat i tuto komunikaci. Proxy musí být pro dosažení vyšší propustnosti schopna paralelně obsluhovat velké množství spojení. Aby bylo možno testovat například i verze stránek pro mobilní zařízení,

Proxy musí umožnit automatickou záměnu zvolených HTTP hlaviček ve zprávách za uživatelem definované hlavičky. Vhodným nastavením hlavičky `User-Agent` tak bude například možno testovat mobilní verzi webové aplikace.

Zachycené HTTP dotazy a odpovědi budou předávány detekčním modulům. Moduly budou v HTTP komunikaci vyhledávat zranitelnosti a hlásit je frameworku. Musí být také umožněno, aby moduly mohly odeslat vlastní HTTP dotazy a získaly na ně odpovědi. Proxy přitom musí brát ohled na specifikaci cíle testu. Komunikace, která nepatří k definovanému cíli, nesmí být předána žádné jiné komponentě frameworku ani modulům. Pokud se modul pokusí odeslat dotaz, který nesměřuje na cíl testu, nesmí být tento dotaz odeslán.

Framework bude obsahovat náhled na adresářovou strukturu testované aplikace, aby uživatel viděl, ve kterých stránkách aplikace byly nalezeny zranitelnosti. Součástí tohoto

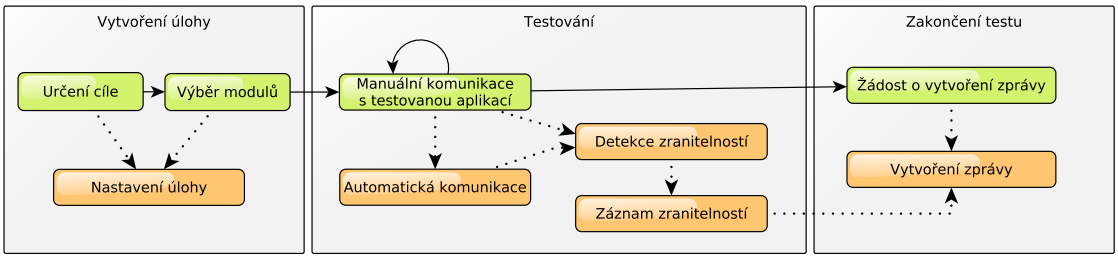
náhledu bude seznam parametrů, které byly pro danou stránku aplikace zachyceny.

Důležitou součástí je seznam zranitelností, který umožní modulům přidávat nalezené zranitelnosti. Uživateli bude poskytnut náhled na tyto zranitelnosti s možností filtrování podle závažnosti, typu a umístění. Uživatel bude také mít možnost změnit hodnocení závažnosti jednotlivých zranitelností. Bude umožněna tvorba výsledné zprávy o penetračním testu, kde budou zranitelnosti uvedeny.

Na požádání bude framework schopen zaznamenat a následně zobrazit všechnu zachycenou HTTP komunikaci. Tato schopnost poskytne detekčním modulům například schopnost statisticky analyzovat historii komunikace a vyhledávat anomálie, které by mohly poukazovat na bezpečnostní zranitelnosti.

3.2 Postup práce s nástrojem

Obrázek 3.1 znázorňuje návrh postupu práce při penetračním testu. Nejdříve uživatel určí cíl testu a vybere moduly pro detekci zranitelností. Na základě těchto informací framework vytvoří *skenovací úlohu*[1]. Poté uživatel interaguje s testovanou aplikací skrz komponentu proxy. Moduly tuto komunikaci sledují a odesílají vlastní dotazy, dochází tedy k automatické komunikaci s testovanou aplikací. Na základě dat z komunikace jsou vyhledávány zranitelnosti. Ty jsou ukládány do záznamu o zranitelnostech. Poté, co uživatel dokončí interakci s testovanou aplikací, požádá o vytvoření zprávy o výsledcích penetračního testu. Framework na základě této žádosti vytvoří výslednou zprávu v podobě HTML dokumentu.



Obrázek 3.1: Posloupnost událostí při testování. V horní části diagramu jsou zeleně znázorněny akce uživatele, ve spodní části jsou oranžově znázorněny odpovídající akce frameworku.

3.3 Struktura frameworku

Framework je rozčleněn do několika částí. UML diagram tříd, který popisuje jeho strukturu, je uveden v příloze na obrázku D.1.

HTTP je balíček, který obsahuje abstrakce různých HTTP entit. Jsou zde definována rozhraní pro webovou stránku, HTTP zprávu, hlavičku a parametr zprávy. Rovněž je zde definován objekt, který tyto entity umí vytvářet (jedná se o návrhový vzor *Factory*).

ScanTask představuje úlohu testování jedné webové aplikace. Obsahuje informace o cíli testu a o používaných detekčních modulech. Pro detekční moduly slouží jako prostředník,

[1] Každá skenovací úloha bude obsahovat přehled o nalezených zranitelnostech, náhled na adresářovou strukturu testované aplikace a zvolené moduly. Nepovinně může obsahovat i záznam zachycené komunikace.

který jim zprostředkovává přístup k ostatním komponentám frameworku. Každá takováto skenovací úloha obsahuje vlastní komponenty SiteExplorer, IssueList a vlastní detekční moduly (viz níže).

Proxy poskytuje ostatním komponentám informace o uživatelově komunikaci s testovanou aplikací. Jednotlivé dotazy a odpovědi zpracovává paralelně za použití fondu vláken (návrhový vzor *Thread Pool*). Je schopna číst jak HTTP, tak HTTPS komunikaci. Její jediná instance je sdílena mezi jednotlivými testovacími úlohami podle návrhového vzoru Jedináček (*Singleton*).

Aby proxy mohla upozorňovat ostatní komponenty na novou HTTP komunikaci, využívá návrhový vzor Pozorovatel (*Observer*), ve kterém figuruje jako pozorovaný objekt. Pokud některá komponenta systému potřebuje dostávat oznámení o nové komunikaci, registruje u proxy svého posluchače (*listener*).

Proxy také jednotlivým komponentám (včetně detekčních modulů) umožňuje odesílat vlastní HTTP požadavky a získávat na ně odpovědi. Pro případy, kdy je třeba odeslat pozměněný HTTP dotaz, umožňuje framework vytvořit kopii tohoto dotazu. Tu lze následně pozměnit a odeslat jako nový dotaz. Zachycené HTTP zprávy však nesmí být změněny, neboť by uživateli mohla například přijít odpověď na jiný dotaz, než odeslal. Pokud dojde k pokusu o změnu zachycené zprávy, je tato změna odmítnuta výjimkou (návrhový vzor *Balking*). Lze však vytvořit kopii zprávy a změnit tu.

SiteExplorer je komponenta, která monitoruje adresářovou strukturu testované aplikace. Obsahuje podčást *crawler*, která následuje odkazy v rámci testované aplikace a simuluje tak uživatelovu interakci s aplikací. Podporuje tak rychlejší a důkladnější testování.

IssueList slouží jako jednotné místo, kam detekční moduly hlásí nalezené zranitelnosti. Na požádání je schopen vytvořit výslednou zprávu o penetračním testu. Jednotlivým zranitelnostem je přidělena úroveň závažnosti z šesti možných úrovní. Jedná se o úrovně *False positive*, *Informational*, *Low*, *Medium*, *High* a *Critical*.

TrafficStorage umožňuje zaznamenat veškerou zachycenou HTTP komunikaci do databáze. Historii komunikace zpřístupňuje ostatním komponentám.

GUI zprostředkovává komunikaci s uživatelem. Dovoluje rovněž detekčním modulům, aby poskytly své vlastní uživatelské rozhraní. Pro komunikaci s ostatními částmi frameworku je opět použit návrhový vzor Pozorovatel (*Observer*).

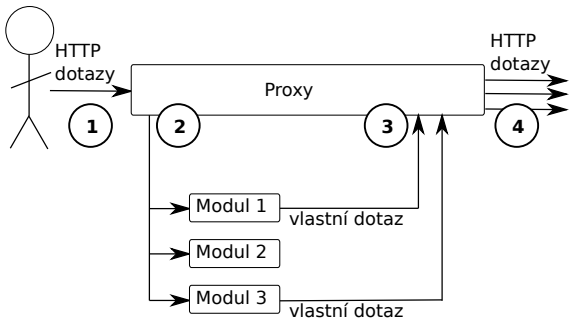
Module specifikuje rozhraní, které musí implementovat každý detekční modul. ModuleManager pak umožňuje vyhledávat detekční moduly, načítat je a předávat jednotlivým detekčním úlohám. Tento přístup k modularitě odpovídá návrhovému vzoru *Service locator*, který je formou vzoru obrácení řízení (*Inversion of Control*).

3.3.1 API

Aby detekční moduly mohly využívat funkce frameworku, bude třeba poskytnout API. Toto API představuje sadu rozhraní k jednotlivým komponentám frameworku. Poskytnutí pouze rozhraní a nikoli implementace posiluje dobrý návrh – kód pak bude tvořen vůči rozhraní a ne implementaci. To v budoucnu může usnadnit výměnu implementace za jinou.

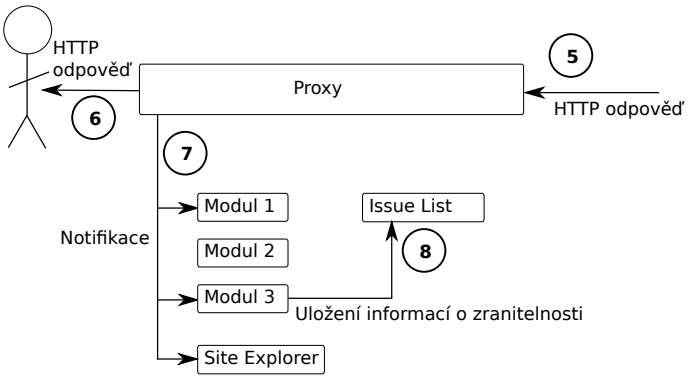
3.4 Průběh penetračního testu

Obrázky 3.2 a 3.3 znázorňují posloupnosti událostí, které proběhnou ve frameworku ve fázi testování při zachycení HTTP dotazu a odpovědi. Jedná se o následující události:



Obrázek 3.2: Posloupnost událostí při zachycení HTTP dotazu

1. Proxy zachytí HTTP dotaz.
2. Registrované komponenty frameworku a moduly jsou upozorněny, že byl zachycen nový dotaz.
3. Některé z modulů se na základě tohoto dotazu rozhodnou odeslat své vlastní dotazy.
4. Všechny dotazy jsou odeslány svým adresátům.



Obrázek 3.3: Posloupnost událostí při zachycení HTTP odpovědi

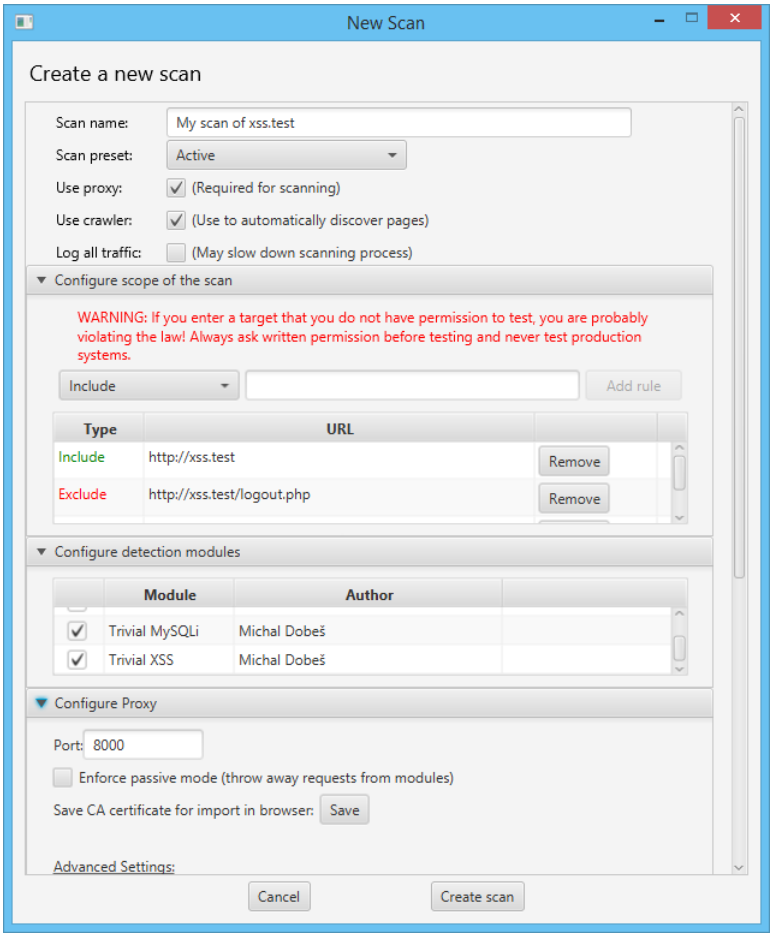
5. Je přijata odpověď na dotaz.
6. Pokud se jedná o odpověď na uživatelův dotaz, je tato odpověď přeposlána uživateli.
7. Registrované komponenty frameworku a moduly jsou upozorněny, že byla zachycena nová odpověď. Mezi nimi i komponenta *Site Explorer*, která mapuje adresářovou strukturu testované aplikace.
8. Pokud modul na základě přijaté HTTP odpovědi nalezne zranitelnost, oznámí tuto skutečnost komponentě *Issue List*.

3.5 Grafické uživatelské rozhraní

Jedním z požadavků na řešení je grafické uživatelské rozhraní (GUI), pomocí kterého bude uživatel komunikovat s frameworkem. GUI je navrženo s ohledem na přehlednost a uživatelskou přívětivost. Po spuštění aplikace je zobrazena uvítací obrazovka, která uživateli pomůže vytvořit novou skenovací úlohu či načíst dříve uloženou úlohu.

3.5.1 Vytvoření nové úlohy

Průvodce vytvořením nové úlohy je zobrazen na obrázku 3.4. Umožňuje nastavit jméno skenovací úlohy, cílovou aplikaci, použité detekční moduly i další komponenty skenovací úlohy. Uživatel například může zvolit hodnoty, na které mají být nastaveny HTTP hlavičky před odesláním.



Obrázek 3.4: Vytvoření nové skenovací úlohy. Výšky tabulek byly pro tuto ukázku sníženy.

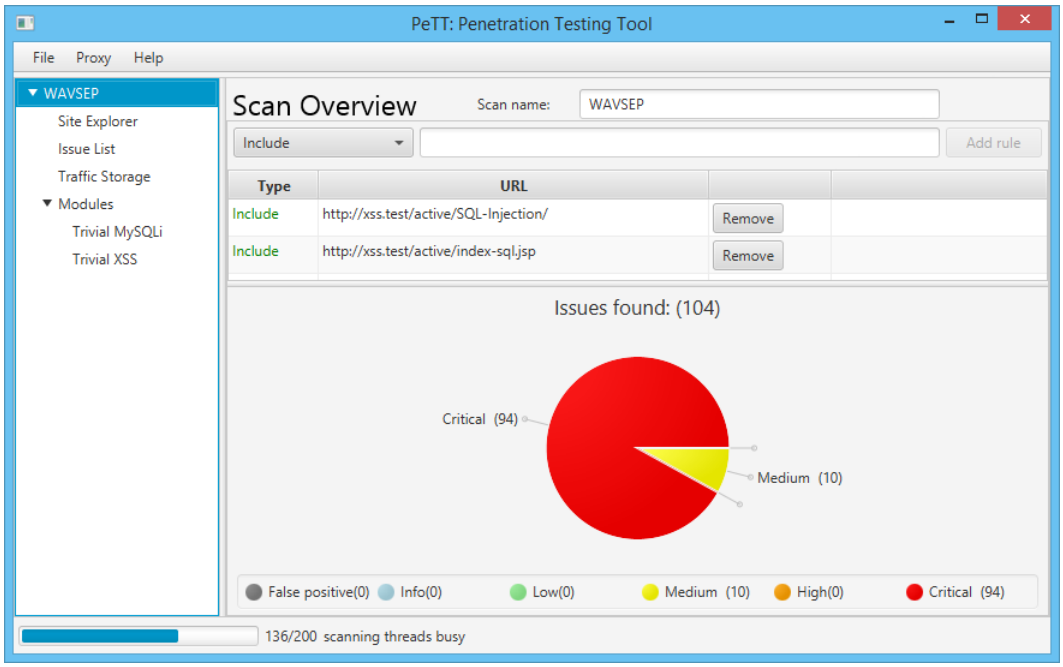
Cílovou aplikaci může uživatel definovat sadou pravidel typu *Include* (zahrň adresu jako cíl) a *Exclude* (vyjmi adresu z cíle). V příkladu na obrázku 3.4 je cíl tvořen doménou `http://xss.test/`, vyjma stránky `http://xss.test/logout.php`. Vyjmutím stránky `logout.php` lze zabránit nechtěnému odhlášení uživatele během automatického skenování.

Chybné adresy jsou při zadávání označeny červenou barvou a není možno je přidat do definice cíle. Pokud u přidávané adresy není uveden protokol, je automaticky doplněn na `http://`. Nad definicí cíle je uvedeno varování, že skenováním aplikací bez svolení jejich majitele může dojít k porušení zákona.

Průvodce umožňuje nastavení nové úlohy urychlit zvolením jednoho z předpřipravených profilů (*Scan preset*). Výběr aktivního testu zvolí doporučené detekční moduly a nastaví vybrané komponenty frameworku tak, aby detekční moduly mohly testované aplikaci odesílat vlastní dotazy. Výběr pasivního módu naopak zakáže modulům odesílat vlastní dotazy.

3.5.2 Hlavní okno a přehledová obrazovka

Hlavní okno aplikace je zobrazeno na obrázku 3.5. Je rozděleno na dvě části. Levá část obsahuje seznam skenovacích úloh. Pro každou skenovací úlohu jsou v seznamu uvedeny její komponenty. Kliknutím na danou komponentu se její GUI zobrazí v pravé straně okna. Ve spodní liště hlavního okna se nachází indikátor množství vláken, která jsou aktuálně zaneprázdněna vyhledáváním zranitelností.



Obrázek 3.5: Obrazovka s přehledem o skenovací úloze.

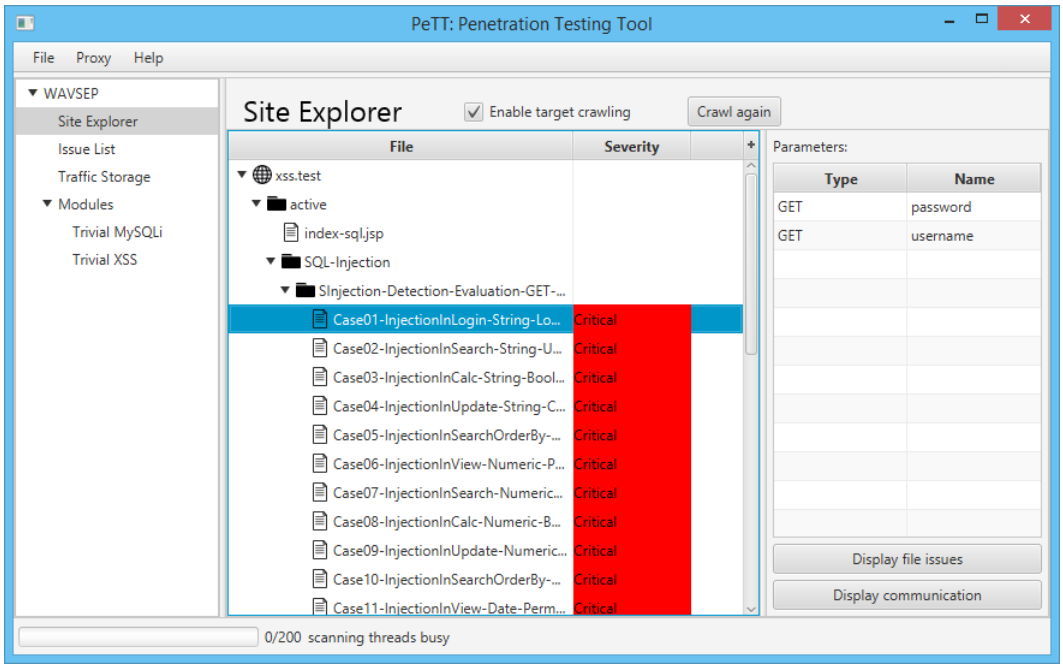
Na obrázku 3.5 je zobrazena přehledová obrazovka. Ta je zobrazena při přidání nové úlohy a také při kliknutí na název úlohy. Její součástí je definice testované aplikace a graf zobrazující počet nalezených zranitelností podle úrovně závažnosti. Na této obrazovce lze také změnit jméno úlohy.

3.5.3 Průzkumník adresářové struktury

Obrázek 3.6 ukazuje obrazovku komponenty *Site Explorer*. Adresářová struktura testované aplikace je zde zobrazena ve formě stromu. Pokud u některé položky byly objeveny zranitelnosti, je tato položka označena závažností podle nejzávažnější nalezené zranitelnosti.

V záhlaví komponenty je možnost povolit či zakázat automatické procházení stránek testované aplikace (*Enable target crawling*).

Po kliknutí na položku stromu je na pravé straně okna zobrazen její detail. Rovněž lze zobrazit zaznamenanou komunikaci a nalezené zranitelnosti pro vybranou stránku pomocí tlačítek *Display communication* a *Display file issues*.



Obrázek 3.6: Náhled na adresářovou strukturu testované aplikace.

3.5.4 Seznam nalezených zranitelností

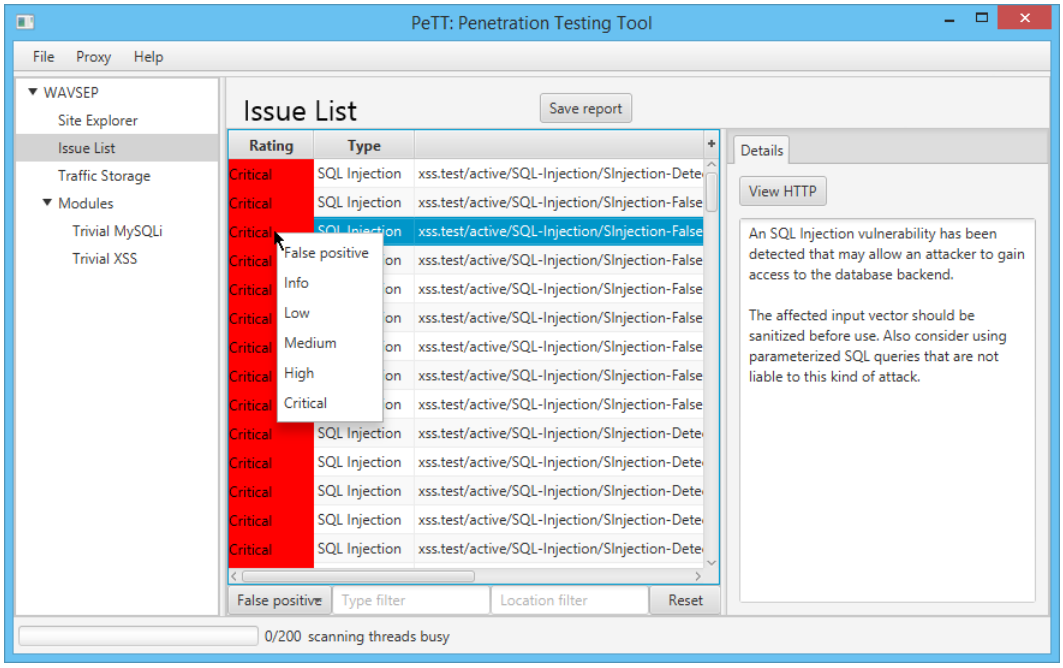
Obrázek 3.7 zachycuje grafické rozhraní komponenty *Issue List*. To obsahuje tabulku s nalezenými zranitelnostmi. Jednotlivé stupně závažnosti jsou barevně označeny. Závažnost jednotlivých záznamů lze změnit kliknutím pravého tlačítka na úroveň závažnosti. Uživatel je o této možnosti informován při prvním zobrazení tabulky.

Kliknutím na zranitelnost dojde k zobrazení jejího detailu v pravé části okna. Tento detail obsahuje obecné informace o daném typu zranitelnosti a také doporučení, jak zranitelnost opravit. Detail také obsahuje možnost zobrazit HTTP komunikaci, ve které byla zranitelnost nalezena. V zobrazené komunikaci jsou vyznačeny části, na jejichž základě byla detekována zranitelnost.

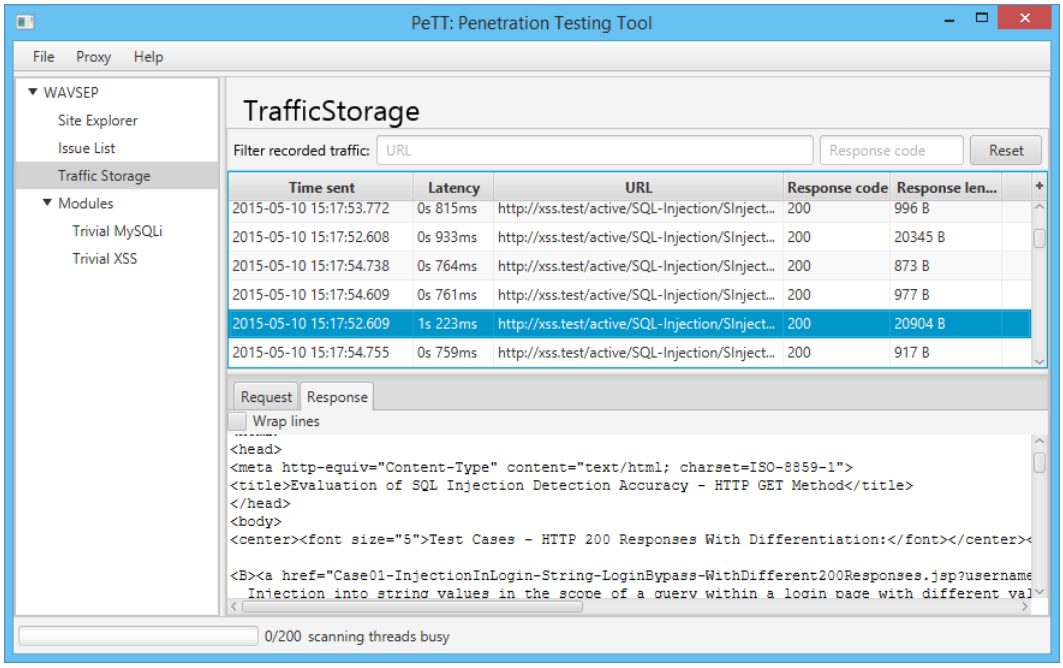
Pod tabulkou se nachází filtr zranitelností. Ten uživateli umožňuje například zobrazit pouze zranitelnosti se závažností vyšší než zvolená závažnost. Také umožňuje filtrovat zranitelnosti podle typu a podle adresy stránky, kde byly nalezeny.

3.5.5 Záznam komunikace

Na obrázku 3.8 je uvedena obrazovka komponenty *Traffic Storage* se záznamem zachycené komunikace. Ta umožňuje uživateli získat detailní informace o odeslaných a přijatých HTTP zprávách. Každý řádek tabulky odpovídá dvojici dotaz-odpověď.



Obrázek 3.7: Seznam nalezených zranitelností.



Obrázek 3.8: Uživatelské rozhraní pro zobrazení historie komunikace.

Pro každou dvojici dotaz-odpověď jsou v tabulce uvedeny informace o čase, kdy byl odeslán dotaz, o době, která uběhla mezi odesláním dotazu a přijetím odpovědi, o cílové adrese dotazu, o typu odpovědi (*response code*) a o délce odpovědi. V oblasti pod tabulkou je zobrazeno přesné znění dotazu i odpovědi pro vybraný řádek tabulky.

Jelikož je během testu obvykle zachyceno veliké množství zpráv, umožňuje rozhraní filtrovat zobrazenou komunikaci. Kritéria pro filtrování může uživatel zadat prostřednictvím polí nad tabulkou. Filtrovat lze podle adresy v HTTP dotazu či podle typu HTTP odpovědi. Informace v tabulce lze navíc řadit podle jednotlivých sloupců. Filtrování a řazení může uživateli pomoci objevit například komunikaci s příliš dlouhou odezvou, což může být využito pro vyhledávání dalších zranitelností na základě anomálního chování testované aplikace.

3.6 Návrh detekčních modulů

Vytvoření detekčních modulů má za cíl prokázat, že vytvořený framework je použitelný pro svůj účel. Vytvořené moduly mohou rovněž sloužit jako vodítko pro implementátory dalších modulů. Z toho vyplývá, že by se mělo jednat spíše o jednoduché příklady než o rozsáhlá díla. Vytvoření modulů, které by do hloubky prověřovaly jednotlivé zranitelnosti, je mimo rozsah této práce.

3.6.1 Modul XSS

Modul pro detekci cross site scripting se bude inspirovat postupem uvedeným v patentu společnosti Microsoft[10][2]. Bude fungovat ve dvou fázích, které se časově mohou prolínat. V první fázi bude modul do parametrů jednotlivých HTTP dotazů vkládat tzv. *tracery*. Tracer je řetězec, který je pro každý parametr webové aplikace jedinečný. Pokud pak následně v některé z odpovědí webové aplikace takovýto tracer objevíme, jsme schopni zjistit, kterým HTTP dotazem byl do webové aplikace vložen a ve které HTTP odpovědi se objevil.

K vyhledávání tracerů slouží druhá fáze. Ta má rovněž za cíl ověřit, zda je místo traceru do webové stránky možno vložit útočný řetězec. Do HTTP dotazu, který byl použit k vložení traceru, zkusí postupně umístit předpřipravené útočné řetězce. Pokud některý z řetězců vyvolá očekávanou odezvu na vhodném místě odpovědi, je nalezena zranitelnost Cross-Site Scripting.

Typ zranitelnosti lze určit podle toho, zda se vložený tracer objevil přímo v odpovědi na HTTP dotaz, který jej do aplikace vložil. Pokud ano, jedná se o Reflected Cross-Site Scripting. Pokud ne, jde o Stored Cross-Site Scripting.

3.6.2 Modul SQLI

Modul pro detekci SQL Injection bude implementovat algoritmus podobný základnímu algoritmu uvedenému v práci pana Bc. Matouše Kutypy[12]. Modul bude pracovat na předpokladu, že testovaná aplikace vkládá informace o chybách při vyhodnocování dotazů SQL do HTTP odpovědi. Rovněž bude předpokládat, že testovaná aplikace používá databázový server MySQL.

Detekce SQL Injection bude probíhat tak, že detekční modul do každého parametru HTTP dotazu vloží znak, který by měl způsobit syntaktickou chybu v SQL dotazu. Následně pro odpověď na takto pozměněný dotaz spočítá skóre výskytu klíčových slov jako „database“

[2]Tento patent byl stažen v roce 2012.

či „query“. Taková slova bývají obsažena v chybových hlášeních. Pokud skóre přesáhne skóre původní odpovědi o určitou hodnotu, je detekována zranitelnost SQL Injection.

Modul pro detekci SQLI završuje návrh nástroje pro penetrační testování. Následující kapitola popisuje implementaci tohoto nástroje.

Kapitola 4

Implementace

Projekt je rozdělen na část API a implementaci tohoto API. API až na výjimky obsahuje pouze rozhraní bez implementace. Použití tohoto přístupu zapouzdřuje vnitřní implementaci a zvyšuje přehlednost kódu. To vede k lepší udržovatelnosti a jednodušší tvorbě modulů.

4.1 Jazyk

Pro implementaci byl použit jazyk Java SE 8, ke spuštění aplikace je tedy zapotřebí Java Runtime Environment alespoň ve verzi 8. GUI bylo realizováno v JavaFX 8, což je doporučováno pro vývoj nových aplikací. Jelikož je Java 8 v době psaní tohoto textu ještě poměrně nový standard, nemusí být plně podporována na všech platformách. Tato situace se však s časem bude měnit. Výsledná aplikace je s použitím Oracle Java Runtime Environment 8 plně funkční jak na OS Linux, tak na Windows.

4.2 Metoda vývoje

Pro vývoj byl zvolen iterativní přístup, při kterém je nejprve vytvořen základ projektu a postupně je přidávána další funkcionalita. Výsledky jednotlivých fází byly testovány, zda splňují požadovanou funkcionalitu. Tento přístup umožňuje získat funkční aplikaci dříve než při vodopádovém modelu.

V první iteraci byl vytvořen základ GUI se správou skenovacích úloh *ScanTaskManager*. Následně byla implementována proxy. Poté následovala komponenta *SiteExplorer*, dále *IssueList*. V závěru byly implementovány moduly pro detekci zranitelností XSS a SQLI.

4.3 Struktura balíčků

Balíčky jazyka Java podporují lepší organizaci kódu, proto jsou použity i v tomto projektu. Kořenový balíček se jmenuje `pett`[1]. Projekt je rozdělen do následujících balíčků:

- `pett.api` představuje API, je nutný pro tvorbu modulů.
- `pett.core.apiimpl` obsahuje vnitřní implementaci API, jedná se o jádro frameworku.
- `pett.core.fxgui` obsahuje GUI a logiku s ním spojenou.
- `pett.modules` sdružuje implementované detekční moduly pro XSS a SQLI.

[1] Penetration Testing Tool

4.4 API

API bylo vyvinuto jako oddělená knihovna. Díky tomu jsou dále skryty implementační detaily. Rovněž není při vývoji nového modulu třeba odkazovat balíček s jádrem frameworku. API je opatřeno komentáři a je pro něj vytvořena dokumentace ve formátu javadoc. Anotace javadoc také značně usnadňují tvorbu modulů, pokud tvůrce využívá prostředí s podporou inteligentního napovídání kódu. Jména rozhraní a jejich metod byla volena s důrazem na intuitivnost, podle principu „*Turn comments into code*“. API je rozděleno do několika balíčků, které odpovídají navrhované struktuře frameworku (viz diagram tříd na obrázku D.1). Výsledný diagram tříd pro API je uveden na obrázku E.1.

4.4.1 Balíček scantask

Balíček `scantask` obsahuje rozhraní skenovací úlohy (*ScanTask*). Toto rozhraní umožňuje mimo jiné zjistit jméno a cíl úlohy. Rovněž umožňuje získat přístup k ostatním komponentám dané úlohy, například metodou *getIssueList()* k seznamu zranitelností.

Podle návrhového vzoru Pozorovatel (*Observer*) upozorňuje *ScanTask* registrované posluchače, pokud je přidán či odebrán některý z modulů. Tato upozornění umožňují, aby o sobě moduly měly navzájem povědomí a měly tak možnost spolupracovat. Metody pro registraci a odregistrování posluchačů jsou v rozhraní taktéž deklarovány.

ScanTask dědí z rozhraní *java.io.Externalizable*, což jeho implementátorům propůjčuje schopnost serializace podle návrhového vzoru Memento. Ta je využita, pokud si uživatel přeje uložit aktuální stav úlohy do souboru, aby mohl v testování pokračovat později.

4.4.2 Balíček http

Balíček `http` poskytuje rozhraní, která usnadňují práci se zprávami protokolu HTTP. Rozhraní *HTTPMessage* představuje obecnou zprávu protokolu HTTP. Metodou *getBytes()* lze získat její binární reprezentaci, ta se ale obvykle nehodí pro další zpracování. Proto rozhraní *HTTPMessage* deklaruje další metody, které abstrahují práci s HTTP zprávou. Prostřednictvím metod *getHeaders()* a *getHeadersByName()* lze přistupovat k hlavičkám dané zprávy. Hlavičky jsou reprezentovány objekty *HTTPHeader*. Pro získání textové reprezentace HTTP zprávy slouží metody *toString()* a *getBodyAsString()*.

Rozhraní *HTTPRequest* a *HTTPResponse* dědí z rozhraní *HTTPMessage* a představují HTTP dotaz a odpověď. Rozhraní *HTTPRequest* rozšiřuje obecnou HTTP zprávu o další potřebnou funkcionalitu. Lze tak měnit metodu i adresáta dotazu, jakožto i libovolně nastavovat GET a POST parametry. Pro správu POST parametrů je využit návrhový vzor *Balking* – pokud již byl dotaz odeslán, jeho změna by vnesla do aplikace nekonzistenci. Proto je při takovém pokusu generována výjimka. Rozhraní dále naznačuje reimplementaci metod *equals()* a s ní spojené *hashCode()*, aby bylo možno porovnávat jednotlivé HTTP dotazy na shodu.

HTTPResponse představuje HTTP odpověď. Obecnou HTTP zprávu sice nerozšiřuje o další funkcionalitu, nabízí však místo, do kterého by v další verzi frameworku mohla být funkcionalita přidána, aniž by došlo k narušení zpětné kompatibility.

HTTPCommunication představuje dvojici dotaz-odpověď, a to včetně informace o délce odezvy testované aplikace na daný dotaz. Informace o délce odezvy je velmi užitečný údaj, který lze využít například pro detekci zranitelnosti blind SQL Injection. Mezi dalšími abstrakcemi nabízí balíček `http` také rozhraní pro HTTP parametr, hlavičku či pro webovou stránku.

Ve všech případech se však jedná o rozhraní, nelze v nich tedy definovat konstruktory objektů. Proto balíček `http` obsahuje podle návrhového vzoru *Factory* také rozhraní *HTTPFactory*. Toto rozhraní prostřednictvím továrních metod umožňuje uživateli frameworku vytvářet vlastní HTTP hlavičky, parametry i dotazy.

4.4.3 Balíček proxy

Balíček `proxy` obsahuje rozhraní komponenty proxy, výjimky, které může proxy generovat, a rozhraní posluchače HTTP dotazů a posluchače HTTP odpovědí. Rozhraní *Proxy* umožňuje modulům registrovat vlastní posluchače. Tito posluchači musí implementovat rozhraní *HTTPRequestListener* či *HTTPResponseListener*. Když pak proxy obdrží nový HTTP dotaz, vyhodnotí jeho cíl a upozorní ty registrované posluchače dotazů, pro které cíl dotazu odpovídá testované aplikaci. Těmto posluchačům je původní dotaz předán. Když proxy obdrží novou HTTP odpověď, vyhodnotí původce odpovědi a uvědomí ty z registrovaných posluchačů odpovědí, pro které původce odpovědi odpovídá testované aplikaci. Těm bude předán objekt implementující rozhraní *HTTPCommunication*. Z takového objektu lze pak získat HTTP dotaz i odpověď, jakožto i informaci o délce odezvy. Díky tomuto přístupu jsou uvědomovány pouze ty moduly, které stojí o daný typ upozornění.

Rozhraní *Proxy* také deklaruje metody, které modulům umožňují odesílat vlastní HTTP dotazy. K tomu slouží metody *sendRequest()* a *sendRequestWithPublicResponse()*. V případě metody *sendRequest()* odešle proxy předaný HTTP dotaz a po získání odpovědi uvědomí posluchače odpovědí, který je této metodě rovněž předán. V případě metody *sendRequestWithPublicResponse()* jsou uvědoměni všichni registrovaní posluchači odpovědí, pro které původce odpovědi odpovídá testované aplikaci. Dotazy jsou však odeslány pouze pokud je jejich cílem testovaná aplikace. Díky tomu je omezena možnost útoku na jinou než testovanou aplikaci.

Pokud uživatel uvede proxy do tzv. *pasivního režimu*, neumožní proxy odeslání žádného vlastního dotazu a bude pouze přeposílat uživatelovu komunikaci. Toto nastavení je vhodné, pokud se uživatel chce ujistit, že při penetračním testu nedojde k útočným dotazům na testovanou aplikaci. Pokud se modul pokusí o odeslání vlastního dotazu v pasivním režimu, proxy generuje výjimku *PassiveModeException*. Při pokusu o odeslání vlastních dat, když je proxy vypnuta, je generována výjimka *ProxyOffException*. Toto chování odpovídá návrhovému vzoru *Balking*.

4.4.4 Balíček siteexplorer

Balíček `siteexplorer` obsahuje rozhraní komponenty Site Explorer a posluchače (*listener*) této komponenty. Rozhraní *SiteExplorer* zprostředkovává modulům informace o adresářové struktuře testované aplikace prostřednictvím metody *getSiteFiles()*. Rovněž deklaruje metodu *addFile()*, která umožňuje přidat nově nalezenou webovou stránku. Webové stránky lze vyhledávat podle jejich URL pomocí metody *getWebfileByURL()*.

Stejně jako v předchozích případech, i rozhraní *SiteExplorer* podporuje registraci posluchačů. Ti jsou uvědomováni, pokud dojde k přidání či změně některé webové stránky. Díky tomu mohou detekční moduly reagovat na nově objevené webové stránky a na aktualizace informací.

4.4.5 Balíček issues

Balíček `issues` obsahuje rozhraní komponenty Issue List, rozhraní posluchače této komponenty, rozhraní pro filtrování zranitelností a abstraktní třídu reprezentující zranitelnost. Rozhraní *IssueList* deklaruje metody pro přidávání nalezených zranitelností a pro získání jejich seznamu. Stejně jako u ostatních komponent, i zde je použit návrhový vzor *Observer*. Díky němu mohou komponenty získávat upozornění, když přibude nová zranitelnost či když je některá změněna.

Abstraktní třída *Issue* reprezentuje zranitelnost. Každý detekční modul tak může definovat vlastní třídu, která reprezentuje zranitelnost. Tento přístup byl zvolen, jelikož dopředu nelze předpokládat všechny typy zranitelností, pro které budou v budoucnu vytvořeny detekční moduly. Zvolené řešení umožňuje snadnou rozšiřitelnost o nové typy zranitelností.

Issue implementuje funkcionalitu, která bude s velkou pravděpodobností společná pro většinu typů zranitelností. Umožňuje získat HTTP dotaz, který úspěšně zaútočil na webovou aplikaci a objekt s rozhraním *HTTPCommunication*, ve kterém se projevily následky útoku. V HTTP komunikaci umožňuje zvýraznit části, na základě kterých byla zjištěna zranitelnost a označit parametry, které byly zranitelností ovlivněny. U každé zranitelnosti je určen stupeň závažnosti z výčtového typu *Severity*. Navíc *Issue* deklaruje abstraktní metody *getDescription()* a *getRecommendations()*, pomocí kterých je uživatel informován o principu zranitelnosti a o tom, jak danou zranitelnost odstranit.

Některé komponenty frameworku či detekční moduly mohou mít zájem získat seznam nalezených zranitelností, které splňují určité kritérium. K tomu slouží rozhraní *IssueSelector*, jehož implementaci lze předat metodě *getIssues()* rozhraní *IssueList*. Pro zranitelnost, která je předána jeho metodě *willBeSelected()*, rozhodne, zda má tato zranitelnost být obsažena ve vráceném seznamu.

4.4.6 Balíček trafficstorage

Tento balíček obsahuje rozhraní *TrafficStorage* pro přístup k záznamu komunikace. Obsahuje také rozhraní posluchače této komponenty, *TrafficStorageChangeListener*.

4.4.7 Balíček modules

Balíček `modules` obsahuje rozhraní *Module*, které musí být implementováno každým modulem. Kromě metod pro získání jména modulu a autora je zde deklarována metoda *setScanTask()*. Jejím zavoláním je modul přiřazen ke skenovací úloze a může tak registrovat své posluchače u příslušných komponent frameworku. Tento postup odpovídá návrhovému vzoru *Inversion of Control*. Metoda *cleanup()* je naopak volána před odebráním modulu ze skenovací úlohy. Slouží k tomu, aby modul uvolnil alokované prostředky, které alokoval. Prostřednictvím metody *getGUIComponentProvider()* může modul poskytnout vlastní GUI (viz balíček `gui`). Stejně jako rozhraní *ScanTask*, i rozhraní *Module* dědí z rozhraní *java.io.Externalizable*, díky čemuž mohou moduly uložit svá data, zároveň se skenovací úlohou.

4.4.8 Balíček gui

Balíček `gui` obsahuje rozhraní *GUIComponentProvider*, které slouží k dalšímu rozšíření GUI poskytovaného frameworkem. Jedná se o jediné rozhraní v API, které je přímo vázáno na grafickou platformu JavaFX. Tato vazba je realizována metodou *getPane()*, jejíž návratový

typ je *javafx.scene.layout.Pane*. Pokud by v budoucnu bylo třeba, je prostřednictvím tohoto rozhraní možno podporovat i jiné platformy pro uživatelské rozhraní. Metoda *getScan()* spojuje poskytované uživatelské rozhraní s příslušnou skenovací úlohou.

4.5 Framework

Vlastní jádro frameworku je obsaženo v balíčku `pett.apiimpl`, jehož struktura odpovídá struktuře balíčku `pett.api`. `pett.apiimpl` obsahuje implementace rozhraní z API a další objekty, které zajišťují funkcionalitu frameworku. Jména tříd, které implementují jednotlivá rozhraní z API, končí řetězcem *Impl*. To usnadňuje orientaci ve zdrojovém kódu. Budou přiblíženy implementace rozhraní z balíčku HTTP, komponenty Proxy, komponenty pro záznam komunikace a modulárního systému.

4.5.1 Implementace zpráv HTTP

Implementace elementů HTTP zpráv jsou připraveny na nasazení v paralelním prostředí. Implementačně zajímavé jsou třídy *HTTPMessageAbstract* a *HTTPRequestImpl*, které budou nyní přiblíženy.

Třída *HTTPMessageAbstract* implementuje obecnou zprávu protokolu HTTP (rozhraní *HTTPMessage*). Jedná se o abstraktní třídu, která sdružuje funkcionalitu společnou pro HTTP dotaz i odpověď. Především umožňuje objekt HTTP zprávy inicializovat binárními daty, která byla přijata po síti. K tomu slouží metoda *initFromStream()*, která je využívána konstruktory tříd *HTTPRequestImpl* a *HTTPResponseImpl*. *HTTPMessageAbstract* dále zprostředkovává přístup k hlavičkám zprávy a také reimplementuje metodu *toString()*, která celou HTTP zprávu převádí do podoby řetězce. Při tomto převodu je brána v úvahu znaková sada zprávy, která je zjištěna z hodnoty hlavičky `Content-Type`.

Třída *HTTPRequestImpl* dědí z třídy *HTTPMessageAbstract*. Implementuje rozhraní HTTP dotazu a umožňuje tak měnit hodnoty hlaviček i GET a POST parametrů. Parametry POST může podle specifikace HTTP[9] obsahovat pouze dotaz HTTP metodou `POST`. Metody, které manipulují s parametry POST tedy nemají účinek, pokud HTTP metoda dotazu není `POST`. Tento přístup odpovídá návrhovému vzoru *Balking*.

Existuje však několik způsobů, jakými mohou být do těla dotazu vloženy POST parametry. Jako abstrakce těchto způsobů slouží třídy v podbalíčku `postdecoders`. Třída *PostDecoderSelector* vybere na základě hodnoty hlavičky `Content-Type` vhodný kodek[2] pro čtení a zápis parametrů POST do těla HTTP dotazu. Při změně parametrů POST je automaticky přepočtena délka zprávy a uložena do hlavičky `Content-Length`. Programátor detekčních modulů tedy tuto úpravu již nemusí provádět. Pokud by v rámci detekce zranitelností bylo třeba úmyslně odeslat zprávu se špatnou hodnotou hlavičky `Content-Length`, lze tak po nastavení POST parametrů učinit prostou změnou hodnoty této hlavičky.

Aby bylo zabráněno úpravám HTTP dotazu po jeho odeslání, poskytuje implementace HTTP dotazu metodu *setLocked()*, pomocí které lze objekt *HTTPRequestImpl* zamknout. Pokud je objekt zamknut, pokusy o jeho změnu vyvolají výjimku.

Metoda *getFingerprint()* umožňuje získat „otisk“ HTTP dotazu. Jedná se o řetězec, který obsahuje parametry tohoto dotazu (URL, GET, POST a hlavičky). Tvůrce detekčních modulů může potřebovat zjistit, zda již byl odeslán dotaz s podobnými parametry. Místo aby modul ukládal všechny odeslané dotazy, může ukládat pouze jejich otisky či jen hashe těchto otisků. Tím lze dosáhnout značné úspory paměťového prostoru.

[2] Kodér-dekodér, třídu implementující rozhraní *HTTPPostDecoder*.

4.5.2 Implementace proxy

V rámci implementace bylo prozkoumáno mnoho opensource implementací zachytávacích proxy v Javě, avšak žádná zcela nevyhovovala účelům penetračního testování. Proto byla zachytávací proxy implementována od základu. Výsledná implementace podporuje jak zachytávání nešifrovaných dat protokolu http, tak zachytávání šifrovaných dat, která jsou posílána protokolem http přes SSL.

Podporováno je velké množství současně otevřených spojení, přičemž jejich nejvyšší počet je možné nastavit. Pro správu vláken obsluhujících jednotlivá spojení je použita upravená třída *ProxyExecutor*. Ta dědí ze třídy *java.util.concurrent.ThreadPoolExecutor*. Tento upravený exekutor uchovává seznam běžících úloh, který může být použit k předání zprávy všem aktuálně běžícím úlohám. Tento přístup je použit v případě vypnutí proxy, kdy je vyžadováno, aby všechny běžící úlohy urychleně ukončily svůj běh.

Proxy očekává spojení na zadaném portu a při zachycení HTTP dotazu tento požadavek analyzuje a přepošle určenému příjemci. Pokud se jedná o dotaz HTTP metodou `CONNECT`, znamená to, že budou následovat data šifrovaná pomocí SSL. Pokud je tedy třeba dále analyzovat data, která tečou mezi prohlížečem uživatele a cílovou aplikací, musí být takto šifrovaná data dešifrována.

K dešifrování dat, která tečou ve spojení SSL, je třeba napodobit chování cílového serveru a ustavit spojení s uživatelovým prohlížečem pomocí falešného ssl certifikátu. Nelze však použít jediný certifikát pro všechny servery testované aplikace, moderní prohlížeče[3] takový postup označí jako porušení bezpečnosti a další komunikaci zablokují. Tento certifikát tedy musí být vytvořen za běhu. Pro vytváření certifikátů je využita knihovna BouncyCastle[4], která danou funkcionalitu poskytuje. Takto vytvořený certifikát však musí být dále podepsán důvěryhodnou certifikační autoritou, aby mu prohlížeč důvěřoval. Proto byl vytvořen certifikát certifikační autority, který je třeba importovat do prohlížeče. Soukromý klíč tohoto certifikátu je obsažen v komponentě proxy a je používán k podepisování za běhu vytvářených certifikátů.

Jelikož Java ve své třídě *SSLSocketFactory* neumožňuje používat různé certifikáty pro různá spojení, bylo třeba implementovat třídu *SSLSocketFactoryGenerator*, která podle URL cílové webové stránky vybere instanci *SSLSocketFactory* s příslušným certifikátem, v případě potřeby vytvoří novou instanci[5]. Jedná se o návrhový vzor *Multiton*.

Při zachycení uživatelova požadavku je tento požadavek uzamknut metodou *setLocked()*, aby detekční moduly neměly možnost měnit jeho obsah. Pokud detekční modul potřebuje odeslat pozměněnou verzi takového požadavku, musí vytvořit kopii (metodou *copy()*) a odeslat tu. Pokud se detekční modul pokusí změnit zamknutý požadavek, je vyvolána výjimka. Tímto je zajištěno, že uživatel obdrží odpověď na svůj původní požadavek.

4.5.3 Implementace komponenty pro záznam komunikace

Jelikož záznam veškeré komunikace s testovanou aplikací může dosáhnout vysokého objemu, je zachycená komunikace ukládána do databáze v souborovém systému. Tento přístup sice vlivem diskových operací zpomalí skenovací proces, zato však umožní uložit veliké množství dat. Implementace využívá databázového systému Apache Derby[6]. Při vzniku komponenty

[3] Zkoušeno s Firefox 36 a Chromium 41.

[4] Knihovna je dostupná na http://www.bouncycastle.org/download/bcprov-jdk15on-151.jar.

[5] Implementace byla inspirována řešením na adrese https://alesaudate.wordpress.com/2010/08/09/how-to-dynamically-select-a-certificate-alias-when-invoking-web-services/ a rozšiřuje jej.

[6] Dostupný na https://db.apache.org/derby/.

TrafficStorageImpl je vytvořena dočasná databáze. Při uložení skenovací úlohy je pak tato databáze exportována a uložena spolu se skenovací úlohou.

Komponenta *TrafficStorage* poskytuje ostatním komponentám možnost získat přístup k zaznamenané komunikaci. K tomuto účelu byly vytvořeny třídy *DBCommWrapper*, *DBRequestWrapper* a *DBResponseWrapper*, které implementují rozhraní *HTTPCommunication*, *HTTPRequest* a *HTTPResponse*.

Tyto třídy jsou předávány ostatním komponentám a modulům namísto výše zmíněných tříd *HTTPRequestImpl* a *HTTPResponseImpl*. Uchovávají metadata o uložených zprávách, včetně databázových identifikátorů. Pokud jiná část frameworku přistupuje pouze k metadatům, například k času odeslání zprávy, není třeba získávat data z databáze. Pokud jsou vyžádána jiná data, například obsah zprávy, zajistí výše zmíněné *wrapper* třídy načtení dat z databáze a příslušné informace předají. Tento přístup přináší značné urychlení.

4.5.4 Implementace automatické komunikace

Generování automatické komunikace má za úkol ulehčit testerovi práci a rekurzivně za něj navštívit stránky a odeslat formuláře testované aplikace. Tato funkcionalita je implementována jako součást komponenty Site Explorer. Pokud je automatická komunikace povolena, žádá Site Explorer při každé zachycené HTTP odpovědi třídu *CrawlAnalyser* o seznam HTTP dotazů, které by měly být odeslány.

Třída *CrawlAnalyser* definuje metodu *generateCrawlRequests()*. Ta prohledá předanou HTTP odpověď a extrahuje z ní odkazy a formuláře. K extrakci je použita knihovna jsoup[7]. Na základě těchto odkazů jsou vytvořeny HTTP požadavky. Ty jsou předány komponentě proxy metodou *sendRequestWithPublicResponse()*, odpovědi na tyto dotazy tak jsou předány komponentám a detekčním modulům k analýze. Toto řešení umožňuje automaticky vyhledávat zranitelnosti.

Detekce některých typů zranitelností, například Stored Cross-Site Scripting, však vyžaduje, aby každá stránka byla navštívena vícekrát (viz dále). Po navštívení všech stránek testované aplikace je tak třeba navštívit všechny stránky podruhé. K tomu slouží metoda *recrawl()* rozhraní *SiteExplorer*.

4.5.5 Implementace modularity

K zajištění modularity je použito Java ServiceLoader API. Moduly jsou tvořeny jako knihovny jazyka Java a ve formě archivu `.jar` jsou umístěny do adresáře `modules` v adresáři frameworku. Aby ServiceLoader API detekovalo moduly, musí archivy těchto modulů obsahovat v balíčku `META-INF.services` informaci o tom, která třída archivu implementuje rozhraní *Module*. Při startu aplikace *ModuleManager* zjistí, které moduly jsou dostupné, a to tak, že přidá všechny `.jar` archivy z adresáře `modules` do *Java classpath*[8] a následně použije službu ServiceLoader. Pomocí ní vyhledá všechny třídy v *classpath*, které implementují rozhraní *Module*. Pro každou takovou třídu vytvoří jednu instanci, kterou si ponechá. Další instance detekčních modulů pak vznikají jako kopie těchto původních instancí. Rozhraní *Module* k tomuto účelu poskytuje metodu *copy()*.

[7] Dostupná na http://jsoup.org/packages/jsoup-1.8.1.jar.

[8] K dynamickému přidání archivů byl využit kód z http://solitarygeek.com/java/a-simple-pluggable-java-application.

4.5.6 Implementace uživatelského rozhraní

Při startu aplikace je vytvořen objekt správce skenovacích úloh a uživatelské rozhraní, které má ke správci přístup. Uživatel poté prostřednictvím GUI zažádá o vytvoření nové skenovací úlohy. Ta GUI rozšíří o GUI svých komponent a modulů.

Implementace GUI jednotlivých komponent frameworku se nachází v balíčku `pett.fx-gui`. Každé GUI je tvořeno souborem `.fxml`, který definuje grafické prvky a jejich rozmístění, a třídou, která slouží jako ovladač (*controller*) tohoto rozhraní. Ovladač definuje funkce, které mají být zavolány při daných akcích uživatelského rozhraní. Pro propojení s komponentami frameworku jsou při vytvoření ovladače u jednotlivých komponent zaregistrováni posluchači. Když je posluchač uvědoměn o změně, aktualizuje uživatelské rozhraní.

Pro uživatelské rozhraní byly použity ikony z webu `flaticon.com` pod licencí Creative Commons. Autoři jednotlivých ikon jsou Yannick a Freepik. Pro zobrazování HTTP komunikace s barevně vyznačenými důležitými částmi byla využita knihovna RichTextFX[9].

4.6 Modul XSS

Tento modul je obsažen v balíčku `pett.modules.trivialxss`. Rozhraní *Module* implementuje třída *TrivialXSSModule*. Po přiřazení ke skenovací úloze modul registruje u komponenty proxy dva posluchače HTTP odpovědí.

Prvním posluchačem je instance třídy *TracerInjector*. Ta pro každou zachycenou komunikaci upraví a odešle HTTP dotaz z této komunikace. Do parametrů tohoto dotazu postupně vloží tzv. *tracer*. Tracer je řetězec, který má nízkou pravděpodobnost výskytu ve stránce webové aplikace a ze kterého lze zjistit, do kterého parametru byl vložen. Jako tracer byl v případě tohoto modulu zvolen řetězec uvozený písmeny „XSSTRACER“ a následovaný šestnácti hexadecimálními číslicemi, které tvoří jeho identifikaci (*ID*). Prvních osm číslic ID je tvořeno spodními čtyřmi byty UNIXového času, ve kterém byl modul načten. Zbylých osm číslic je na začátku testu nulových. Hodnota ID traceru je pro každý použitý tracer inkrementována, každý tracer tedy má vlastní unikátní ID. K tomuto ID je zároveň v modulu vytvořen záznam, ze kterého lze zjistit, pro který parametr které stránky bylo použito. UNIXový čas byl zvolen proto, aby byl tracer unikátní i v případě, že bude aplikace testována vícekrát za sebou bez čištění databáze.

Druhým posluchačem je instance třídy *TracerDetector*. Ta v odpovědích testované aplikace vyhledává tracery. Pokud tracer nalezne, zavolá metodu *testExploits()* třídy *Exploiter*. Ta má za úkol ověřit, že nalezené spojení mezi HTTP parametrem a obsahem stránky lze zneužít pro útok XSS. Pokud byl tracer nalezen v odpovědi na dotaz, který tento tracer do aplikace vložil, je testováno Reflected XSS, v opačném případě Stored XSS. Podle umístění traceru v DOM modelu stránky vybere jednu ze tří sad útoků. Sada je tvořena dvojicemi řetězců. První ze dvojice je použit jako hodnota HTTP parametru. Pokud je pak druhý ze dvojice nalezen v testované odpovědi, je detekována zranitelnost XSS. Pro hodnocení závažnosti Reflected XSS byla zvolena úroveň *Medium*, pro Stored XSS úroveň *High*.

Třída *Exploiter* definuje dva posluchače HTTP odpovědí. Jedná se o vnitřní třídy *Verifier* a *ExploitReflector*. *Verifier* ověřuje, zda byl daný útok úspěšný. Je používán při testování Reflected i Stored XSS. *ExploitReflector* je použit pouze při testování Stored XSS. U toho je nejdříve odeslán dotaz, který vkládá útočný řetězec do testované aplikace. Odpověď na tento dotaz je předána *ExploitReflector*u. Jeho úkolem je odeslat dotaz, jehož

[9] Dostupná na `https://github.com/TomasMikula/RichTextFX`.

odpověď ověří úspěšnost útoku. Jako posluchač pro tuto odpověď je uveden *Verifier*. Z uvedeného postupu vyplývá, že pro detekci Stored XSS je třeba po vložení traceru navštívit stránku, na které se tracer objeví. Pokud byla navštívena před vložením traceru, je třeba ji navštívit znovu.

Pokud uživatel povolil záznam komunikace (komponenta *Traffic Storage*), lze všechny stránky navštívit znovu automatizovaně po stisknutí tlačítka v GUI modulu. Pokud záznam komunikace povolen není, je po stisknutí tohoto tlačítka volána metoda *recrawl()* komponenty *Site Explorer*. Ta ovšem není v průchodu testovanou aplikací tak dokonalá jako uživatel. Uživatel tak by tak měl stránky navštívit manuálně.

4.7 Modul SQLI

Tento modul je obsažen v balíčku `pett.modules.trivialsqli`. Rozhraní *Module* implementuje třída *TrivialSQLiModule*. Po přiřazení ke skenovací úloze modul registruje u komponenty posluchače odpovědí, který zachycenou komunikaci nechává analyzovat instancí třídy *Detector*. Ta pro každý zachycený požadavek odešle sérii požadavků, ve kterých je do hodnot jednotlivých parametrů dosazován znak apostrofu. Tento znak má za cíl způsobit syntaktickou chybu v dotazu SQL.

Odpovědi na upravené dotazy jsou analyzovány vnitřní třídou *ResponseAnalyzer*. Ta v odpovědích vyhledává klíčová slova, která se objevují v chybových hlášeních databázového serveru, a počítá jejich celkovou délku. Seznam klíčových slov byl převzat z bakalářské práce Bc. Matouše Kutypy[12]. Pokud skóre (celková délka nalezených klíčových slov) převyšuje skóre originální odpovědi o více než 15, je detekována zranitelnost SQLI. Hranice 15 byla určena experimentálně. Vzhledem k tomu, že zranitelnost SQLI může vést až ke kompletnímu ovládnutí serveru útočníkem, byla závažnost této zranitelnosti stanovena na nejvyšší stupeň, tedy *Critical*.

Kapitola 5

Testování

Aby byla ověřena funkčnost implementovaného řešení, byly jednotlivé komponenty frameworku podrobeny testům. Po dokončení bylo provedeno srovnání jak s komerčním řešením Burp Suite firmy Portswigger, tak s dalšími řešeními. Nástroj Burp Suite byl vybrán po dohodě s vedoucím práce.

5.1 Testování za účelem ověření funkčních požadavků

Jako celek byl framework testován na operačních systémech Arch Linux a Windows 8.1, za použití Oracle Java Runtime Environment 8. Na obou platformách byla aplikace stabilní a byla schopna zachytávat webový provoz a reagovat na něj.

5.1.1 Virtuální prostředí pro testování

Pro testování komponent, které mohou generovat útočné dotazy, bylo vytvořeno virtuální prostředí. Toto prostředí představují dva virtuální počítače propojené virtuální sítí oddělenou od internetu. Tento přístup značně omezuje možnost nechtěného útoku na cizí počítač v internetu. První počítač je vybaven systémem Windows 8.1 a je na něm nasazen vyvinutý penetrační nástroj *PeTT* (Penetration Testing Tool). Jeho cílem je útočit na druhý počítač.

Druhý počítač představuje webový server se zranitelnými aplikacemi. Je vybaven operačním systémem Arch Linux, webovým serverem Nginx 1.6.2 a databázovým serverem MariaDB 10.0.17[1]. Pro usnadnění přístupu k webovým aplikacím byl zprovozněn server DNS. Například ke stránce se zranitelností XSS tak lze přistoupit pomocí URL `http://xss.test`. Na tomto serveru byla rovněž nainstalována platforma WebGoat[14], která byla k testování také použita.

Pro účely testování modulu XSS byla implementována stránka se zranitelnostmi XSS. Tato stránka úmyslně obsahuje jak Reflected, tak Stored zranitelnosti XSS. Pro testování modulu SQLI byla na serveru nasazena zranitelná stránka vyvinutá Bc. Matoušem Kutypou v rámci jeho bakalářské práce[12]. Testování bylo prováděno za účelem ověření funkčních požadavků. Bylo prováděno ručně, za použití testovacích scénářů.

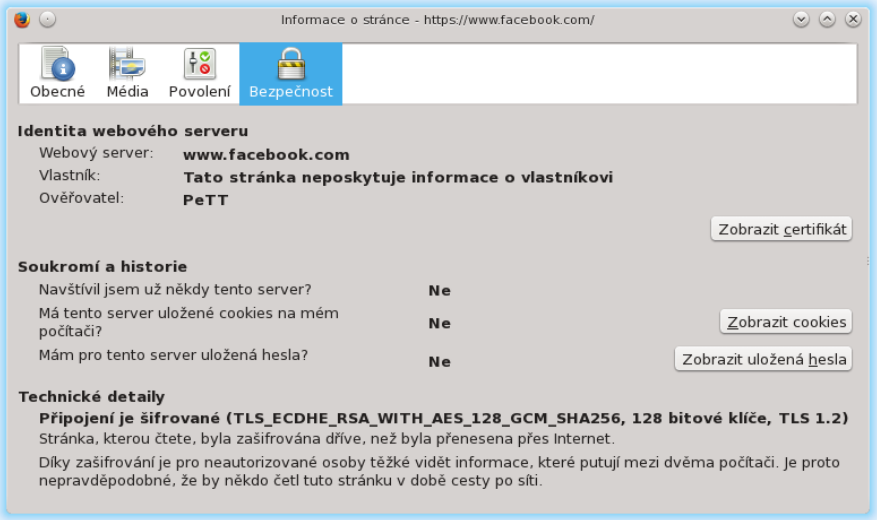
5.1.2 Komponenta proxy

Proxy, nejdůležitější komponenta frameworku, byla testována při reálném webovém provozu. K testování byl použit prohlížeč Firefox 36, který byl nastaven, aby se k internetu

[1] MariaDB je pokračovatel MySQL vybraný distribucí Arch Linux po převzetí MySQL společností Oracle.

připojoval přes tuto proxy. Proxy byla schopna zachytit a přeposlat komunikaci tak, že mezi procházením stránek s a bez proxy nebyl pozorovatelný rozdíl[2]. Při zachytávání HTTPS spojení prohlížeč přijal certifikát vytvořený komponentou proxy, což zachycuje obrázek 5.1. Během testu byla proxy schopna přenášet následující webový provoz:

- HTTP provoz při procházení webem www.fit.vutbr.cz.
- HTTPS provoz při procházení images.google.com.
- HTTPS provoz při používání web 2.0 aplikace facebook.com.
- HTTPS provoz při procházení youtube.com.



Obrázek 5.1: Informace o certifikátu při HTTPS spojení přes komponentu proxy.

5.1.3 Komponenta Issue List a modulární systém

Komponenta Issue List byla testována ve spojení s modulárním systémem. Byl vytvořen testovací detekční modul, který po stisknutí tlačítka simuloval detekci zranitelnosti. Tento modul zároveň prověřil i schopnost rozšíření GUI o uživatelské rozhraní modulu, jakožto i schopnost frameworku komunikovat s moduly. V rámci testovacího modulu byl také ověřen mechanismus ukládání dat – modul mohl uložit vlastní data a při opětovném načtení skenovací úlohy data načetl.

5.2 Porovnání s existujícími nástroji, včetně Burp Suite

Výslednou aplikaci bylo třeba porovnat s komerčním nástrojem Burp Suite. Společnost Portswigger, která nástroj Burp Suite vyvíjí, zdarma poskytuje pouze limitovanou verzi tohoto nástroje. Tato limitovaná verze ovšem nepodporuje detekci zranitelností. Byla odeslána žádost o zkušební licenci k plné verzi pro účely této práce, společnost však nereagovala.

[2] S výjimkou zpoždění při načítání videí na facebook.com.

Plnou verzi nástroje Burp Suite však *Sectoolmarket.com* zahrnul do průzkumu nástrojů pro penetrační testování[4], a to ve verzi 1.5.20. Tento průzkum byl prováděn na veřejně dostupné sadě testů *WAVSEP*[5] ve verzi 1.5. Pro porovnání vyvinutého nástroje s nástrojem Burp Suite byl tedy vyvinutý nástroj *PeTT* (Penetration Testing Tool) podroben sadě testů WAVSEP 1.5. Tato sada byla nasazena na serveru ve výše zmíněném virtuálním prostředí.

Sectoolmarket.com uvádí[4], že nástroj Burp Suite Professional 1.5.20 odhalil 136 ze 136ti zranitelností SQL Injection a 64 z 66 zranitelností Cross-Site Scripting. Vyvinutý nástroj PeTT odhalil celkem 91 ze 136 zranitelností SQL Injection a devět zranitelností Cross-Site Scripting z 66. Výsledky jsou shrnuty v tabulkách 5.1 až 5.4.

Do srovnání byly zařazeny i nástroje uvedené v sekci 2.3, jakožto i nástroje *Paros Proxy* a *Gamja*, které dle [4] dosáhly podobných výsledků jako vyvinutý nástroj PeTT. Zařazen byl i nástroj *SecuBat*, který dosáhl horších výsledků. Nástroj *Nessus* testován nebyl, proto není zařazen. Srovnání bere v úvahu úspěšnost detekce zranitelností a míru *false positive*[3] v sadě testů WAVSEP.

Nástroj	**Nalezeno SQLI**	**Testů**	**Úspěšnost**
Acunetix WVS	136	136	100.00 %
HP WebInspect	136		100.00 %
Burp Suite	136		100.00 %
OWASP ZAP	136		100.00 %
Paros Proxy	105		77.21 %
• PeTT	87		63.97 %
Gamja	68		50.00 %
w3af	48		35.29 %
SecuBat	25		18.38 %

Tabulka 5.1: Srovnání vyvinutého nástroje PeTT (označen tečkou) s vybranými nástroji v úspěšnosti detekce zranitelnosti SQL Injection.

Nástroj	**Falešné detekce SQLI**	**Testů**	**Míra falešných detekcí**
Acunetix WVS	0	10	0.00 %
Burp Suite	0		0.00 %
HP WebInspect	0		0.00 %
OWASP ZAP	3		30.00 %
w3af	3		30.00 %
Paros Proxy	4		40.00 %
• PeTT	5		50.00 %
SecuBat	7		70.00 %
Gamja	8		80.00 %

Tabulka 5.2: Srovnání vyvinutého nástroje PeTT (označen tečkou) s vybranými nástroji v množství falešných detekcí (false positive[3]) při detekci zranitelnosti SQL Injection.

[3]Termín false positive je vysvětlen na straně 5.

Nástroj	Nalezeno RXSS	Testů	Úspěšnost
Acunetix WVS	66	66	100.00 %
HP WebInspect	66		100.00 %
OWASP ZAP	66		100.00 %
Burp Suite	64		96.97 %
w3af	25		37.88 %
Paros Proxy	16		24.24 %
Gamja	12		18.18 %
• PeTT	8		12.12 %
SecuBat	5		7.58 %

Tabulka 5.3: Srovnání vyvinutého nástroje PeTT (označen tečkou) s vybranými nástroji v úspěšnosti detekce zranitelnosti Reflected Cross-Site Scripting.

Nástroj	Falešné detekce RXSS	Testů	Míra falešných detekcí
Acunetix WVS	0	7	0.00 %
Burp Suite	0		0.00 %
HP WebInspect	0		0.00 %
OWASP ZAP	0		0.00 %
• PeTT	0		0.00 %
SecuBat	0		0.00 %
w3af	0		0.00 %
Gamja	1		14.29 %
Paros Proxy	3		42.86 %

Tabulka 5.4: Srovnání vyvinutého nástroje PeTT (označen tečkou) s vybranými nástroji v množství falešných detekcí (false positive[3]) při detekci zranitelnosti Reflected Cross-Site Scripting.

Výše uvedené srovnání může pro vyvinutý nástroj (PeTT) vyznít poměrně negativně. Je však třeba si uvědomit, že Burp Suite patří k nejlepším dostupným penetračním nástrojům a za jeho mnohaletým vývojem stojí komerční společnost. Dosažené výsledky jsou srovnatelné s nekomerčními nástroji *Paros Proxy* a *Gamja*, které v seznamu Sectoolmarket.com[4] figurují také. PeTT dosáhl lepších výsledků než například nástroj *SecuBat*.

Poměrně nízké výsledky v detekci XSS lze z velké míry zdůvodnit zaměřením na jazyk Java Script. Detekční modul XSS se totiž zaměřuje pouze na skriptování v jazyce Java Script, zatímco sada testů WAVSEP obsahuje také nezanedbatelné množství testů v jazyce VBScript.

Detekční moduly slouží spíše pro demonstraci funkcionality frameworku a implementované detekční algoritmy nejsou nikterak komplikované. Z tohoto hlediska představuje téměř 64 % detekovaných zranitelností SQLI velmi dobrý výsledek. Tento výsledek rovněž ukazuje, že díky vyvinutému frameworku lze jednoduše vytvořit poměrně schopný detekční modul.

Kapitola 6

Závěr

V této práci byla diskutována problematika zranitelností webových aplikací. Byla objasněna motivace k vyhledávání zranitelností metodou penetračního testování i princip, kterým tyto zranitelnosti vznikají. Práce také popsala existující nástroje pro penetrační testování a jejich funkcionalitu. Rovněž přiblížila 10 nejběžnějších zranitelností podle žebříčku OWASP Top 10[25]. Aby byl vytvořen dobrý základ k pochopení zranitelností, byly diskutovány technologie, na kterých webové aplikace stojí.

Jelikož je součástí práce implementace detekce zranitelností SQL Injection a Cross-Site Scripting, byly tyto zranitelnosti popsány detailněji, včetně jejich dopadů. Práce uvedla krátké ukázky zranitelných aplikací i příklady útoků z tisku. Uvedeny byly také možnosti obrany před těmito zranitelnostmi.

Implementovaný nástroj

V rámci práce byl navržen a implementován framework pro podporu penetračního testování. Jedná se o nástroj v jazyce Java, který poskytuje prostředky pro zachytávání HTTP komunikace, jednotný systém pro hlášení zranitelností a nezbytné abstrakce pro jednoduchou práci se zachycenou komunikací. Jeho součástí je rovněž uživatelské rozhraní. Pro odchyt komunikace byla implementována komponenta proxy, která umožňuje zpracování velkého množství spojení zároveň a podporuje také záchyt komunikace HTTPS. Aby bylo zabráněno nechtěným útokům na cizí infrastrukturu, bylo pro testování nástroje vytvořeno virtuální prostředí oddělené od fyzické sítě. V rámci tohoto prostředí byly nasazeny a vytvořeny webové aplikace pro ladění a testování nástroje.

Framework sám o sobě neprovádí detekci zranitelností. K tomuto účelu slouží detekční moduly, kterými jej lze rozšířit. V rámci této práce byly vytvořeny moduly pro detekci zranitelností SQL Injection a Cross-Site Scripting. Tyto moduly úspěšně demonstrují funkcionalitu frameworku a použitelnost jeho API. Výsledný nástroj ulehčuje práci nejen penetračním testerům, ale také vývojářům nových penetračních nástrojů. Ti mohou využít služeb, které framework poskytuje, a vyvinout tak nástroj rychleji ve formě detekčního modulu pro framework. K API byla vytvořena dokumentace ve formátu javadoc.

Porovnání s existujícími nástroji

Vytvořený nástroj PeTT (Penetration Testing Tool) byl pro porovnání s existujícími penetračními nástroji podroben sadě testů WAVSEP[5]. Výsledky poté byly srovnány s výsledky jiných nástrojů uvedenými v [4]. Vytvořený nástroj PeTT dosáhl úspěšnosti téměř 64 % při

detekci zranitelnosti SQL Injection a 13 % při detekci zranitelnosti Reflected Cross-Site Scripting. Výsledky vytvořeného nástroje jsou tak dle Sectoolmarket.com[4] srovnatelné s nekomerčními nástroji Paros Proxy či Gamja a převyšují například nástroj SecuBat. Díky rozšiřitelnosti o detekční moduly má navíc vyvinutý nástroj dobrou šanci na zvýšení úspěšnosti, pokud dojde k vylepšení detekčních modulů.

Vytvořený nástroj splňuje požadavky definované v sekci 3.1, rovněž byly vytvořeny požadované detekční moduly. Zmíněné detekční moduly byly vytvořeny pro účely demonstrace funkcionality řešení a nabízejí prostor pro vylepšení. V současném stavu je vyvinutý nástroj využitelný v praxi a je schopen ulehčit práci penetračnímu testerovi. Detekce zranitelností však zatím nedosahuje kvalit komerčních nástrojů.

Další vývoj

V rámci dalšího vývoje aplikace by mohl být zdokonalen modul pro detekci Cross-Site Scripting (XSS), například rozšířením o detekci DOM-based XSS. Modul pro detekci SQL Injection (SQLI) by mohl být rozšířen o detekci Blind SQLI. Rovněž by mohly být vytvořeny moduly pro detekci dalších zranitelností, například zranitelností Cross-Site Request Forgery či Path Traversal[22, str. 333].

Potenciál vlastního frameworku by mohl být navýšen implementací pokročilejších algoritmů pro rekurzivní průchod stránkami testované aplikace. Vhodným rozšířením by rovněž byl detekční modul pro statistické vyhledávání anomálií v zaznamenané komunikaci.

Literatura

[1] Andreu, A.: *Professional pen testing for Web applications*. Indianapolis: John Wiley & Sons, 2006, ISBN 978-0-471-78966-6.

[2] Antal, L.; Barabas, M.; Hanáček, P.: Kompromitace dat pomocí SQL Injection, část III. *DSM Data Security Management*, ročník 18, č. 3, 2014: s. 25–29, ISSN 1211-8737.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=10732

[3] CERT Carnegie Mellon University: *Malicious HTML Tags Embedded in Client Web Requests*. [online], 2000 [cit. 2015-04-10].
URL http://www.cert.org/historical/advisories/CA-2000-02.cfm

[4] Chen, S.: *Price and Feature Comparison of Web Application Scanners*. [online], 2015-01-07 [cit. 2015-04-26].
URL http://www.sectoolmarket.com/price-and-feature-comparison-of-web-application-scanners-unified-list.html

[5] Chen, S.: *The Web Application Vulnerability Scanner Evaluation Project*. [online], cit. 2015-04-26.
URL https://code.google.com/p/wavsep/

[6] Chris Shiflett: *addslashes() Versus mysql_real_escape_string()*. [online], 2006 [cit. 2015-04-10].
URL http://shiflett.org/blog/2006/jan/addslashes-versus-mysql-real-escape-string

[7] Fielding, R.; Gettys, J.; aj.: *Hypertext Transfer Protocol – HTTP/1.1*. [online], 1999 [cit. 2015-04-10].
URL http://tools.ietf.org/html/rfc2616

[8] Fielding, R.; Reschke, J.; aj.: *RFC7230: Hypertext Transfer Protocol (HTTP/1.1)*. [online], 2014 [cit. 2015-04-10].
URL http://tools.ietf.org/html/rfc7230

[9] Fielding, R.; Reschke, J.; aj.: *RFC7231: Hypertext Transfer Protocol (HTTP/1.1)*. [online], 2014 [cit. 2015-04-10].
URL http://tools.ietf.org/html/rfc7231

[10] Gallagher, T.: *Automated detection of cross site scripting vulnerabilities*. 2004-05-19, EP Patent App. EP20,030,023,125.
URL http://www.google.com/patents/EP1420562A2?cl=en

[11] Grossman, J.: *XSS Attacks: Cross-site scripting exploits and defense*. Burlington: Syngress, 2007, ISBN 978-1-59749-154-9.

[12] Kutypa, M.: *Nástroj pro detekci zranitelnosti SQL Injection*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, 2013.

[13] Leyden, J.: *Watchdog bites hotel booking site: Over 3k card details slurped*. [online], 2014-11-05 [cit. 2015-05-01].
URL http://www.theregister.co.uk/2014/11/05/hotel_booking_website_fined_over_breach_that_exposed_credit_card_details/

[14] OWASP; Mayhew, B.: *OWASP WebGoat Project*. [online], 2015-03-08 [cit. 2015-04-27].
URL https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project

[15] OWASP Foundation: *XSS (Cross Site Scripting) Prevention Cheat Sheet*. [online], 2015-03-02 [cit. 2015-03-26].
URL https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet#A_Positive_XSS_Prevention_Model

[16] Palmer, C. C.: *Ethical hacking*. *IBM Systems Journal*, ročník 40, č. 3, 2001: s. 769–780, ISSN 0018-8670.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5386933;http://www.elifree.com/downloads/p_work/chapter_one/books/palmer.pdf

[17] Pauli, D.: *Drupal-opcalypse! Devs say best assume your CMS is owned*. [online], 2014-10-30 [cit. 2015-05-01].
URL http://www.theregister.co.uk/2014/10/30/drupal_sites_considered_hosed_if_sqli_hole_unclosed/

[18] Pauli, D.: *Death by comments: WordPress XSS vuln is biggest for years*. [online], 2014-11-24 [cit. 2015-03-25].
URL http://www.theregister.co.uk/2014/11/24/worst_wordpress_hole_for_five_years_affects_86_of_sites/

[19] Pauli, D.: *Weather Channel forecast: Bleak, with prolonged XSS*. [online], 2014-12-01 [cit. 2015-03-25].
URL http://www.theregister.co.uk/2014/12/01/weather_channel_forecast_bleak_with_a_chance_of_xss/

[20] Pauli, D.: *I helped Amazon.com find an XSS hole and all I got was this lousy t-shirt*. [online], 2015-03-26 [cit. 2015-03-26].
URL http://www.theregister.co.uk/2015/03/26/amazon_shutters_xss_hijack_hole/

[21] Rescorla, E.: *RFC2818: HTTP Over TLS*. [online], 2000 [cit. 2015-04-27].
URL https://tools.ietf.org/html/rfc2818

[22] Stuttard, D.; Pinto, M.: *The web application hacker's handbook: discovering and exploiting security flaws*. Indianapolis: John Wiley & Sons, 2007, ISBN 978-0-470-17077-9.

[23] Symantec Corporation: *Internet Security Threat Report 2014*. [online], 2014 [cit. 2015-03-21].
URL
http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf

[24] The MITRE Corporation: *CVE – Common Vulnerabilities and Exposures*. [online], 2015-04-23 [cit. 2015-04-25].
URL http://cve.mitre.org/index.html

[25] Wichers, D.: *OWASP Top-10 2013*. [online], 2013 [cit. 2015-03-25].
URL
http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202013.pdf

Příloha A

Slovník

API Application Programming Interface

DOM Document Object Model

GUI Graphical User Interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS HTTP over TLS

OWASP Open Web Application Security Project

SQL Structured Query Language

SQLI SQL Injection

SSL Secure Socket Layer

TCP Transmission Control Protocol

XSS Cross-Site Scripting

Příloha B

Obsah CD

Disk, který je k této práci přiložen, obsahuje následující adresáře a soubory:

- `dist/` obsahuje přeloženou aplikaci v archivu `.jar`. Tento archiv obsahuje i potřebné knihovny. V adresáři `dist/modules/` se nacházejí přeložené detekční moduly. Ke spuštění aplikace je zapotřebí Java Runtime Environment alespoň ve verzi 8.
- `licenses/` obsahuje licence použitých knihoven.
- `src/` obsahuje zdrojové kódy vytvořeného nástroje, včetně API a detekčních modulů.
- `thesis/` obsahuje elektronickou verzi tohoto textu ve formátu `.pdf`.
- `thesis-latex/` obsahuje zdrojové soubory textu této práce.
- `vulnweb/` obsahuje vytvořenou napadnutelnou webovou aplikaci se slabinami Stored a Reflected XSS.
- `readme.txt` obsahuje stručný popis obsahu CD a manuál k aplikaci.

Příloha C

Manuál

Nástroj pro penetrační testování webových aplikací

Vytvořil Michal Dobeš (xdobes13) v rámci bakalářské práce na FIT VUT.
ak. rok 2014/2015

Aplikace

Aplikace je nástroj pro podporu penetračního testování webových aplikací. Je rozšiřitelná prostřednictvím modulů pro detekci jednotlivých zranitelností. Přiloženy jsou moduly pro detekci zranitelností Cross-Site Scripting (XSS) a SQL Injection.

Překlad a spuštění

Aplikace byla vytvořena vývojovým prostředím NetBeans, nejjednodušší cestou je tedy otevřít projektové soubory v tomto prostředí a zvolit překlad a spuštění.

Aplikaci je také možno přeložit z terminálu. Stačí přejít do adresáře `src/PeTT` a spustit příkaz `ant` bez argumentů. Po překladu lze aplikaci spustit. Přeložená aplikace se nachází v adresáři `src/PeTT/dist`. Jedná se o archivy `PeTT.jar` (vyžaduje pro spuštění adresář lib se závislostmi) a `PeTT-standalone.jar`, který závislosti obsahuje uvnitř archivu.

Obdobným způsobem lze přeložit i detekční moduly. Pro použití musí být přeložené moduly umístěny v adresáři `modules/`, který se musí nacházet ve stejném adresáři jako spustitelný archiv `PeTT.jar` (či `PeTT-standalone.jar`).

Použití aplikace

Po startu aplikace je zobrazeno hlavní okno s průvodcem. Ten umožňuje vytvořit novou skenovací úlohu. V rámci průvodce novou skenovací úlohou je třeba definovat cíl penetračního testu. Tato definice se skládá z pravidel Include (zahrň do cíle) a Exclude (vyjmi z cíle). Lze například zadat:

```
Include xss.com
Exclude xss.com/logout.php
```

V takovém případě bude cíl testu představovat celá aplikace na xss.com, vyjma stránky logout.php. Tato definice zabrání nechtěnému odhlášení uživatele během penetračního testu.

Po vytvoření skenovací úlohy začne být definovaná aplikace testována. Je silně doporučeno, aby se tester zároveň k testované aplikaci připojil skrz proxy, kterou tento nástroj

poskytuje. Ve výchozím nastavení běží tato proxy na portu 8000 lokální adresy (127.0.0.1 a ::1).

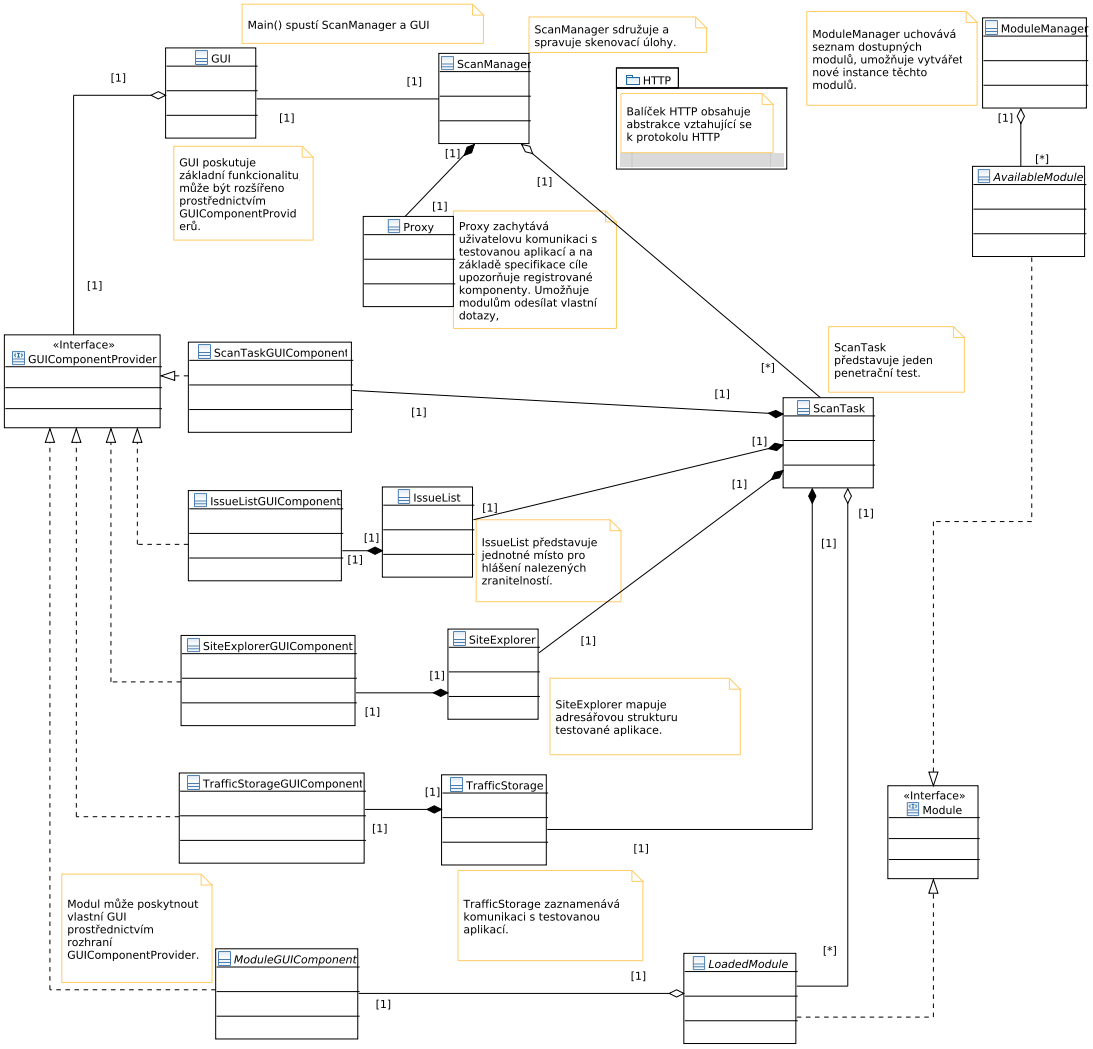
Nástroj zachytává komunikaci, která prochází přes proxy, což mu pomáhá objevovat stránky, které vestavěný algoritmus nebyl schopen objevit, například odkazy ze skriptů, atd.

O nalezených zranitelnostech testera informuje uživatelské rozhraní. Po dokončení testu lze prostřednictvím komponenty IssueList vytvořit zprávu o penetračním testu.

Více informací o další funkcionalitě nástroje je uvedeno v textu práce.

Příloha D

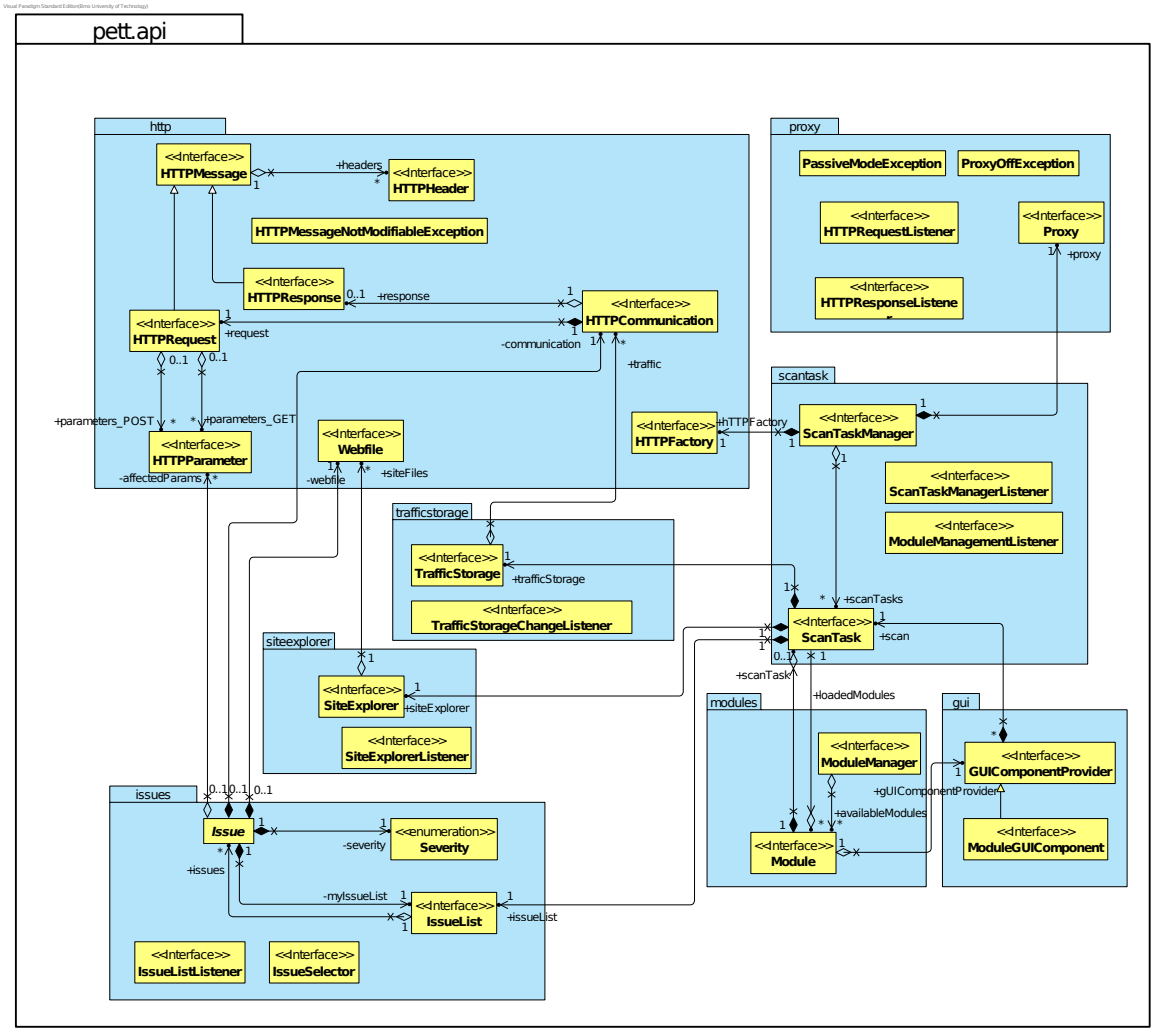
Návrh struktury frameworku: diagram tříd



Obrázek D.1: UML diagram tříd – návrh rozdělení frameworku do tříd

Příloha E

API: diagram tříd



Obrázek E.1: UML diagram tříd API.