VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

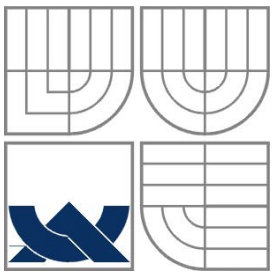
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

APLIKACE PRO PENETRAČNÍ TESTOVÁNÍ WEBOVÝCH ZRANITELNOSTÍ TYPU DENIAL OF SERVICE

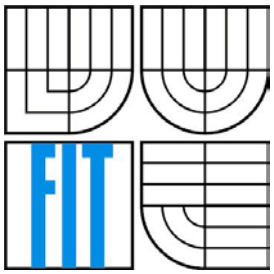
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Jaroslav Vrána
AUTHOR

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 DENIAL OF SERVICE

Penetration testing application for DoS based web vulnerabilities

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Jaroslav Vrána

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Michal Drozd

BRNO 2011

Abstrakt

Práce se zabývá problematikou zranitelností DoS v oblasti webových aplikací. Nejdříve jsou popsány principy počítačové bezpečnosti, obecné principy DoS a penetračního testování. Dále text popisuje části OWASP Testing Guide v3, které se zabývají DoS testováním webových aplikací. Na základě dosažených zkušeností je navržena vlastní aplikace na automatizovaný audit. Tato aplikace je následovně implementována a testována na webových aplikacích.

Abstract

This work deals with a issue of a DoS vulnerability in web applications. At first, there are described principles of a computer security, general principles of the DoS and a penetration testing. Further text describes a OWASP Testing Guide v3 for the DoS in web applications. There is a design of own application on basis own experiences. This application is implemented and tested by the web applications.

Klíčová slova

HTTP generátor požadavků, fuzzing, Denial of Service - DoS, OWASP Testing Guide v3, Microsoft .NET, penetrační testování

Keywords

HTTP request generator, fuzzing, Denial of Service - DoS, OWASP Testing Guide v3, Microsoft .NET, penetration testing

Citace

Vrána Jaroslav: Aplikace pro penetrační testování webových zranitelností typu Denial of Service, diplomová práce, Brno, FIT VUT v Brně, 2011

APLIKACE PRO PENETRAČNÍ TESTOVÁNÍ WEBOVÝCH ZRANITELNOSTÍ TYPU DENIAL OF SERVICE

Prohlášení

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

……
Jaroslav Vrána
18. 5. 2011

Poděkování

Rád bych touto cestou poděkoval svému vedoucímu diplomové práce panu Ing. Michalu Drozdovi za cenné připomínky a trpělivost při odborných konzultacích.

Obsah

1 Úvod

Webový prohlížeč je základní součástí softwarové výbavy každého osobního počítače a je pravděpodobně nejpoužívanějším typem software vůbec. Webové aplikace se stávají snad ještě běžnější součástí našeho života než klasické desktopové. Dlouhou dobu byla, a ne zřídka ještě je, zanedbávána bezpečnost webových aplikací, které jsou využívány jako zdroj informací, pomocníci při práci či správě našeho majetku. Zanedbávat bezpečnost je v dnešní době velké riziko. Existují profesionální skupiny, které se živí kybernetickým zločinem, např. z ruského území operující zločinecká organizace Russian Bussines Network. Tato organizace se původně zaměřovala na legální obchod, ovšem časem se přeorientovala na pole, kde jsou výdělky řádově vyšší, na kybernetický zločin. Tato organizace vydělává sto padesát milionů dolarů za rok, viz [34]. Počítačová bezpečnost není jenom byznys, ale je i národním zájmem. Důkazem mohou být útoky na vládní servery Jižní Koreje a USA. Tyto útoky má na svědomí podle neoficiálních vyjádření „armáda“ profesionálních vojáků KLDR vycvičených pro kybernetický boj, viz [35]. Existují další podobné útoky, kdy např. ruští hackeři v průběhu konfliktu v Gruzii zcela vyřadili gruzijský parlamentní web a země vydávala oficiální prohlášení prostřednictvím blogovacího systému Blogger.

Tato práce se zabývá DoS (Denial of Service) útoky na webové aplikace a okrajově také DoS útoky na systém. DoS je útok, který způsobí odepření služby legitimním uživatelům. Odepření služby může nastat např. pádem webové aplikace, které vložíme na vstup velké množství dat, které neočekává (buffer overflow). Další možností je např. požadavek na operaci, která spotřebuje velké množství prostředků serveru a již nejsou prostředky pro obsloužení dalších účastníků (SQL Wildcards). Zanedbávání zranitelností způsobujících DoS je velice nebezpečné, jak ukazuje např. kybernetický útok na Gruzii.

Cílem této práce je seznámit se s testováním webových zranitelností typu DoS – Denial of Service pomocí metodologie OWASP Testing Guide v3 a následně navrhnout a implementovat aplikaci pro testování webových aplikací na zranitelnosti typu DoS. Dále bude aplikace otestována na příkladech existujících i uměle vytvořených webových aplikacích a porovnat s již existujícími aplikacemi podobného zaměření.

1.1 Struktura práce

Druhá kapitola se zabývá základními pojmy počítačové bezpečnosti, jako je bezpečnost a použitelnost, základními cíli počítačové bezpečnosti, hrozbami, aktivy atd.

Třetí kapitola popisuje pojmy DoS a DDoS. Popisuje, jakým způsobem jsou vedeny útoky v prostředí webových aplikací. Dále je uveden popis některých útoků (SYN Flood, Ping of death a

teardrop) na systém z dob Windows NT a jeden útok aktuální (slowloris), který může způsobit DoS na webový server Apache a podobné.

Čtvrtá kapitola popisuje penetrační testování. Nejdříve představuje postup vedení skutečného útoku a následuje popis penetračního testu a jejich srovnání.

Pátá kapitola popisuje aplikační aspekty penetračního testování v oblasti webových aplikací. Popisuje HTTP protokol, některé existující aplikace a především konfiguraci systému, na kterém probíhali testy v rámci této práce.

Šestá kapitola se zaměřuje na popis organizace OWASP a jejího projektu OWASP Testing Guide v3 a především části věnované problematice DoS. Zabývá se i způsobem ohodnocení rizika a tvorbou dokumentace, které jsou velmi důležité při penetračním testování.

Sedmá kapitola se věnuje návrhu vlastní aplikace na základě zkušeností získaných při studiu této problematiky.

Osmá kapitola se zabývá implementací vlastní aplikace. Nejdříve je popsán Microsoft .NET Framework, na kterém je implementována celá aplikace. Následuje představení aplikace z implementačního hlediska. Nakonec je aplikace představena z hlediska uživatelského rozhraní.

Kapitola devátá popisuje testy, které ověřuji funkčnost implementované aplikace prostřednictvím detekce některých (Zamykání účtů, Buffer overflow) zranitelností představených v kapitole šest. Nakonec je implementovaná aplikace srovnána s aplikacemi URLStress a JBroFuzz.

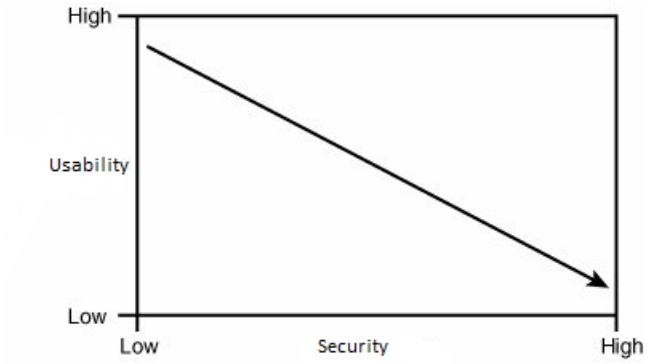
Na konci práce, v desáté kapitole, jsou uvedeny další cíle vývoje a diskuze dosažených výsledků.

2 Základní pojmy počítačové bezpečnosti

Tato kapitola vysvětluje základní pojmy počítačové bezpečnosti (dále jen bezpečnosti) důležité pro pochopení jejího významu. Je zde popsáno, jakou roli hraje rovnováha bezpečnosti a použitelnosti při návrhu zabezpečení systému. Dále jsou vysvětleny tři hlavní cíle bezpečnosti důvěrnost, integrita a dostupnost. Následuje popis vzniku bezpečnostního rizika za účasti vzniklé hrozby, zranitelného místa a aktiv.

2.1 Bezpečnost vs. použitelnost

Bezpečnost je o hledání rovnováhy. Systém můžeme před síťovým útokem chránit tak, že ho zcela odpojíme ze sítě. Tím je systém v bezpečí, ale není schopen poskytovat požadované služby. Dalším extrémem je připojit systém přímo do internetu. Nyní nejsou uživatelé nijak omezováni, ale systém je velice náchylný na hrozby. Situaci zobrazuje obrázek 2-1. Úkolem bezpečnostních expertů je, v závislosti na typu organizace, její politice a požadavkům, najít rovnováhu mezi použitelností a zabezpečením systému.



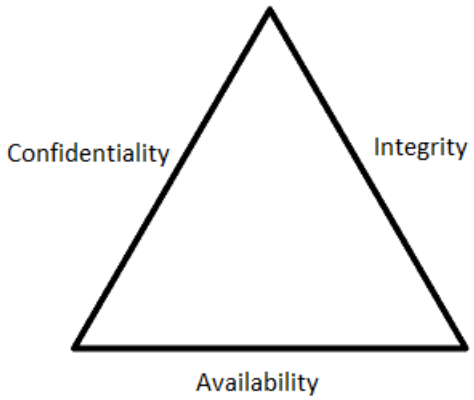
Obrázek 2-1: Použitelnost vs. Bezpečnost [Zdroj: [2]]

2.2 Hlavní cíle bezpečnosti

Hlavní cíle bezpečnosti jsou tři. **Důvěrnost** (angl. confidentiality), **integrita** (integrity) a **dostupnost** (availability). Důvěrnost je ochrana proti neoprávněnému prozrazení informace. Docílit ji můžeme například použitím šifrování (symetrické, asymetrické). Integrita je ochrana proti neoprávněné modifikaci informace. Pro kontrolu integrity se používá hash, jako reálně používanou metodu lze

zmínit např. MD5 hash. Dostupnost je ochrana proti odepření dat nebo služby. Právě posledním cílem bezpečnosti, dostupností, se bude tato práce v dalších kapitolách zabývat.

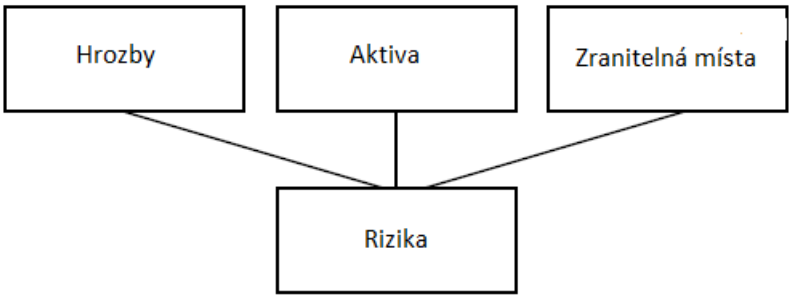
Klasickým zobrazením tří hlavních cílů bezpečnosti je bezpečnostní trojúhelník (CIA triangle). Na obrázku 2-2 je bezpečnostní trojúhelník znázorněn.



Obrázek 2-2: CIA trojúhelník [Zdroj: HTTP://www.sqlservercentral.com/]

2.3 Hrozby, aktiva a zranitelná místa

Dalšími důležitými pojmy jsou **zranitelná místa** (vulnerabilities), **hrozby** (threats), **aktiva** (assets) a **rizika** (risks). Vztah těchto tří entit znázorňuje obrázek 2-3.



Obrázek 2-3: Vznik rizika

Aktiva jsou všechny cennosti dané organizace. Může se jednat o software, hardware, data, postupy, ale dokonce i o lidi, kteří mají např. určitou znalost.

Hrozba je taková vlastnost prostředí, která může způsobit narušení bezpečnosti, pokud dostane příležitost. Hrozby lze rozdělit na neúmyslné a úmyslné. Mezi neúmyslné hrozby patří živelné události (požár, zemětřesení, …), poruchy zařízení, chyby v software a selhání osob. Cílem úmyslných hrozeb nemusí být jenom data, ale např. i neoprávněné využívání hardware.

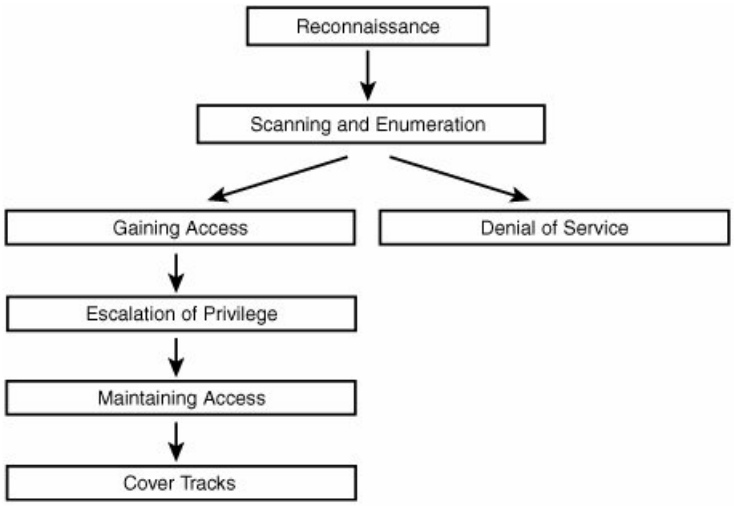
Zranitelné místo je chyba nebo slabina v návrhu, implementaci nebo provozu systému, která může být využita pro narušení bezpečnosti systému. Taková chyba může vzniknout při návrhu, implementaci či při provozu (viz [3]).

3 Denial of Services (DoS)

Tato kapitola popisuje pojem DoS, znázorňuje jeho umístění v rámci útoku a popisuje entity systému náchylné na DoS útok v prostředí webových aplikací. Dále je v kapitole podán základní popis DDoS. Dále jsou popsány nejzmámenější útoky na systém z dob Windows 95/NT, jedná se o Ping of death, SYN flood a Teardrop. Nakonec je popsán aktuální možný útok na webové servery Apache, Slowloris.

3.1 Základní popis DoS

Jak již bylo napsáno výše, dostupnost je jedním ze tří cílů bezpečnosti a právě DoS útok je zaměřen na dostupnost. Postup útoku podle kapitoly 4 je znázorněn na obrázku 3-1.



Obrázek 3-1: Strom vedení útoku (Zdroj: [4])

Cílem DoS útoku v prostředí webových aplikací, mohou být:

- **systémové zdroje** – zneužitím zranitelnosti ve webové aplikaci může dojít k vyčerpání systémových zdrojů (diskové místo, RAM, strojový čas, …), viz kapitola 5.3
- **samotná aplikace** – chyby v implementaci nebo návrhu mohou zanést chyby, které mohou způsobit např. buffer overflow (jazyky, kde má sám programátor na starosti správu paměti)
- **uživatel** – útok na uživatele zapříčiní, že oprávněný uživatel se nemůže přihlásit do systému
- **databáze** – mnoho aplikací využívajících databázi je citlivých např. na SQL Injection, což umožní např. modifikovat strukturu databáze a služba na ní postavena je vyřazena z provozu.

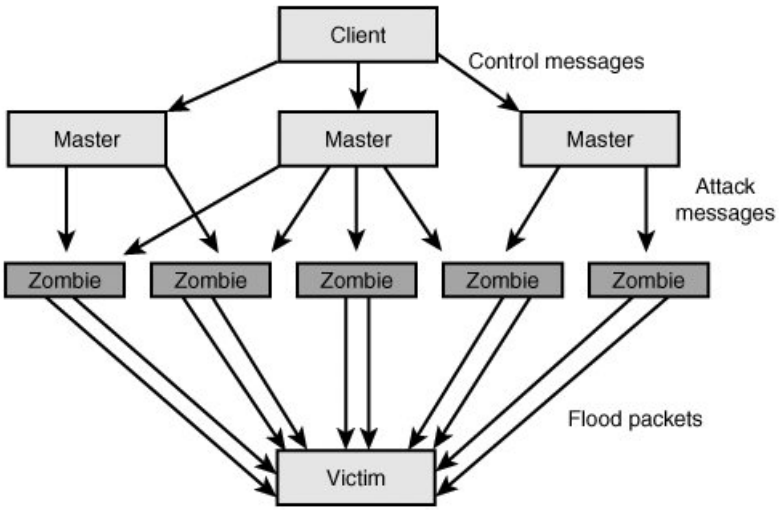
Konkrétní typy zranitelností webových aplikací budou popsány podle metodologie OWASP Testing Guide v3 v kapitole 6.

3.2 Základní popis DDoS

Speciálním případem DoS je DDoS. DDoS (Distributed DoS) má stejný cíl jako obyčejný DoS, ale je veden prostřednictvím velkého počtu útočících stanic, které zpravidla bývají součástí botnetu[1] a jsou označovány jako zombie[2]. Situaci zobrazuje obrázek 3-2. Nejníže na obrázku je oběť (victim) útoku, která je „zaplavována“ požadavky. Tyto požadavky jsou posílány infikovanými počítači (zombies). Podle typu malwaru, kterým jsou počítače infikovány, tak vykonávají určitý typ úloh. Parametry pro vykonání úloh jsou posílány komunikačním kanálem přes internet od nadřazených entit (Masters). Komunikační kanál často bývá realizován pomocí IRC[3] (Internet Relay Chat). Řízení je většinou decentralizované, proto vyřazení jednoho Master neznamená zhroucení botnetu. Na vrcholu hierarchie je klient, který platí za využívání těchto služeb a zadává požadavky, které jsou dále předávány pomocí Masters. Tento typ útoku se díky stále rozrůstajícím botnetům stává velkou hrozbou.

Obranou (viz [5]) může být integrovaný systém geolokace (většina útoků je provozována ze zahraničí), který umožní přístup na webový server pouze uživatelům vyskytujícím se v definované lokaci, např. v ČR. Tento způsob nemusí ovšem vyhovovat všem. Další možností je využít IPS (Intrusion Prevention System). IPS (viz [6]) je systém, který monitoruje síťové aktivity a při detekci určitého vzorku chování dokáže cíleně reagovat. Výhodou je, že administrátor může definovat vlastní vzorky škodlivého chování, tzv. IPS signatury. Tyto IPS signatury lze získat např. pomocí tzv. Honeypotů[4] (viz [7]).

[1] Botnet označuje síť počítačů, které jsou infikovány malwarem a mohou být centrálně nebo decentralizovaně ovládány.
[2] Zombie se nazývá infikovaná stanice, která je součástí botnetu.
[3] IRC byl jedním z prvních protokolů pro komunikaci v reálném čase na internetu a používá se doposud.
[4] Honeynet je sw systém emulující reálný systém v síťové infastruktůře určený pro analýzu útoků.



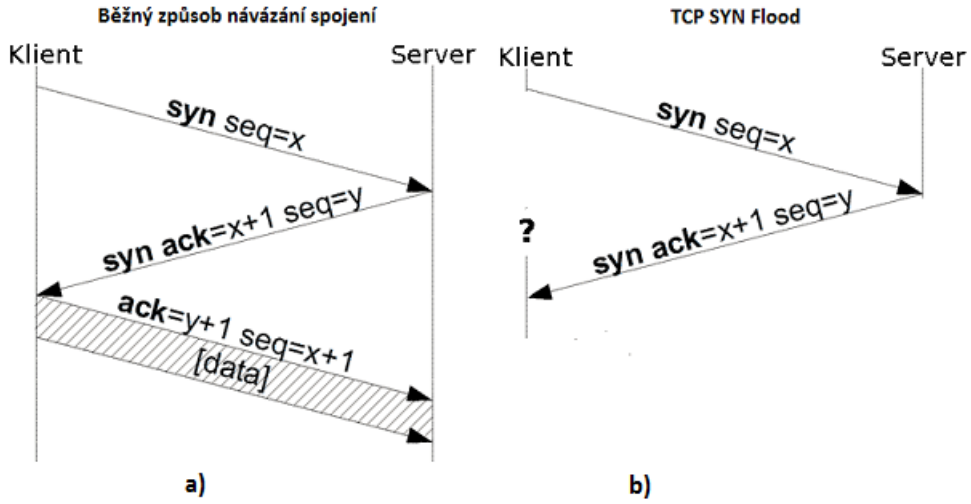
Obrázek 3-2: Struktura DoS útoku pomocí botnetu (Zdroj: HTTP://www.networkworld.com)

3.3 Příklady DoS útoků na systém

Zde jsou popsány nejznámější DoS útoky na systém. Jedná se především o útoky z dob Windows 95 a Windows NT. Konkrétně jsou popsány útoky SYN flood, Ping of death a Teardrop. Aktuálním útokem je Slowloris, který je popsán na konci kapitoly. Na tento útok je náchylná i aktuální verze webového serveru Apache2.

3.3.1 SYN Flood

SYN Flood je příkladem DoS útoku na zdroje serveru. Při tomto útoku je odesláno velké množství packetů s příznakem SYN a podvrženou zdrojovou IP adresou. To vede k tomu, že na bufferu oběti je alokováno velké množství částečně otevřených spojení, které čekají na packety s příznakem ACK, které by TCP spojení ustanovily. K tomu ovšem z důvodu podvržené zdrojové IP adresy nedojde. Cílem je vyčerpat celý buffer, aby nebylo možné přijmout legitimní spojení.



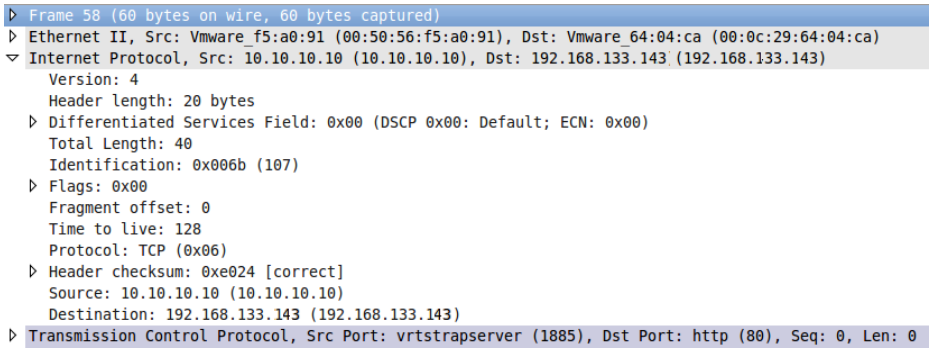
Obrázek 3-3: a) TCP handshake, b) SYN Flood

Útok lze provést např. pomocí programu hping:

```
sudo hping -a 10.10.10.10 -p 80 -S 192.168.133.143 -i u1
```

Text 3-1: Podvrhnutí zdrojové IP adresy

Tento příkaz pošle packet s příznakem SYN na server s IP 192.168.133.143 na port číslo 80 s podvrženou zdrojovou adresou. Obrázek 3-4 zobrazuje podvrhnutou IP adresu zobrazenou pomocí aplikace Wireshark. Na cílovém systému lze otevřená neustavená spojení sledovat pomocí aplikace netstat. Detailně o SYN Flood pojednává [8]. Studium i již neaktuálních zranitelností je důležité pro pochopení nových. Např. pro současné webové servery (Apache2, atd) je aktuální útok Slowloris popsaný v kapitole 3.3.3 je analogií SYN Flood na aplikační vrstvě.



Obrázek 3-4: Wireshark: Podvržená zdrojová IP adresa

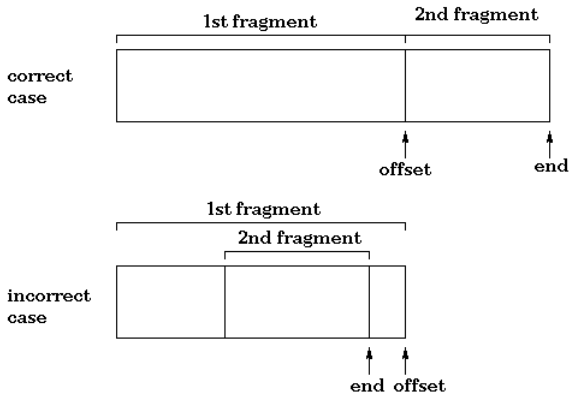
3.3.2 Ping of death a Teardrop

Teardrop a ping of death jsou cíleny na programovou chybu v operačních systémech. Ping of death využíval chyby, kdy systémy neuměly zpracovat packet větší než 65 536 bajtů. V případě, že systém přijal packet o větší velikosti, došlo ke zhroucení systému.

V případě teardrop je poupraven offset v hlavičce IP socketu, tím je způsobeno překrytí částí fragmentovaných packetů (viz obrázek 3-5). Cílový systém neví, jak zpracovat překrývající se fragmenty a nastává jeho pád. Webový zdroj [9] uvádí detailní popis teardrop včetně zdrojového kódu. Po přeložení se program spustí následovně:

```
./teardrop zdrojová_IP cílová_IP -p port -n počet_opakování
```

Text 3-2: Použití aplikace teardrop



Obrázek 3-5: Fragmentace packetu

3.3.3 Slowloris HTTP DoS

Tento útok nevyužívá chyby v operačním systému jako předchozí, ale je cílen na aplikační vrstvu. Na rozdíl od SYN Flood je ustanoveno korektní TCP spojení. Slowloris je perlový skript ([10]), který produkuje HTTP provoz na zvolený server. Slowloris útok se nesnaží vytvořit velký síťový provoz jako jiné DoS útoky, ale produkuje legitimní HTTP provoz. V daném časovém intervalu posílá HTTP požadavky, které neukončí HTTP spojení a před vypršením doby, kdy má být spojení ukončeno pošlou další legitimní HTTP požadavek, který otevřené spojení opět prodlouží. Tento typ útoku nevyžaduje velké vytížení přenosového pásma. Náchylné jsou servery, které používají vlákna pro příjem spojení jako např. Apache ve verzi 2 (standardně dodáván k současným linuxovým distribucím [11]), GoAhead Webserver ([12]) atd. Postupně jsou všechny tyto vlákna „uzamčena“ a čekají na další data. Klienti, kteří jsou již připojeni na webový server, mají stále přístup ke službám, proto musí skript Slowloris běžet tak dlouho dokud nebude sám držet všechna spojení.

Nejdříve je třeba zjistit časový interval, po kterém je potřeba poslat další část dat, aby nedošlo k uzavření spojení. Toto přímo umožňuje skript Slowloris, který spojení otestuje a následně sdělí vhodný časový interval, viz text 3-3.

```
jarda@ubuntu:~/Desktop$ ./slowloris.pl -dns 192.168.133.132 -port 80 -test
Welcome to Slowloris - the low bandwidth, yet greedy and poisonous HTTP client
Defaulting to a 5 second tcp connection timeout.
Multithreading enabled.
This test could take up to 14.3666666666667 minutes.
Connection successful, now comes the waiting game...
Trying a 2 second delay:
       Worked.
Trying a 30 second delay:
       Worked.
Trying a 90 second delay:
       Worked.
Trying a 240 second delay:
       Worked.
Trying a 500 second delay:
       Worked.
Remote server closed socket.
Use 500 seconds for -timeout.
```

Text 3-3: Slowloris – test na velikost timeoutu spojení

Následuje vlastní Slowloris útok. V této fázi se nastaví parametry jako v předchozím příkladu plus získaná hodnota časového intervalu a počet socketů, které se mají otevřít. Následuje výpis skriptu Slowloris, který provádí úspěšný DoS útok, viz text 3-4. Následující výpis text 3-5 zobrazuje pokus o připojení na inkriminovaný server. Připojení se nepodařilo, DoS útok proběhl úspěšně.

```
jarda@ubuntu:~/Desktop$ ./slowloris.pl -dns 192.168.133.132 -port 80 -timeout 500 -num 500 -
tcpto 5
Welcome to Slowloris - the low bandwidth, yet greedy and poisonous HTTP client
Multithreading enabled.
Connecting to 192.168.133.132:80 every 500 seconds with 500 sockets:
               Building sockets.
               Building sockets.
               Building sockets.
               Sending data.
Current stats: Slowloris has now sent 478 packets successfully.
This thread now sleeping for 500 seconds...
```

Text 3-4: Slowloris - DoS útok pomocí skriptu Slowloris

```
C:\Users\Jarda>telnet 192.168.133.132 80
Připojování k 192.168.133.132...Nelze navázat spojení s hostitelem. na portu 80:
Připojení se nezdařilo
```

Text 3-5: Telnet - pokus o připojení

Skript Slowloris je možné provozovat na unixových systémech bez omezení, ale systémy MS Windows XP a vyšší mají zabudovanou ochranu, která omezuje držení více otevřených (např. Windows 7 mají limit dvou spojení) spojení na jeden webový server.

Obrana proti Slowloris může být provedena několika způsoby. Detekce pomocí IPS systémů může být obtížná, protože na první pohled se jedná o legitimní HTTP provoz (převzato z [13]). Ochrana může být provedena pomocí hardware nebo přidanou konfigurací systému. Hardwarovým řešením může být HW Firewall, který nepošle dále HTTP požadavky, které nejsou kompletní. Po té co přijme kompletní HTTP požadavek, tak ho předá webovému serveru. Další možností je konfigurace softwarového firewallu, který umožní přijmout pouze několik HTTP požadavků z jedné příchozí adresy. Možná konfigurace firewallu iptables je na ext 3-6.

```
iptables -A INPUT -p tcp --syn --dport 80 -m connlimit --connlimit-above 100 -j DROP
```

Text 3-6: Konfigurace iptables

4 Penetrační testování

Penetrační testování je označován za tzv. etický hacking. Jedná se o simulovaný útok na objekt zájmu (TOE – Target Of Evaluation) s vědomím jeho majitele za účelem nalezení chyb v systému a jejich následným odstraněním.

K testování se může přistupovat třemi způsoby. Tester může k testování TOE přistupovat jako k černé skříňce (**Blackbox**), což znamená, že nemá žádné informace o jejím vnitřním fungování. Další přístup je zcela opačný, tester si je zcela vědom fungování TOE, tento přístup je označován jako **Whitebox** test. Třetí, poslední způsob, jde střední cestou. Tester částečně ví, jak TOE funguje. Tento způsob je označován jako **Graybox** test.

4.1 Postup vedení útoku

Skutečný útok probíhá zpravidla v šesti krocích. Podle účelu a cíle mohou být některé kroky vynechány, ale pořadí zůstává neměnné.

1) **Provádění průzkumu (Performing Reconnaissance)**
2) **Skenování a enumerace systému (Scanning and enumeration)**
3) **Získání přístupu (Gaining access)**
4) **Eskalace práv (Escalation of privilege)**
5) **Udržování přístupu (Maintaining access)**
6) **Zametení stop (Covering tracks)**

Provádění průzkumu je první krok před útokem. Jedná se o pasivní typ získávání informací. Tento krok zahrnuje hledání, shromažďování a zaznamenávání informací o cíli. Útočník se snaží najít o cíli tolik informací, kolik jen je možné. Zajímavé informace mohou být například i v odpadkách. Ovšem vstupní branou pro tuto fázi je webová stránka organizace. Další mediálně omílanou možností je sociální inženýrství [14], jejímž průkopníkem je Kevin Mitnick. Další možností může být dnes populární google hacking [15]. Infromace od registrátora (whois) a DNS záznamy mohou být také užitečným zdrojem (např. nslookup).

Skenování, tato fáze zahrnuje aktivní způsob získávání informací a jedná se o druhý krok před samotným útokem. Cílem je získat hlubší informace o TEO. Zde se získávají informace o operačním systému (OS fingerprinting), o webovém serveru (Web Server fingerprinting), o aplikacích (Application fingerprinting) a mnoho dalších v [4]. V rámci enumerace se útočník snaží získat informace o účtech, uživatelích a jejich přístupových informacích.

Skutečný útok začíná až v době, kdy se útočník pokouší získat přístup do systému. Lze se do něj dostat např. pomocí špatně zabezpečeného bezdrátového přístupového bodu nebo zranitelným

místem v některé aplikaci. Z tohoto důvodu bylo důležité provést předchozí kroky, kdy byl zmapován systém a následně nalezeno slabé místo systému. Třetí krok nemusí znamenat pouze získání přístupu, jak název napovídá, ale také znemožnění přístupu všem ostatním, viz následující kapitola 4.

Eskalace privilegií následuje po získání přístupu. Útočník má sice přístup do systému, ale pravděpodobně pomocí některého zaměstnaneckého účtu, který má omezená práva. Nyní je čas získat některou správcovskou entitou.

Udržení přístupu je prioritní. Pro tento účel jsou často použity různé druhy rootkitu. Získání dalších hesel uživatelů je další možností (/etc/passwd). Často je odstraněna původní zranitelnost přímo útočníkem, aby nebyl dále ohrožován.

Posledním krokem je **zametení stop**. Zde je třeba smazat logovací soubory, vytvořit neviditelné složky a případně nainstalovat nějaký druh zadních vrátek (Backdoors). Většinu těchto úkonů dělají za útočníka nástroje nazývané rootkity.

4.2 Postup vedení penetračního testování

Rozdíl mezi skutečným útokem a penetračním testováním je především v dokumentování a oprávnění. V rámci penetračního testování je třeba celý průběh důkladně dokumentovat. Důležitý je výstup, který musí být srozumitelný i jiným než technickým pracovníkům (managementu, …).

Kroky penetračního testu mohou vypadat např. takto [4]:

1) **Průzkum**
2) **Analýza průzkumu**
3) **Vedení útoku**
4) **Analýza útoku**
5) **Dokumentace výsledků**
6) **Prezentace výsledků**

V prvním kroku s názvem **Průzkum** se tester pokouší získat informace o cíli (struktura sítě, používané OS atd.), tento krok odpovídá krokům 1) a 2) z kapitoly 3.1. Následuje krok **Analýza průzkumu**, kde se provádí analýza získaných informací z předchozího kroku a průběžná dokumentace. Ve třetím kroku **Vedení útoku** se tester pokouší získat přístup a následně eskalovat práva. Vedení útoku odpovídá krokům skutečného útoku 3), 4), 5) a 6) z kapitoly 3.1. V následujícím kroku čtyři následuje opět analýza s názvem **Analýza útoku**, zde se provádí rozbor všech kroků útoku a vzniká průběžná dokumentace. V krocích 5) a 6) je tvořen výstup testu a jeho prezentace zákazníkovi. Musí vzniknout srozumitelný dokument, který podá výsledky testu zainteresovaným osobám (viz 5.5).

Útok v rámci penetračního testování na rozdíl od skutečného útoku musí být systematický, dobře dokumentovaný a opakovatelně proveditelný. Z výše zmíněných důvodů existuje mnoho předpisů a standardů, které různým způsobem popisují způsob testování a vyhodnocování. Mezi ty

nejznámější patří NIST (National Institute of Standards and Technology), OSSTMM (Open Source Security Testing Metodology Manual) nebo OWASP Testing Guide (nyní ve verzi 3), který je přímo určený pro testování webových aplikací (Open Web Application Security Project).

5 Aplikační aspekty penetračního testování

Tato kapitola popisuje hlavní komunikační kanál, kterým je protokol HTTP, případně HTTPS. Těmito protokoly se komunikuje s testovanými webovými aplikacemi. Popis testovacích systémů, na kterých byly provozovány webové aplikace v rámci této práce, je uveden dále. Na konci kapitoly jsou popsány techniky a nástroje používané při penetračním testování.

5.1 HTTP protokol

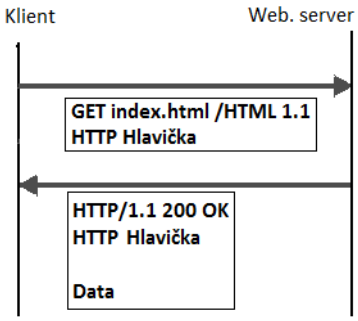
HTTP (HyperText Transfer Protocol) je síťový protokol pro přenos textových i multimediálních dat. Jedná se o protokol typu požadavek/odpověď. Klient (např. webový prohlížeč) odešle požadavek webovému serveru pro obsah (např. HTML stránku) a webový server odešle zpět odpověď s HTML (HyperText Markup Language) stránkou. Protokol HTTP pracuje na aplikační vrstvě. Dnes existují dva standardy HTTP, standard HTTP/1.0 a HTTP/1.1. Hlavní rozdíl mezi nimi je, že novější verze 1.1 dokáže držet TCP spojení pro více než jeden požadavek/odpověď. Speciálním případem HTTP protokolu je HTTPS (HyperText Transfer Protocol Secure). HTTPS komunikuje stejným způsobem jako HTTP s tím rozdílem, že komunikační kanál je šifrovaný pomocí SSL nebo TSL.

Ukázka HTTP komunikace požadavek/odpověď je ukázána na obrázku 5-1. Konkrétní HTTP požadavek v textu 5-1 používá metodu GET, která znamená, že požadujeme z adresy 192.168.133.177 objekt /index.html pomocí protokolu HTTP/1.1. HTTP protokol poskytuje i další metody:

- **GET** - Požadavek na uvedený objekt se zasláním případných dat v odpovědi.
- **HEAD** - To samé jako metoda GET, ale už nepředává data. Poskytne pouze metadata o požadovaném cíli (velikost, typ, datum změny, …).
- **POST** - Odesílá uživatelská data na server. Používá se například při odesílání formuláře na webu. S předaným objektem se pak zachází podobně jako při metodě GET. Data může odesílat i metoda GET, metoda POST se ale používá pro příliš velká data (víc než 512 bajtů, což je velikost požadavku GET) nebo pokud není vhodné přenášená data zobrazit jako součást URL.
- **PUT** - Nahraje data na server. Objekt je jméno vytvářeného souboru. Používá se velmi zřídka, pro nahrávání dat na server se běžně používá FTP nebo SCP/SSH.

- **DELETE -** Smaže uvedený objekt ze serveru. Jsou na to potřeba jistá oprávnění stejně jako u metody PUT.
- **TRACE -** Odešle kopii obdrženého požadavku zpět odesílateli, takže klient může zjistit, co na požadavku mění nebo přidávají servery, kterými požadavek prochází.
- **OPTIONS -** Dotaz na server, jaké podporuje metody.
- **CONNECT -** Spojí se s uvedeným objektem přes uvedený port. Používá se při průchodu skrze proxy pro ustanovení kanálu SSL.

Parametry Host a Connection dále určují vlastnosti HTTP požadavku. Např. Connection: close říká protokolu HTTP/1.1, že se má odeslat odpověď a ihned uzavřít spojení. HTTP parametry určují vlastnosti daného požadavku a jsou součástí celku, který se nazývá HTTP hlavička. Formát parametru je následující. Na začátku řádku je název zvoleného parametru, následuje dvojtečka a za ní požadovaná hodnota parametru. Těchto parametrů jsou desítky (viz [16]).



Obrázek 5-1: Průběh HTTP komunikace

```
telnet 192.168.133.177 80
GET /index.html HTTP/1.1
Host: 192.168.133.177
Connection: close
```

Text 5-1: HTTP požadavek

HTTP odpověď má podobný tvar jako HTTP požadavek. Odpověď je uvozena verzí HTTP protokolu, statusem odpovědi a popisem statusu. Kód statusu říká, jak dopadla operace, která byla požadována v HTTP požadavku. Kód 200 značí úspěch, kód 500 interní chybu serveru, těchto statusů je více a jsou důkladně zdokumentovány na stránkách komunit W3 (viz [26]). Následuje HTTP hlavička, prázdný řádek a samotná data, což je v tomto případě HTML stránka.

```
HTTP/1.1 200 OK
Date: Thu, 21 Apr 2011 14:58:41 GMT
Server: Apache/2.2.12 (Ubuntu)
Last-Modified: Tue, 23 Nov 2010 08:45:27 GMT
ETag: "7ede-b1-495b464f31e2e"
Accept-Ranges: bytes
Content-Length: 177
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

<html><body><h1>It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
</body></html>
```

Text 5-2: HTTP odpověď

5.2 Testovací systém

Testování implementované aplikace probíhalo na vlastní instalaci webového serveru, který běžel ve vizualizovaném prostředí GNU/Linuxu. Webový server byl nakonfigurovaný nejen pro protokol HTTP, ale i pro protokol HTTPS a umožňoval autentizaci Basic a Digest. Webové aplikace využívaly především jazyků HTML, PHP a CGI skripty byly napsané v programovacím jazyce C.

Další testování probíhalo na platformě Microsoft Windows v prostředí testovacího webového serveru Microsoft Visual Studia 2010. V tomto prostředí běžely testované webové aplikace vytvořené v jazyce ASP.NET. Prostředí umožnilo testování především zranitelností typických pro prostředí Microsoft Windows jako je zranitelnost SQL wildcards popsaná v kapitole 5.3.1.

Testování HTTP generátoru, zda je schopen zpracovávat reálné webové aplikace, bylo testováno také v prostředí internetu. Aplikace (penNet) byla testována na větším množství reálných webových aplikací na české i zahraniční doméně.

5.2.1 Příprava prostředí GNU/Linux

Byla zvolena dnes velmi oblíbená distribuce GNU/Linux Ubuntu (popis v [17]) ve verzi 9.10 (označení Karmic Coala).

Distribuce Ubuntu běžela ve virtualizovaném stroji vytvořeném pomocí aplikace VMware Player, což je volně dostupné prostředí. Společnost VMware poskytuje velkou škálu aplikací pro virtualizaci. Produkt VMware Player (detainí popis v [18]) je nejjednodušší alternativou, avšak volně dostupnou, která pro běžné domácí a testovací účely plně dostačuje. Placený produkt, který je alternativou, se nazývá VMware Workstation a umožňuje větší variabilitu nastavení atd. Mezi

společnosti, které poskytují konkurenční řešení, se řadí např. Oracle se svým produktem VM Virtual Box (detailní popis v [19]).

Jako webový server byl vybrán populární open source projekt Apache. Instalace webového serveru v prostředí Ubuntu ulehčuje použití správce balíčků, jak zobrazuje text 5-1.

```
jarda@ubuntu:/$ sudo apt-get install apache2
```

Text 5-1: Instalace apache2

Konfigurace autentizace Basic[5] pro Apache2 (dokumentace [11]) zahrnuje vytvoření souboru s uživatelskými jmény a odpovídajícími hesly, viz text 5-2. Soubor musí mít nastavena příslušná přístupová práva, aby nemohlo dojít k jejich prozrazení, např. systematickým procházením url adres webového serveru (angl. path traversal). Dalším krokem je konfigurace konfiguračního souboru httpd.conf (příklad konfigurace 5-3).

```
jarda@ubuntu:/$ htpasswd -c /usr/local/apache/passwd/basic jarda
jarda@ubuntu:/$ chmod 640 /usr/local/apache/passwd/basic
```

Text 5-2: Vytvoření souboru s hesly – Basic

```
<Directory /var/www>
      AuthType Basic
      AuthUserFile /usr/local/apache/passwd/basic
      AuthGroupFile /dev/null
      Require valid-user
</Directory>
```

Text 5-3: Konfigurační soubor httpd.conf a následný restart serveru – Basic

Obdobným způsobem se konfiguruje Digest[6] autentizace, rozdíl je pouze v programu, který vytváří soubor se jmény a hesly a přibývá několik dalších nastavení v souboru httpd.conf.

```
jarda@ubuntu:/$ htdigest -c /usr/local/apache/passwd/digest realm username

jarda@ubuntu:/$ chmod 640 /usr/local/apache/passwd/digest
```

Text 5-4: Vytvoření souboru s hesly – Digest

[5]Basic je způsob webové autentizace, kdy se heslo a jméno přenáší v otevřené podobě. Využívá se proto například ve spojení se šifrovaným protokolem HTTPS.
[6] Digest je způsob webové autentizace, kdy se heslo a jméno přenáší v šifrované podobě. Součástí šifrování je MD5 hash.

```
HTTPd.conf:
<Directory /var/www>
      AuthType Digest
      AuthName “Private“
      AuthDigestProvider file
      AuthUserFile /usr/local/apache/passwd/digest
      Require valid-user
</Directory>

jarda@ubuntu:/$ /etc/init.d/apache2 restart
```

Text 5-5: Konfigurační soubor httpd.conf a následný restart serveru – Digest

Při testování jsem také používal vynucení spojení přes šifrovaný kanál pomocí protokolu HTTPS. Toto chování se konfiguruje opět nastavením v souboru httpd.conf. Následující konfigurace 5-6 zobrazuje vynucení HTTPS spojení pro autentizaci Basic, kde se jméno a heslo přenáší v otevřené podobě.

```
<Directory /var/www>
      SSLRequireSSL
      AuthType Basic
      AuthUserFile /usr/local/apache/passwd/basic
      AuthGroupFile /dev/null
      Require valid-user
</Directory>
```

Text 5-6: Vynucení šifrovaného spojení při Basic autentizaci

Testování také probíhalo na lokální instalaci systému Wordpress. Wordpress je open source redakční publikační systém napsaný v PHP[7] a MySQL[8]. Instalace vyžaduje nakonfigurované prostředí MySQL a přítomnost PHP, instalace těchto součástí v prostředí GNU/Linux Ubuntu je uvedena v 5-7. Instalace samotného redakčního systému Wordpress je možno provést dvěma způsoby. Manuální konfigurací nebo průvodcem ve webovém prohlížeči. Já jsem zvolil druhou možnost tedy instalaci pomocí průvodce. Oba způsoby instalace jsou uvedeny na webové stránce [20]. Dále byl nainstalován plugin user_locker, pro uzamykání uživatelských účtů po určitém počtu chybných přihlášení.

[7] PHP je skriptovací jazyk využívaný pro tvorbu interaktivních webových stránek. Základní tutoriál je dostupný na [31].
[8] MySQL je volně dostupný databázový systém spravovaný společností Oracle.

```
jarda@ubuntu:/$ sudo apt-get install php5
jarda@ubuntu:/$ sudo apt-get install libapache2-mod-php5
jarda@ubuntu:/$ sudo /etc/init.d/apache2 restart

jarda@ubuntu:/$ sudo apt-get install mysql-server
jarda@ubuntu:/$ sudo apt-get install php5-mysql
jarda@ubuntu:/$ mysqladmin create <databasename>
```

Text 5-7: Instalace PHP a MySQL

5.2.2 Příprava prostředí Microsoft Windows

Virtuální stroj VMWare Player a testování implementované aplikace probíhalo na systému MS Windows 7 Ultimate. Testovací webové aplikace běžely přímo v testovacím webovém serveru nástroje Visual Studio 2010. Webové aplikace tedy běžely v prostředí .NET Framework, které je popsané v kapitole 8.1. Mezi nevýhody takto provozované webové aplikace patří především fakt, že nelze navázat větší množství konkurentních spojení a mají delší dobu odezvy. Ovšem převažuje jednoduchost provozu a fakt, že pro daný účel je plně dostačující. Pro testování jsem napsal vlastní aplikace. V této práci využívám především aplikace se zranitelností typu SQL wildcards nebo nesprávně provedeného logování. Existují i uměle vytvořená prostředí jako například OWASP WebGoat[9] pro testování webových zranitelností. Já jsem prošel oběma fázemi, tj. tvorbou webových aplikací a jejich následného testování.

5.3 Nástroje a techniky pro penetrační testování

Kapitola popisuje techniky a nástroje používané při testování webových zranitelností, DoS zranitelností nevyjímaje. Na těchto nástrojích a technikách je založeno testování v následující kapitole OWASP.

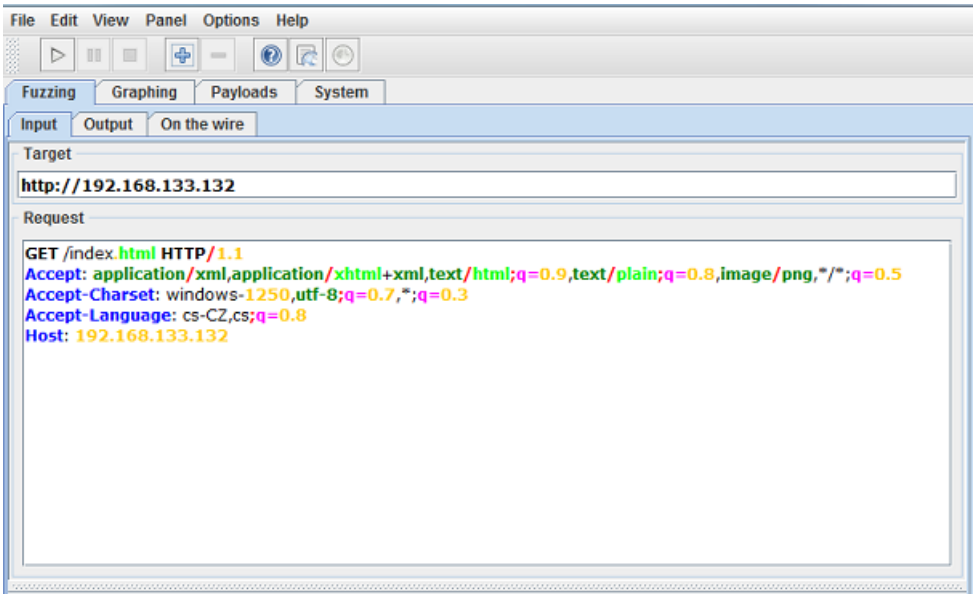
5.3.1 Generátor HTTP požadavků

Základním stavebním kamenem pro testování webových aplikací je generátor HTTP požadavků. Generátor posílá HTTP požadavky na webový server a zobrazuje odpovědi, případně přidané statistiky. Umožňuje zadávat položky HTTP hlavičky a modifikace jejich hodnot.

Obrázek 5-2 zobrazuje aplikaci JBroFuzzer (HTTP generátor požadavků) spravovanou organizací OWASP (viz kapitola 6). Tato aplikace bude použita dále v práci a svoji strukturou představuje klasický HTTP generátor požadavků. Umožňuje i pokročilé funkce, které budou popsány

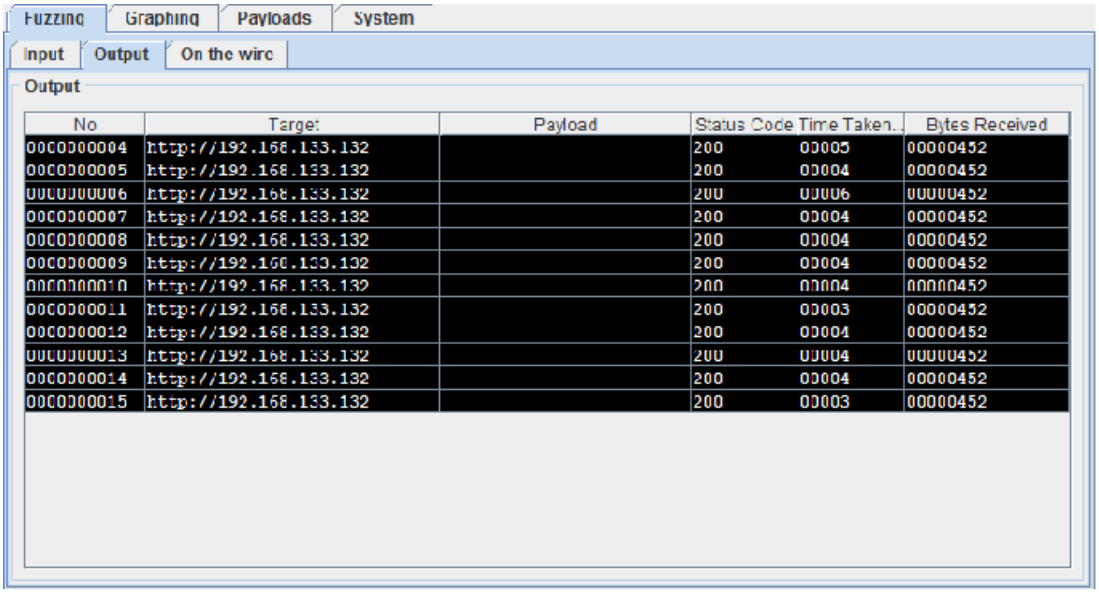
[9] WebGoat je webová aplikace vytvořená v J2EE pro výuku bezpečnosti webových aplikací. [32]

dále. Hlavní okno aplikace je rozděleno do tří záložek. První (Input) umožňuje vytvořit HTTP požadavek v textové podobě (RAW formát – surový, nezpracovaný), viz obrázek 5-2.



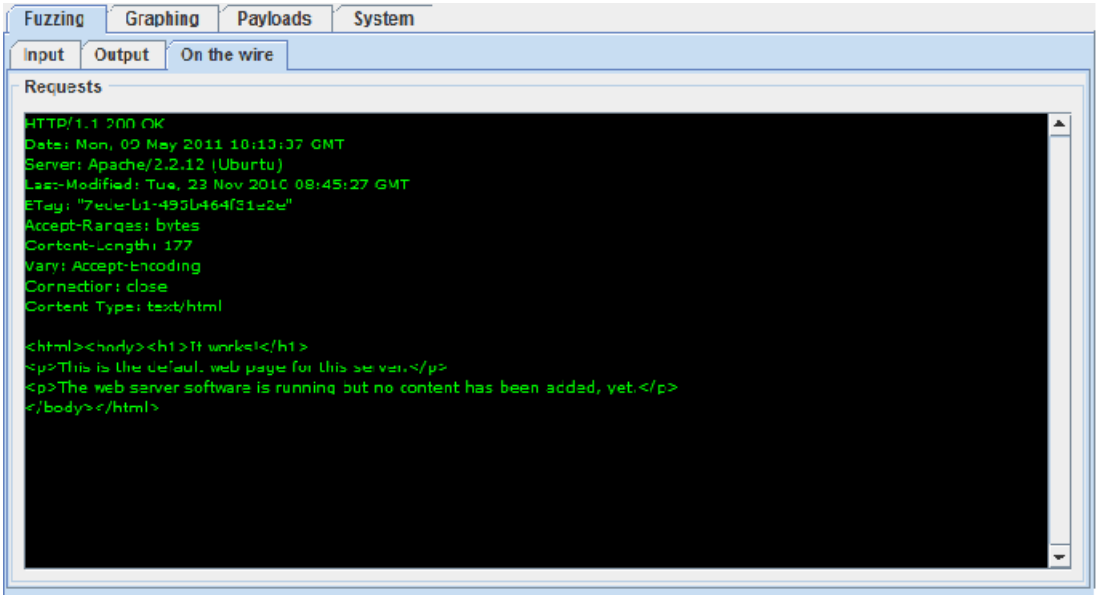
Obrázek 5-2: JBroFuzz – tvorba HTTP požadavku

Druhá záložka (Output) poskytuje informace o přijatých odpovědích (viz obrázek 5-3), jako je velikost, doba odezvy a status. Tyto informace jsou důležité při testování, ukazují nám, jak reaguje HTTP server na rozdílné HTTP požadavky.



Obrázek 5-3:JBroFuzz – statistika HTTP odpovědí

JBroFuzzer umožňuje náhled přijaté HTTP odpovědi. Tato možnost se vyskytuje pod záložkou „On the wire“, viz obrázek 5-4. V některých případech je důležité vidět výslednou strukturu odpovídající HTTP odpovědi, nejen souhrnnou statistiku HTTP odpovědí.



Obrázek 5-4: JBroFuzz - Náhled HTTP odpovědi

Aplikace navržená v rámci diplomové práce by měla poskytnout podobnou funkčnost jako aplikace JBroFuzz. Ovšem s rozšířenou možností nastavení.

5.3.2 Fuzzer

Fuzing je technika, která manipuluje se vstupními parametry aplikace za účelem nalezení chyb. Fuzzing se typicky používá pro odhalení chyb v programech, protokolech atd. Pojem Fuzzing pochází z roku 1989, kdy byl vyvinut profesorem Bartonem Millerem, který pomocí fuzzingu dokázal, že v té době moderní operační systémy byly náchylné na jednoduchý fuzzing, viz [21]. Nástroje, které provádějí fuzzing se nazývají fuzzery. V současnosti se používají fuzzery dvou typů, rekurzivní (angl. recursive) a nahrazovací (replacive). Takto definuje fuzzer organizace OWASP (viz kapitola 5) pro testování webových aplikací.

Recursive fuzzer iterativně vytváří řetězce daného jazyka na jeho abecedě a dosazuje je za daný parametr. Formálně zapsaný příklad jazyka recursive fuzzeru a jím generované řetězce mohou vypadat následovně.

L = {w | w = {0,1,2,3,4,5,6,7,8,9}$^{+}$}

w	**Testovaná URL**
0	www.test.com/id=0
1	www.test.com/id=1
2	www.test.com/id=2
..	
12	www.test.com/id=12
..	
25874	www.test.com/id=25874
..	

Text 5-8: : Funkce iterativního fuzzeru

Replacive fuzzer iterativně dosazuje řetězce z předem definované množiny prvků za daný parametr.

L = {“admin“,“admin007“,“user“,…}

www.test.com/usr=admin

www.test.com/usr=admin007

www.test.com/usr=user

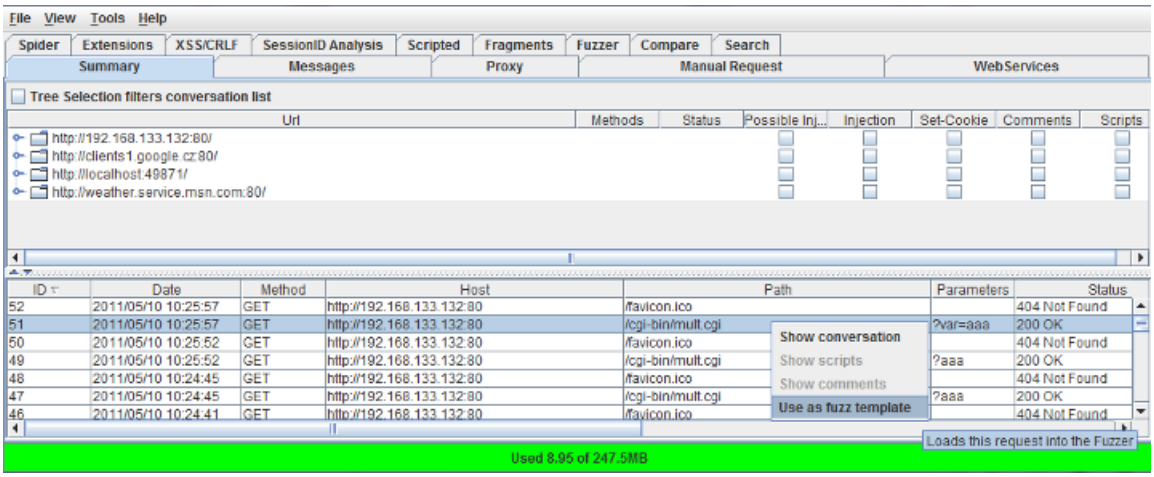
Text 5-9: Funkce nahrazovacího fuzzeru

V rámci projektu OWASP jsou vyvíjeny tři aplikace s možností fuzzingu, WebScarab, JBroFuzz a WSFuzzer. Poslední jmenovaný je zaměřený na Webservices[10], takže není cílen na námi požadovanou oblast, protože OWASP tento druh testů řadí do vlastní kategorie, viz kapitola 5.

5.3.2.1 WebScarab

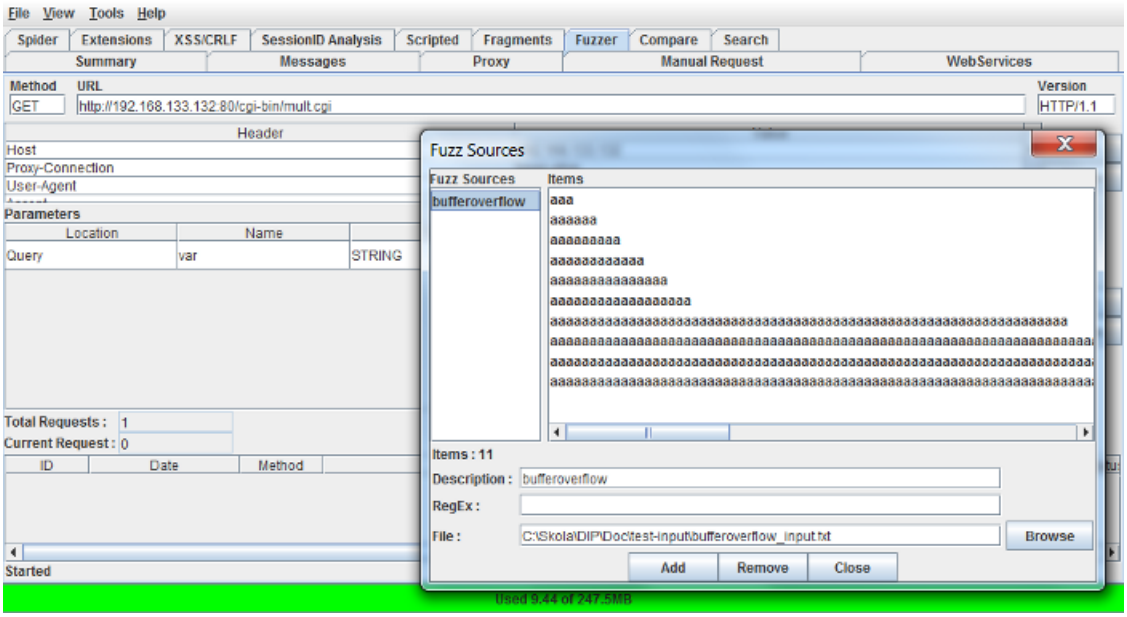
WebScarab je velice univerzální nástroj určený pro analýzu webových aplikací, které komunikují pomocí protokolu HTTP(s). Je napsaný v jazyce Java. WebScarab má několik modulů a jedním z nich je právě fuzzer. WebScarab umožňuje replacive fuzzing (vstupem je soubor předefinovaných hodnot) a recurive fuzzing (regulární výraz).

[10] WebService je technologie pro vzdálené volání procedur pomocí přenosu zpráv v jazyku XML(Extensible Markup Language) protokolem HTTP, viz [30]. XML je textový, strojově čitelný formát podobný HTML pro výměnu dat.



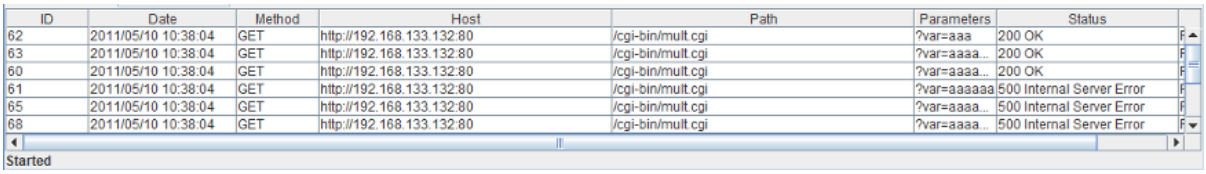
Obrázek 5-5: WebScarab – proxy

Další velice důležitou funkcí je HTTP proxy, která umí zachycený HTTP dotaz rovnou připravit pro fuzzer. Obrázek 5-5 znázorňuje použití proxy serveru a následující obrázek 3-6 zobrazuje definici replacive fuzzeru ze souboru v aplikaci WebScarab.



Obrázek 5-6: WebScarab – fuzzer

Obrázek 5-6 popisuje test na buffer overflow (popsáno v kapitole 6). WebScarab vrací status HTTP odpovědi, výsledek testu je zobrazen na obrázeku 5-7. Obrázek představuje formát výpisu odpovědí pomocí aplikace WebScarab. Dále je patrné, že pro daný test začne webový server vracet HTTP kód odpovědi 500, interní chyba.



ID	Date	Method	Host	Path	Parameters	Status
62	2011/05/10 10:38:04	GET	http://192.168.133.132:80	/cgi-bin/mult.cgi	?var=aaa	200 OK
63	2011/05/10 10:38:04	GET	http://192.168.133.132:80	/cgi-bin/mult.cgi	?var=aaaa...	200 OK
60	2011/05/10 10:38:04	GET	http://192.168.133.132:80	/cgi-bin/mult.cgi	?var=aaaa...	200 OK
61	2011/05/10 10:38:04	GET	http://192.168.133.132:80	/cgi-bin/mult.cgi	?var=aaaaaa	500 Internal Server Error
65	2011/05/10 10:38:04	GET	http://192.168.133.132:80	/cgi-bin/mult.cgi	?var=aaaa...	500 Internal Server Error
68	2011/05/10 10:38:04	GET	http://192.168.133.132:80	/cgi-bin/mult.cgi	?var=aaaa...	500 Internal Server Error

Started

Obrázek 5-7: WebScarab - Výpis odpovědí

Nevýhodou aplikace WebScarab je absence údajů o časové odezvě a zobrazení výsledků v grafické podobě. Problémem může být také stabilita aplikace. Výhodu lze spatřit v otevřeném zdrojovém kódu.

5.3.2.2 JBroFuzz

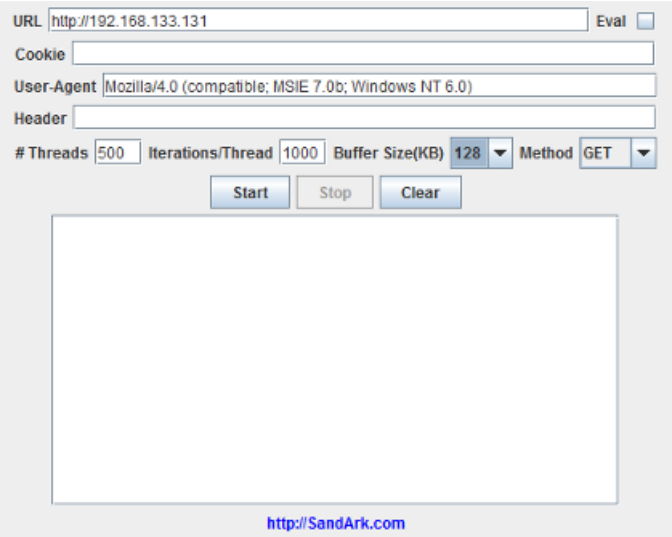
Aplikace JBroFuzz je čistý HTTP(s) fuzzer s velkým počtem předdefinovaných rekurzivních i nahrazovacích fuzzerů. Velké množství již definovaných fuzzerů umožňuje okamžitě testovat webové aplikace bez nutnosti definovat si vlastní vstupní hodnoty. Na druhou stranu definovat nový typ fuzzeru není zcela triviální. Použití aplikace JBroFuzz jako HTTP generátoru je popsáno v kapitole 5.3.1 a následně ve spojení s fuzzingem je používán v kapitole 6.2. Umožňuje sledovat HTTP status, velikost odezvy i velikost odpovědi a následně vynést v grafické podobě, což je využito a předvedeno v již zmíněné následující kapitole. Opět se jedná o aplikaci s otevřeným zdrojovým kódem a napsanou v jazyce Java. Neobsahuje proxy, ale ten je možné nahradit některým open source projektem. JBroFuzz poskytuje grafický výstup i dobu odezvy každého požadavku. Obrázek 5-2 zobrazuje konstrukci HTTP dotazu v JBroFuzz.

5.3.3 Strest test

Stress test je technika používaná pro testování software a hardware. Stress testy se používají pro testování stability systému tím, že vytěžují daný systém stálou zátěží nebo v určených intervalech. Pro testování webových aplikací existuje celá řada aplikací a skriptů.

5.3.3.1 URLStress

Pro útok DoS, na nesprávně nakonfigurovaný webový server, lze použít obyčejný stress test určený pro testování webových serverů, viz aplikace URLStress [22]. Aplikace URLStress umožňuje posílání velkého množství HTTP požadavků na webový server. Uživatel má možnost definovat HTTP hlavičku, množství otevřených klientů atd.



Obrázek 5-8: Konfigurace aplikace URLStress

Zahlcení webového serveru zobrazuje Obrázek 5-9, procesorový čas strávený v systémové oblasti je cca 85 procent. URLStress byl nakonfigurován pro odeslání jednoho tisíce požadavků v pěti stech vláknech. Zahlcení web serveru dokazuje chybová zpráva ve webovém prohlížeči (viz Obrázek 5-10), což znamená, že webový server není schopen vyřizovat legitimní požadavky.

```
top - 14:38:20 up  9:16,  2 users,  load average: 120.08, 57.16, 22.14
Tasks: 287 total,  60 running, 227 sleeping,   0 stopped,   0 zombie
Cpu(s): 18.1%us, 85.7%sy,  0.0%ni,  0.0%id,  0.0%wa,  1.4%hi, 24.9%si,  0.0%st
Mem:    509336k total,   490384k used,    18952k free,    89072k buffers
Swap:   915664k total,     3472k used,   912192k free,   176916k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 4714 www-data  20   0 34904 4492 1056 R  1.1  0.9   0:01.32 apache2
 4745 www-data  20   0 34904 4492 1056 R  1.1  0.9   0:01.21 apache2
 4791 www-data  20   0 34904 4492 1056 R  1.1  0.9   0:01.11 apache2
 4832 www-data  20   0 34904 4492 1056 D  1.1  0.9   0:00.93 apache2
 4838 www-data  20   0 34904 4492 1056 D  1.1  0.9   0:00.99 apache2
 2507 www-data  20   0 34904 4492 1056 D  0.9  0.9   0:02.47 apache2
 4701 www-data  20   0 34904 4492 1056 S  0.9  0.9   0:01.93 apache2
 4702 www-data  20   0 34904 4492 1056 R  0.9  0.9   0:01.80 apache2
 4703 www-data  20   0 34904 4492 1056 R  0.9  0.9   0:01.80 apache2
 4720 www-data  20   0 34904 4492 1056 R  0.9  0.9   0:01.05 apache2
 4730 www-data  20   0 34904 4492 1056 D  0.9  0.9   0:01.22 apache2
 4732 www-data  20   0 34904 4492 1056 D  0.9  0.9   0:01.18 apache2
 4734 www-data  20   0 34904 4492 1056 R  0.9  0.9   0:01.12 apache2
 4737 www-data  20   0 34904 4492 1056 R  0.9  0.9   0:01.17 apache2
 4739 www-data  20   0 34904 4492 1056 R  0.9  0.9   0:01.11 apache2
 4740 www-data  20   0 34904 4492 1056 D  0.9  0.9   0:01.22 apache2
 4746 www-data  20   0 34904 4492 1056 R  0.9  0.9   0:00.85 apache2
```

Obrázek 5-9: Ukázka zahlcení webového serveru pomocí aplikace URLStress - top

Obrázek 5-10: Webový server je zahlcen, neposkytl odpověď v legitimním čase

Možná obrana by mohla být konfigurace maximálního počtu otevřených spojení od jednoho klienta. Konfiguraci iptables pro omezení počtu konkurentních spojení z jedné IP adresy je ukázán v textu 5-10. Dále je možné omezit maximální počet HTTP spojení na konkrétním webovém serveru. Webový server Apache toto omezení umožňuje nastavit v konfiguračním souboru HTTPd.conf.

```
/sbin/iptables  -A INPUT -p tcp --syn --dport 80 -m connlimit --connlimit-above 2 --
connlimit-mask 24 -j DROP
```

Text 5-10: Konfigurace iptables - max. počet spojení na jednu IP

6 OWASP

OWASP (Open Web Application Security Project) je nevýdělečná organizace (Předseda představenstva Matt Tesauro), tato organizace je podporována firmami jako je IBM, Oracle nebo třeba Nokia. Tato kapitola představuje její projekt OWASP Testing Guide v3. Stručně je popsán průběh testu a rozsah testovaných oblastí. Blíže je zdokumentováno testování webových aplikací na zranitelnosti typu DoS dle metodologie OWASP Testing Guide v3. Jsou zde uvedeny ukázky zdrojových kódů a grafické statistiky získané pomocí aplikací popsaných v kapitole 5.3.

6.1 OWASP Testing Guide v3

OWASP Testing Guide v3 je příručka pro penetrační testování webových aplikací. Verze 3 byla vydána v roce 2008, další verze je plánována na rok 2011.

Testovací model podle OWASP Testing Guide v3 se skládá ze tří entit:

- Tester: osoba, která provádí penetrační testování.
- Nástroje a metodologie: jádrem je příručka Testing Guide v3 a další projekty pod záštitou OWASP.
- Cíl zájmu: Testing Guide v3 je založen na testování stylem černá skříňka (black box), případně gray box.

Samotná testovací fáze je rozdělena do dvou částí. První je pasivní. Tester zkouší funkčnost dané webové aplikace, zkoumá HTTP hlavičky, parametry, cookies atd. Další fáze je aktivní. V této fázi tester testuje aktivně zranitelnosti dle OWASP Testing Guide v3.

Aktivní testy rozlišují devět kategorií:

- Configuration Management
- Business Logic
- Authentication
- Authorization
- Session Management
- Data Validation
- **Denial of Service**
- Web Services
- Ajax

Configuration Management je sada testů zabývající se testováním nastavení dané webové aplikace a aplikací, které jí umožňují chod. Testuje se např. na možnost stáhnout staré zálohy souborů, nastavení SSL/TSL, konfigurace HTTP metod atd.

Testy s názvem **Business Logic** se zabývají tím, zda webová aplikace neumožňuje provedení akce, která není přípustná z obchodního hlediska. Např. pokud zadáme záporný počet položek v obchodě, tak můžeme dané zboží získat zadarmo atd.

Testy ze sady **Authentication** se zabývají procesem autentizace. Autentizace je proces ověření proklamované identity subjektu. Testuje se zde na možnost získání uživatelských jmen, možnost uhodnout heslo, získat ho pomocí Brutal force atd.

Authorization je proces při kterém se ověřuje, zda daný subjekt má práva pro využívání zvoleného objektu. Testuje se zde např. možnost eskalace privilegií (viz kapitola 4).

Jádrem každé webové aplikace je způsob, kterým se udržuje její stav a tím řídí uživatelské interakce. Testy zaměřené na **Session management** se týkají testování prostředků zvaných session a cookies umožňující aplikacím udržet stav.

Nejvíce bezpečnostních zranitelností webových aplikací pochází z nedostatečné kontroly vstupních parametrů. Patří sem například zranitelnost XSS (cross site scripting), která umožňuje spouštět externí kód prostřednictvím webové aplikace. Testy zaměřené na tento druh zranitelností se nazývají **Data Validation**.

Testy nazvané **Denial of Service** jsou zaměřeny na hledání chyb, které zapříčiňují odepření služby autorizovanému subjektu, viz kapitola 3.

SOA, na služby orientovaná architektura je dnes velmi rychle se prosazující technologie. Základem jsou kooperující služby, které komunikují pomocí XML zpráv. Testy na tuto oblast se nazývají právě **Web Services**.

AJAX je zkratka pro Asynchronous JavaScript and XML. Zmíněná technika je používána pro větší uživatelské pohodlí při užívání webových aplikací, ty se díky AJAXu chovají podobně jako desktopové. AJAX umožňuje asynchronně měnit pouze část webové aplikace za použití XMLHTTPRequest a JavaScriptu.

Každá kategorie obsahuje sadu tematicky zaměřených testů. Každý test má svůj jedinečný identifikátor (referenční číslo), pomocí něhož je identifikován ve výsledné zprávě. Formát identifikátoru je následující, OWASP-XX-YYY, kde XX označuje kategorii testu a YYY pořadové číslo testu v dané kategorii. Např. z kategorie Denial of Service druhý test (Locking customer account) je označen identifikátorem OWASP-DS-002.

6.2 Testy Denial of Service

Kapitola popisuje testy na zranitelnost DoS dle metodologie OWASP Testing Guide v3. Jsou zde uvedeny ukázky zdrojových kódů a grafické statistiky získané pomocí aplikací popsaných v kapitole 5.3. Testování zranitelností probíhalo v prostředím popsaném v kapitole 5.2.

6.2.1 SQL wildcard (OWASP-DS-001)

Test se zaměřuje na SQL dotazy na databázi, které jsou více náročné na CPU než jiné. Tuto zranitelnost lze najít především ve vyhledávacích funkcích webových aplikací.

Zranitelnost SQL wildcard je známá především z Microsoft SQL serveru, kde klíčové slovo LIKE podporuje speciální znaky (wildcard) „[]“, „[^]“, „„“, „_“ a „%“ a s jejich použitím může stoupat výpočetní náročnost dotazu.

Běžný SQL dotaz vytvořený v MS Visual Studiem je znázorněn zdrojovým kódem 6-1. Graf 6-1 zobrazuje dobu vykonávání dotazů s řetězci v textu 6-2. První řetězec označuje existující prvek v databázi, následuje neexistující záznam a další dva představují řetězce obsahující „nesmyslné“ uspořádání wildcard znaků.

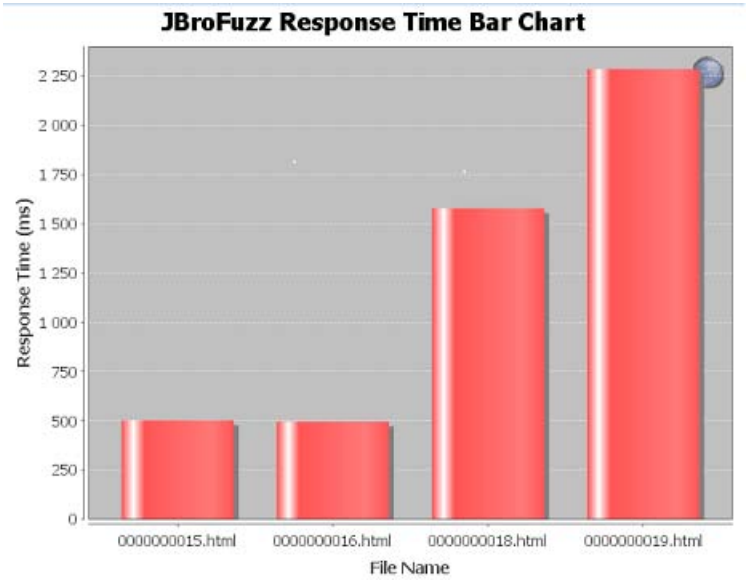
```
SELECT * FROM [zipcode] WHERE ([city] LIKE '%' + @city + '%')
```

Zdrojový kód 6-1: SQL dotaz vytvořený MS Visual Studiem

```
LAUREL
aaaaaaa
%_[aaaaacccvbbaaaa[! -z]@$!_%
%_[aaaabbbbbbbbbccaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa[! -z]@$!_%
```

Text 6-2: Sada řetězců pro testování zranitelnosti SQL Wildcards

Tyto řetězce byly posílány pomocí aplikace JBroFuzzer, která zobrazuje výsledné doby odezev HTTP dotazů a umí je dále zobrazit v grafické podobě (viz Graf 6-1). Aplikace WebScarab nevede statistiku doby odezvy, proto se pro tento účel nehodí. Test probíhal na aplikaci, kterou jsem implementoval pomoci frameworku .NET, viz kapitola 8.1.



Graf 6-1: JBroFuzz - Doba vykonávání SQL dotazů

Z grafu 6-1 vyplívá, že daná webová aplikace je pravděpodobně náchylná na SQL wildcard útok. Svědčí o tom vzrůstající doba odezvy webové aplikace, je ovšem třeba dále otestovat další komplikovanější řetězce a test provést ve větším rozsahu. V publikaci [23] se podrobně pojednává o útoku i o sestavování řetězců.

Pokud má tester nebo správce aplikace přístup k logům HTTP serveru, tak by se měl zaměřit na záznamy s vysokou dobou odezvy a odhalit příčinu v odpovídající části webové aplikace.

6.2.2 Uzamykání uživatelských účtů (OWASP-DS-002)

Některé webové aplikace z bezpečnostních důvodů uzamykají uživatelské účty po několika neúspěšných pokusech. Při návrhu aplikace je nutné zvážit dopady pohodlí vs. bezpečnosti. Lze využít systém CAPTCHA, který může zabránit automatizovanému útoku, ale opět klesá pohodlí při používání webové aplikace.

Pokud tester má validní seznam uživatelů není problém několikrát zadat smyšlené heslo a zjistit, zda dojde k zablokování účtu. Pokud ovšem uživatelské účty nejsou k dispozici, je třeba je získat. Ideální vstupní branou jsou formuláře pro přihlášení do systému, vytvoření nového uživatele a resetování hesla. Tyto formuláře mohou poskytnout cenné informace, zvláště pokud jsou chybová hlášení příliš konkrétní. Různá hlášení pro existující a neexistující uživatele mohou vést k odhalení existujících uživatelských účtů a k jejich následnému zablokování.

6.2.3 Buffer overflow (OWASP-DS-003)

Tato zranitelnost se vyskytuje především v aplikacích napsaných v programovacích jazycích, kde má programátor na starosti práci přímo s paměťovým prostorem. Nejběžněji používané jazyky jako ASP.NET a PHP v podstatě neumožňují zranitelnost tohoto typu. Až do doby, kdy je volán nějaký externí kód v podobě COM komponent nebo např. CGI skriptu napsaném v C/C++.

Zranitelnosti typu buffer overflow jsou v podstatě trojího typu. Prvním typem je **heap overflow**. V případě heap overflow dojde k přepsání oblasti paměti, která uchovává velikost dynamicky alokované paměti a při pokusu o uvolnění paměti pravděpodobně dojde k Segmentation fault.

```
void processURLParam(char *param) {
        char *tmp = NULL;
        tmp = (char *) malloc(sizeof(char) * 10);
        strcpy(tmp, param);          // Pravděpodobnost přepsání informací o paměti
        …
        free(tmp)                    // Může nastat Segmentation fault
}
```

Zdrojový kód 6-3: Heap overflow

Dalším typem je **stack overflow**. V případě stack overflow dochází k zápisu za hranice staticky alokovaného pole. Z tohoto důvodu (stejně jako v předchozím případě) se doporučuje při programování v jazycích C/C++ používání bezpečných příkazů, jako je např. strncpy.

```
void processURLParam(char *param){
        char tmp[10];
        strcpy(tmp, param);
        …
}
```

Zdrojový kód 6-4: Stack overflow

Posledním typem buffer overflow je **format string**. V případě využití jazyka C může být tato zranitelnost napadena v případě, že není použit formátovací řetězec s následným pokusem vypsat sekvenci %x následovanou sekvencí %n.

```
void showResult(char *res){
        …
        printf("<TITLE>Buffer ovewflow</TITLE>\n");
        printf(res);
        …
}
```

Zdrojový kód 6-5: String overflow

Testovat webové aplikace (např. následující webová adresa) lze pomocí generátoru HTTP požadavků ve spojení s technikou známou jako fuzzing (viz následující kapitola). Test lze provést např. tak, že místo řetězce abc budeme posílat různě dlouhé řetězce. Testovaná aplikace v jazyce C je zobrazen na zdrojovém kódu 6-6. Tento kód byl přeložen jako CGI skript a testování probíhalo ve vlastním prostředí popsaném v kapitole 5.2.

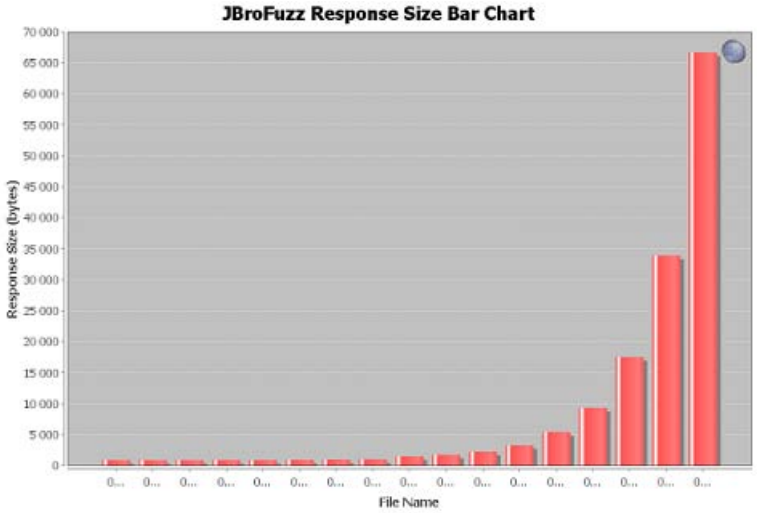
```
HTTP://localhost/cgi-bin/test.cgi?n=abc
```

Text 6-1: URL adresa s parametrem

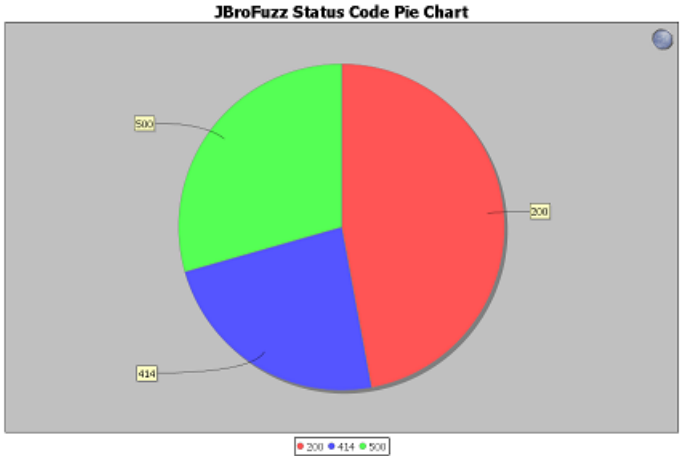
```
void main(void)
{
  char n[100];
  char *data;

  printf("%s%c%c\n", "Content-Type:text/html;charset=iso-8859-1",13,10);
  printf("<TITLE>Buffer ovewflow</TITLE>\n");
  printf("<H3>Buffer overflow test</H3>\n");

  data = getenv("QUERY_STRING");

  if (data == NULL) {
         printf("<p>getenv() error</p>");
         return 1;
  }

  strcpy(n, data);
  printf("<p>Buffer lenght: %d</p>", strlen(n));

  free(data);
}
```

Zdrojový kód 6-6: Testovací CGI skript

Následující graf 6-2 zobrazuje délku dat odeslaných pro zpracování CGI skriptu (viz Zdrojový kód 6-6) a graf 6-3 zobrazuje chování testované aplikace, tyto grafy jsou opět výstupem aplikace JBroFuzz. Nejdříve server vrací kód 200, všechno v pořádku. Následuje kód 500, interní chyba (lze předpokládat, že aplikace byla náchylná na buffer overflow). Kód 414 poukazuje na překročení maximální délky URL.

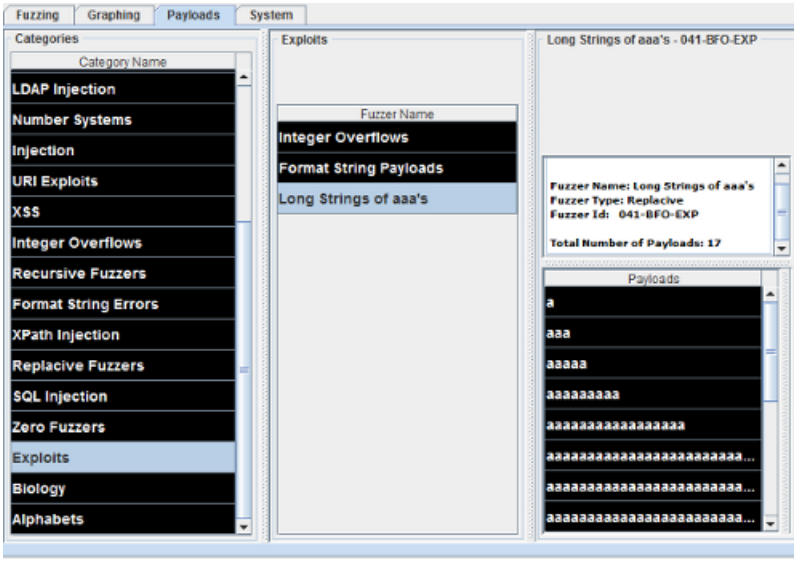


Graf 6-2: JBroFuzz - Velikost testujícího řetězce



Graf 6-3: JBroFuzz - Kód odezvy serveru

Testování probíhalo pomocí aplikace JBroFuzz s použitím integrovaného fuzzeru Long Strings of aaa (viz Obrázek 6-1). Fuzzer Long Strings of aaa generuje postupně řetězce se zvětšující se délkou (jak zobrazuje Graf 6-2) a ty nahrazuje za zvolený parametr v HTTP požadavku. Díky velkému množství integrovaných fuzzerů, lze JBroFuzz rychle použít pro daný účel.



Obrázek 6-1: JBroFuzz - Long Strings of aaa

6.2.4 Uživatelem alokované objekty (OWASP-DS-004)

Test se zaměřuje na možnost vyčerpat zdroje serveru. V daném případě se jedná o dostupnou paměť alokováním velkého počtu objektů. Pokud ve webové aplikaci má uživatel možnost přímo nebo nepřímo alokovat nějaké typy objektů, tak může dojít k DoS.

Zajímavé pro uplatnění těchto testů jsou textová pole, kde uživatel může přímo zadávat počet položek. Následuje událost tlačítka, která neošetřuje horní hranici alokovaných objektů v jazyce C#.

```
protected void btnAllocate_Click(object sender, EventArgs e)
{
      if (txtNum.Text != string.Empty)
      {
            myGreatObj[] obj = new myGreatObj[Convert.ToUInt32(txtNum.Text)];
            lblResult.Text = "It was created " + Convert.ToUInt32(txtNum.Text) + "
object(s).";
      }
}
```

Zdrojový kód 6-7: Alokace objektů

6.2.5 Uživatelský vstup jako čítač cyklu (OWASP-DS-005)

Podobně jako v předchozím případě, kdy uživatel webové aplikace může přímo nebo nepřímo ovlivnit počet opakování nějaké operace může nastat DoS. Špatně napsaná aplikace nemá shora omezený počet opakování, tudíž může nastat velký počet opakování nějaké operace a vytížení CPU.

Při hledání vhodné vstupní brány pro začátek testu jsou opět vhodná textová pole, kam se zadávají číselné hodnoty a skrytá pole. Pokud se zvyšující se hodnotou, vzrůstá doba odezvy, je

pravděpodobné, že webová aplikace bude na zranitelnost náchylná. Zranitelný kód napsaný v C# by mohl vypadat jako zdrojový kód 6-8.

```
protected void btnAllocate_Click(object sender, EventArgs e)
{
      for (UInt32 i = 0; i < System.Convert.ToUInt32(txtNum.Text); i++)
      {
                  // Any difficult process
      }
}
```

Zdrojový kód 6-8: Vstup jako čítač

6.2.6 Zapisování uživatelem poskytnutých dat na disk (OWASP-DS-006)

Tento typ testu se opět jako předchozí zaměřuje na možnost vyčerpání zdrojů. V případě, že není správně navržena logika logování, např. není vyhrazen samostatný oddíl pro logovací záznamy a není zvolena vhodná logika logovacích záznamů, může dojít k vyčerpání diskového prostoru. Nemusí se jednat jenom o logovací záznamy. V podstatě kdykoliv, kdy uživatel má možnost přímo nebo nepřímo uložit na server neomezené nebo velké množství dat, může nastat vyčerpání diskového prostoru.

Zranitelnost typu OWASP-DS-006 je velice těžké odhalit a manuálně se jedná o téměř neproveditelný úkol. V případě, že tester nemá žádné přidané informace o stavu serveru, je třeba vhodný skript a dostatečnou časovou dotaci. Při testování je vhodné využít informace z protokolu SNMP. To velice ulehčí testování a rychle umožní odhalit efekt skriptu. Následující zdrojový kód 6-9 zobrazuje chybně provedené logování v C#, ukládá se celé vstupní pole. Tester může v tomto případě ukládat libovolné množství dat, což může vést k vyčerpání diskového prostoru.

```
try
{
     // Bad data format
}
catch (Exception ex)
{
      logToFile(ex.Message + ":" + txtInput.Value, logFile)
}
```

Zdrojový kód 6-9: Metoda logování

6.2.7 Chybné uvolňování prostředků (OWASP-DS-007)

Jak již název říká, test OWASP-DS-007 se zabývá chybným uvolňováním prostředků. Může se jedna o **soubory**, **paměť**, **databázové objekty** atd.

Pokud je otevřen soubor programem, je následně uzamknut pro ostatní. Není-li po ukončení práce uvolněn, tak nemůže být jinde použit, zároveň program může mít otevřen jenom určitý počet souborů a v případě, že nejsou uvolňovány, tak se časem narazí na max. hodnotu otevřených deskriptorů. Chybné uvolnění prostředků může nastat např., pokud je vyvolána výjimka a nejsou v ní řádně uvolněny všechny prostředky.

Chybné uvolňování paměti je dominantou především v C/C++, kde programátor má plnou kontrolu nad správou paměti. Neefektivní uvolňování paměti pomocí garbage collectoru[11] může také způsobit problémy.

Chybné uvolňování DB objektu principiálně odpovídá chybnému uvolňování souborů. Může dojít k otevření max. počtu spojení a další již nelze navázat.

Následující kód v C# zobrazuje správné uvolňování DB objektu. Pokud by byla vynechána finally klauzule, tak by mohlo dojít k vyčerpání max. počtu otevřených spojení.

```
try
{
      myConnection.Open();
      reader = myCommand.ExecuteReader();
      myDataGrid.DataSource = reader;
      myDataGrid.DataBind();
}
catch(Exception ex)
{
            // Catches and logs the exception
}
finally
{
      reader.Close();
      myConnection.Close();
}
```

Zdrojový kód 6-10: Správná práce s uvolňováním objektů

Nalezení chybného uvolňování prostředků vyžaduje dostatečnou časovou dotaci a v ideálním případě by tester měl mít k dispozici SNMP informace nebo přístup k filesystemu.

[11] Garbage collector je část běhového prostředí aplikace (Java, .NET, …), která se stará o uvolňování již nepoužívané paměti aplikace.

6.2.8 Ukládání velkého množství dat v Session (OWASP-DS-008)

Cílem testu OWASP-DS-008 je zjistit, zda není možné uvést mimo provoz server prostřednictvím velkého množství dat uložených v session. Session představuje mechanismus pro uchování stavu v HTTP protokolu. Jednou z možností jak uchování stavu dosáhnout jsou tzv. cookie, které jsou předávány v HTTP hlavičce. Ukládání velkého množství dat v session může nastat např., když uživatel může nějakým způsobem data v session ovlivňovat. Problém může být o to horší, když webová aplikace umožňuje podobnou operaci všem uživatelům (i nepřihlášených).

Testování není opět snadný úkol. Dobrou vstupní branou pro testování jsou webové stránky, které zobrazují velké množství záznamů. Test je vhodné opět provést pomocí automatizovaného skriptu, který bude vytvářet velké množství session. V případě testování je vhodné opět poskytnout SNMP data pro monitorování aktivit serveru.

6.3 Ohodnocení rizika

Nalezení zranitelnosti je důležité, ovšem tím penetrační test nekončí. Je třeba každou zranitelnost nějakým způsobem ohodnotit, tím se zabývá hodnocení rizik. Určí váhu známých zranitelností a jejich technických a obchodních dopadů.

Ohodnocení rizik se provádí standardním modelem

$$Riziko = pravděpodobnost * dopad,$$

kde pravděpodobnost označuje jak „velká“ je možnost, zda bude daná zranitelnost zneužita a dopad označuje jak „velké“ důsledky ponese zneužití dané zranitelnosti.

Postup hodnocení rizik:

1) **Nalezení rizika**
2) **Ohodnocení faktorů pro odhad pravděpodobnosti**
3) **Ohodnocení faktorů pro odhad obchodních a technických dopadů**
4) **Určení závažnosti rizik**
5) **Rozhodnutí o opravení nalezených zranitelností**

Každý faktor pro odhad pravděpodobnosti se ohodnotí na stupnici 0 až 9. Následující tabulka 6-1 zobrazuje hodnocené faktory s příkladem ohodnocení. Detailní popis, jak hodnotit faktory, je popsán v [24]. Tabulka 6-2 zobrazuje způsob ohodnocení výsledné pravděpodobnosti a dopadu.

Threat agent factors				Vulnerability factors			
Skill level	Motive	Opportunity	Size	Ease of discovery	Ease of exploit	Awareness	Intrusion detection
5	2	7	1	3	6	9	2
Overall likelihood=4.375 (MEDIUM)							

Tabulka 6-1: Příklad ohodnocení faktorů pravděpodobnosti (Zdroj: [24])

Likelihood and Impact Levels	
0 to <3	HIGH
3 to <6	MEDIUM
6 to 9	LOW

Tabulka 6-2: Výsledné ohodnocení slovní hodnotou (Zdroj: [24])

Faktory pro ohodnocení dopadu se opět hodnotí pomocí stupnice 0 až 9. Následující tabulka 6-3 zobrazuje používané faktory dle OWASP Testing Guide v3 s příklady jejich ohodnocení.

Technical Impact				Business Impact			
Loss of confidentiality	Loss of integrity	Loss of availability	Loss of accountability	Financial damage	Reputation damage	Non-compliance	Privacy violation
9	7	5	8	1	2	1	5
Overall technical impact=7.25 (HIGH)				Overall business impact=2.25 (LOW)			

Tabulka 6-3: Příklad ohodnocení faktorů dopadu (Zdroj: [24])

Hodnota výsledného rizika, jak technického tak obchodního, se hodnotí dle tabulky 6-4.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Tabulka 6-4: Výsledné ohodnocení rizika (Zdroj: [24])

Nyní, když jsou nalezeny všechny zranitelnosti a jsou ohodnoceny, je možné sestavit seznam podle jejich závažnosti a doporučit způsob řešení.

6.4 Tvorba dokumentace

Důležitou součástí penetračního testu je detailní dokumentace. Dokumentace musí všem osobám srozumitelně podat výsledky testů. Od vedoucího pracovníka, který nemá hluboké technické znalosti až po technický tým, který bude pracovat na nápravě nalezených zranitelností.

Proto i OWASP Testing Guide v3 předepisuje osnovu zprávy o penetračním testu. Technická zpráva o výsledku penetračního testu by měla mít čtyři části.

První část se nazývá Souhrn (**Executive summary**). V této části se nepoužívají příliš technické výrazy. Má podat globální pohled na výsledek testu vlastníkovi testovaného systému, možné hrozby a jejich následky. Ideální je používání grafických prostředků.

Následuje část s názvem Technický přehled (**Technical management overview**). Ten je určen vedoucím pracovníkům technických týmů. Vyskytují se zde větší detaily než v samotném souhrnu, rozsah testů, jejich omezení atd. Je zde uveden způsob hodnocení rizik, který je použit v následující části a souhrn nalezených problémů.

Předposlední kapitola se nazývá Hodnocení nálezu (**Assessment findings**) a podává detailní technický popis nalezených zranitelností a postup jejich odstranění. Tabulka 6-5 zobrazuje souhrn, který by mohl být součástí této kapitoly. Srozumitelně a strukturovaně podává výsledky testů pro technické pracovníky.

Category	Ref. Number	Test Name	Finding	Solution	Risk
Denial of Service Testing	OWASP-DS-001	Testing for SQL Wildcard Attacks			
	OWASP-DS-002	Locking Customer Accounts			
	OWASP-DS-003	Testing for DoS Buffer Overflows			
	OWASP-DS-004	User Specified Object Allocation			
	OWASP-DS-005	User Input as a Loop Counter			
	OWASP-DS-006	Writing User Provided Data to Disk			
	OWASP-DS-007	Failure to Release Resources			
	OWASP-DS-008	Storing too Much Data in Session			

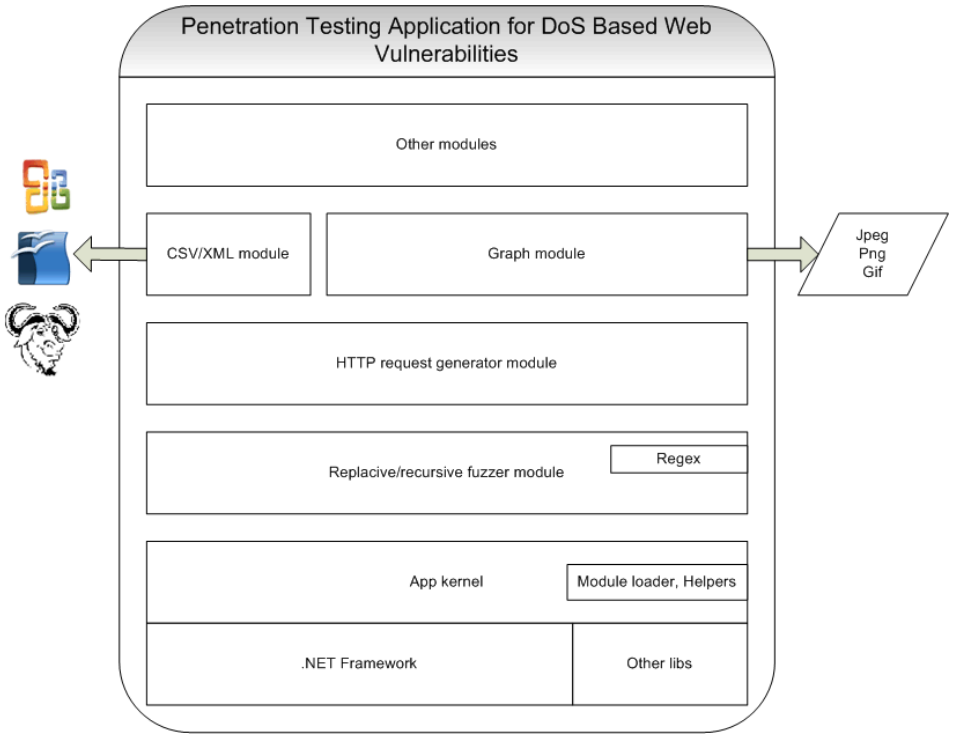
Tabulka 6-5: DoS testy dle OWASP Testing Guide v3 (Zdroj: [24])

Poslední kapitola se nazývá Použité nástroje (Toolbox). Zde jsou uvedeny nástroje, které byly použity během testu. Může se jednat o open source nástroje, komerční nebo o použité skripty atd.

7 Návrh vlastní aplikace

Vlastní aplikace musí umožňovat automatizovaný audit zranitelností typu DoS ve webových aplikacích. Ovládání musí být realizováno pomocí přehledného uživatelského rozhraní. Jako platforma pro běh aplikace je vyžadován Microsoft Windows. Není implicitně definován požadavek na použité vývojové prostředí a jazyk.

Aplikace bude spojovat funkčnost HTTP generátoru požadavků a fuzzeru a to jak rekurzivního, tak nahrazovacího. Dále by aplikace měla umožňovat grafický a textový výstup přijatých HTTP odpovědí. Diagram na obr. 7-1 zobrazuje blokovou strukturu navržené aplikace. Navržená aplikace je modulární, tzn., že bude rozšiřitelná o libovolný modul, nejen o moduly znázorněné v diagramu a zároveň stávající moduly mohou být nahrazeny.



Obrázek 7-1: Blokový diagram návrhu aplikace

Jádro aplikace poskytuje základní funkčnost pro běh celé aplikace, jako jsou prostředky pro práci s vlákny, podporu pro multijazykové prostředí, logování událostí atd. Základním kamenem bude zavaděč modulů aplikace.

Fuzzer, jak byl popsán v předcházejících kapitolách je jedním z hlavích modulů, který umožní testování různých kombinací vstupů webových aplikací. Fuzzer by měl podporovat zadávání hodnot pomocí souborů a regulárních výrazů, což bude pravděpodobně znamenat navržení jednoduchého jazyka, který bude umět generovat požadované výrazy a předávat je generátoru HTTP požadavků.

Aplikace pro penetrační testování webových zranitelností typu DoS předpokládá kvalitní HTTP generátor, viz kapitola 5.3.1. Tento HTTP generátor musí umět použít nejen protokol HTTP, ale i protokol HTTPS. Důležitým faktorem je také možnost přístupu na webové aplikace, které vyžadují autentizaci. Předpokládají se autentizační mechanismy basic, digest a NTML. Generátor HTTP požadavků spravuje v textové podobě informace o příchozích HTTP odpovědích, jako jsou doba odezvy, velikost HTTP odpovědi atd. HTTP generátor by také mohl umožňovat export přijatých HTTP odpovědí do externích formátů. Jako vhodné se mohou jevit formáty CSV a XML. Exportovaná data mohou být dále zpracována dle konkrétních požadavků.

Pro rychlý grafický náhled slouží modul pro vykreslování grafů, který by mohl vykreslovat charakteristiky časové odezvy, velikosti HTTP odpovědí atd. Umožněn by měl být export do grafického formátu. Jako vhodné se mohou jevit formáty PNG, JPEG nebo GIF.

Aplikace je navržena jako modulární tzn., že bude poskytovat API pro tvorbu dalších nebo náhradu existujících modulů.

8 Implementace aplikace

Kapitola detailně popisuje implementaci navržené aplikace. Popisuje jednotlivé projekty celého řešení v prostředí Microsoft Visual Studio 2010. Jsou popsány nejen implementační detaily, ale také se kapitola dále zabývá uživatelským rozhraním aplikace.

8.1 Vývojové prostředí

Hlavním požadavkem byl běh na platformě Microsoft Windows. Z tohoto důvodu byla zvolena pro vývoj platforma Microsoft .NET Framework, která je na platformě Microsoft Windows velice populární. Výhodou je možnost psaní modulů v libovolném jazyce ze skupiny .NET jazyků a v případě dodržení jistých pravidel bude možné aplikaci přeložit díky projektu Mono na unixových systémech. Dále může být použito velké množství knihoven vytvořených pro .NET Framework. K vývoji bylo použito vývojové prostředí Visual Studio 2010. Jedná se o nejnovější IDE od společnosti Microsoft. V korporátní sféře by pravděpodobně nebylo možné používat nejnovější nástroje, které ještě nejsou dostatečně prověřené. Já jsem využil akademického prostředí, abych mohl využít nejnovější Microsoft .NET Framework 4.0, což se mi při vývoji vrátilo v podobě komponent, které byly na rozdíl od starších verzí kvalitnější jak přístupem, tak funkčností.

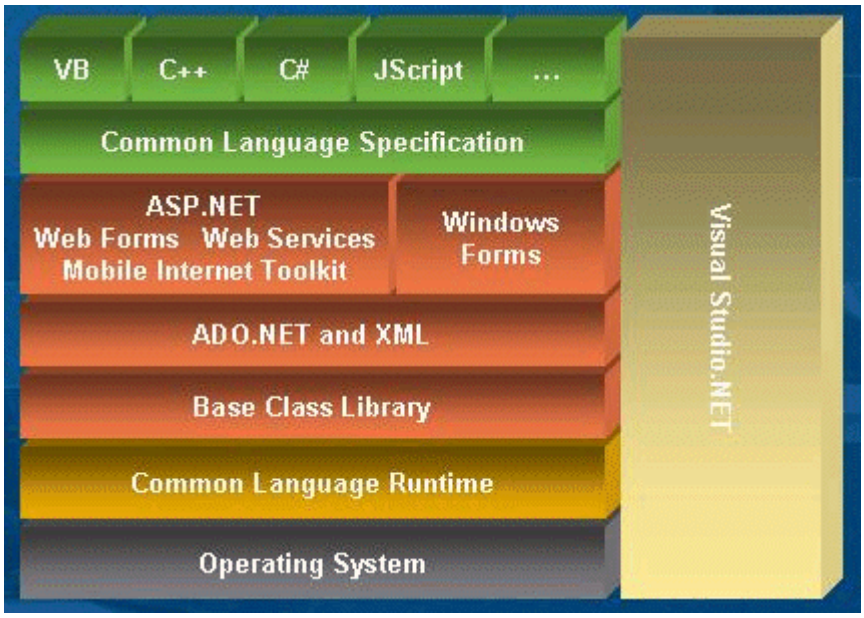
Struktura Microsoft .NET Framework je znázorněna na obrázku 8-1. Aplikace vytvořené pomocí Microsoft .NET Framework po spuštění běží ve virtuálním stroji, takže jsou teoreticky nezávislé na hardware a také na operačním systému. Nezávislost na operačním systému potvrzují i implementace pro další platformy jakou jsou Mac OSX nebo GNU/Linux v rámci projektu MONO.

Vrstva nazvaná Common Language Runtime (CLR) na obrázku 8-1 plní funkci virtuálního stroje. CLR za vývojáře software řeší správu paměti, některá bezpečnostní hlediska vývoje software atd. Správu paměti vykonává komponenta Garbage Collector (GC), která vykazuje nedeterministické chování, tzn. když „uváží“, že aplikace spotřebovala velké množství paměti, projde alokované zdroje a uvolní ty, které již nebudou v budoucnu použity [25]. Tudíž se vývojář nemusí soustředit na nízko úrovňová hlediska programování a může se soustředit na řešení problémů v rámci vývoje.

Další vrstva, která poskytuje funkce operačního systému, vlastní funkce a zapouzdřuje do podoby, která je standardní pro .NET Framework se nazývá Base Class Library (BCL). Vrstva BCL zapouzdřuje funkce operačního systému takovým způsobem, že vývojář software nemusí mít znalosti o použitém operačním systému.

Následují další vrstvy, které slouží pro manipulaci s daty, tvorbu uživatelských rozhraní, webových služeb, atd. Výhodou Microsoft .NET Frameworku je, že v jednom prostředí lze vytvářet aplikace pro velkou škálu aplikací, od mobilních aplikací až pro cloudové aplikace nasazené na serveru. S tím souvisí existence několika verzí Microsoft .NET Frameworku. Jedná se například o

Microsoft .NET Framework Compact určený pro mobilní aplikace, Microsoft Framework Client určený pro systémy, na kterých běží pouze klientské aplikace, Microsoft Framework Full určený na servery, které obsahují kompletní funkčnost potřebnou na serverech.



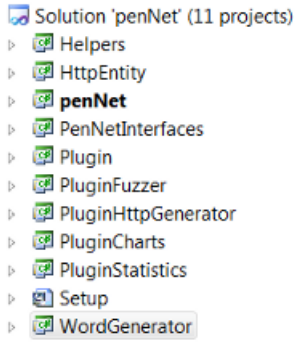
Obrázek 8-1: Microsoft .NET Framework [HTTP://www.microsoft.cz]

8.2 Struktura implementované aplikace

Aplikace byla implementována v Microsoft .NET Frameworku 4 Client ve Visual Studiu 2010. Celé Solution[12] se skládá z jedenácti dílčích projektů, které představují jádro aplikace, moduly a vlastní knihovny. Solution aplikace je pojmenováno penNet a z toho vyplývá i jméno výsledné aplikace. Na obrázku 8-2 je znázorněno uspořádání řešené aplikace.

Hlavním projektem je penNet, který se stará o lokalizaci a načítání jednotlivých modulů. Projekt Helpers je sada tříd, které poskytují funkčnost, která je často v celé aplikaci opakovaně použita. Plugin a PenNetInterface jsou knihovny pro práci s obecnými moduly, respektive se speciálními moduly aplikace penNet. Projekty PluginFuzzer, PluginHTTPGenerator, PluginCharts a PluginStatistics jsou jednotlivé moduly, kterými aplikace penNet disponuje. Projekt HTTPEntity poskytuje třídy pro práci s HTTP požadavky a HTTP odpověďmi. Poslední projekt WordGenerator slouží pro generování řetězců pro modul PluginFuzzer.

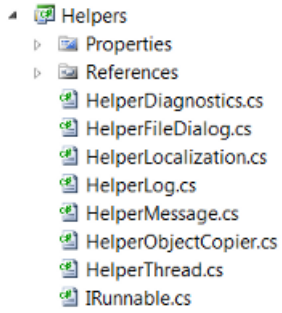
[12] Solution (Řešení) v MS Visual Studiu se nazývá implementace celé jedné aplikace. Solution se dále skládá z dílčích projektů.



Obrázek 8-2: Uspořádání solution implemetované aplikace

8.2.1 Projekt Helpers

Celá aplikace využívá podpůrné knihovny Helpers, do které jsem zahrnul klasicky nejvíce opakované operace. Všechny typy pomocníků (angl. Helpers) jsou znázorněny na obrázku 8-3. Obecně se jedná o třídy se statickými metodami, které lze volat v celém Solution.



Obrázek 8-3: Helpers

- HelperDiagnostics.cs – Vrací stav systému, tj. vytížení CPU, paměti, atd., tato třída umožňuje sledovat vytížení systému při penetračním testování.
- HelperFileDialog.cs – Zobrazuje dialogová okna pro otevírání a ukládání souborů. Tento Helper je volán vždy, když uživatel aplikace požaduje operaci ukládání nebo otevírání souboru. Pouze jeden typ dialogových oken na globální úrovni umožňuje v celém projektu jednotnou práci a vzhled.
- HelperLocalization.cs – Získává a nastavuje nastavení systému z hlediska lokalizace. Tato třída umožňuje získat současné jazykové nastavení systému a podle toho říci aplikace pentNet, kterou jazykovou mutaci má použít.
- HelperLog.cs – Ukládá logovací záznamy různých priorit na zvolený výstup. Při ladění programu je důležité zaznamenávat prováděné akce pro případnou reprodukci chybového stavu, tato třída umožňuje zaznamenávat události šesti úrovní a programátor může zvolit cíl, např. soubor.

- HelperMessages.cs – Zobrazuje oznamovací okna různých priorit uživateli. Aplikace penNet zobrazuje PopUp okna pouze typů, které jsou definovány v této třídě. Výhodou je jednotný vzhled a manipulace s nimi. Jsou podporovány PopUp okna Error (chybové hlášení uživateli) a Info (informace pro uživatele).
- HelperObjectCopier.cs – Vytváří mělkou kopii souboru. Mělká kopie objektu je kopie, kde se hodnoty nereferenčních datových typů zkopírují do kopie objektu a proměnné, které vytváří reference na další objekty, stále ukazují na tytéž objekty. Nevytváří se tedy kopie referencovaných objektů. Pokud tedy v originálním objektu nebo kopii změníme hodnotu v objektu, na který oba ukazují tak se změna promítne v originálu i kopii stejně.
- HelperThread.cs – Vytváří vlákna. Aplikace penNet umožňuje mutlthreading[13]. Posílání HTTP požadavků ve více vláknech současně a to je realizováno touto třídou.
- IRunnable.cs – Rozhraní třídy, která bude spuštěna jako vlákno.

Ukázka kódu Zdrojový kód 8-1 zobrazuje část třídy pro manipulaci s vlákny s využitím rozhraní IRunnable. Ukázka kódu představuje použití rozhraní, které implementuje metodu Start, což umožňuje jednotnou práci se všemi objekty, jejichž metody budou vykonávány v samostatném vlákně. V celé aplikaci penNet jsou ošetřovány výjimky, což vede ke stabilitě celé aplikace.

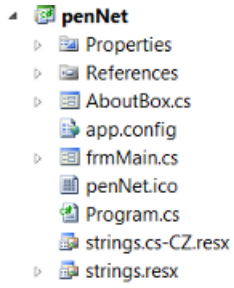
[13] Multithreading označuje podporu více současně běžících vláken v aplikaci. Vlákno je odlehčený proces, který nevyžaduje tolik režie jako klasický proces při přepínání kontextu.

```
    /// <summary>
    /// Class for a thread manipulation
    /// </summary>
    public static class HelperThread
    {
        /// <summary>
        /// This method returns new a thread
        /// </summary>
        /// <param name="threadClass">IRunnable class</param>
        public static Thread StartThread(IRunnable threadClass)
        {
            Thread thread = null;
            try
            {
                thread = new Thread(new ThreadStart(threadClass.Start));
                thread.Start();
            }
            catch
            {
                throw;
            }

            return thread;
        } // StartThread
    }

    /// <summary>
    /// Thread interface
    /// </summary>
    public interface IRunnable
    {
        void Start();
    }
```

Zdrojový kód 8-1: HelperThread.cs, IRunnable.cs

8.2.2 Projekt penNet

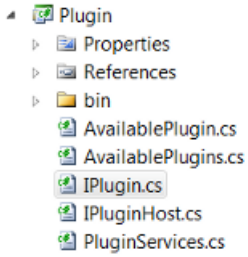
Projekt penNet je hlavní projekt a první, který se spouští. Tento projekt tvoří jádro celé aplikace. Při jeho spuštění si v kořenovém adresáři načte Dostupný moduly a zobrazí je. Dále nastaví jazykovou mutaci aplikace podle systému, na kterém běží. Provádí logování všech činností do souboru a do okna Zprávy. Projekt je napsán univerzálně, což znamená, že může být využit jako součást jiných projektů, které mají modulární charakter. Projekt také umí zobrazovat informace o autorovi a verzi aplikace. Obrázek 8-4 zachycuje uspořádání projektu. Každý modul musí obsahovat soubor strings.resx, který obsahuje anglické popisky grafických komponent. Přidáním dalších lze vytvářet další jazykové lokalizace, např. strings.cs-CZ.resx.



Obrázek 8-4: penNet

8.2.3 Projekt Plugin

Tento projekt slouží k manipulaci s moduly. Umožňuje načítat moduly, odebírat atd. Obsahuje rozhraní potřebná jak pro moduly, tak pro aplikaci, která moduly spravuje. Struktura projektu je na obrázku 8-5.



Obrázek 8-5: Plugin

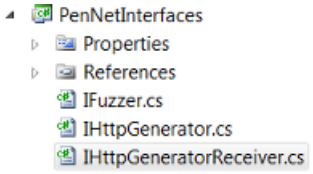
- AvailablePlugin.cs – Informace o jednom modulu.
- AvailablePlugins.cs – Informace o všech modulech.
- IPlugin.cs – Rozhraní modulu, podle něhož je modul identifikován. Zdrojový kód 8-2 zobrazuje rozhraní IPlugin s položkami, které musí každý modul obsahovat. Moduly nejsou anonymní kusy kódu, ale každý modul nese informace o autorovi, verzi, popis atd.
- IPluginHost.cs – Rozhraní aplikace, která pracuje s moduly.
- PluginService.cs – Logika práce s moduly. Vstup je aktuální umístění modulů a výstupem kolekce získaných modulů typu AvailablePlugins.

```
    /// <summary>
    /// General plugin interface
    /// </summary>
    public interface IPlugin
    {
        IPluginHost Host { get; set; }

        string PluginName { get; }
        string Description { get; }
        string Author { get; }
        string Version { get; }
        uint Order { get; }
        string LocalizePluginName { get; }
        void Localize();

        System.Windows.Forms.UserControl MainInterface { get; }

        void Initialize();
        void DisposePlugin();
    }
```

Zdrojový kód 8-2: IPlugin

8.2.4 Projekt PenNetInterface

Předchozí kapitola popisovala obecné rozhraní pro moduly. Ovšem implementovaná aplikace penNet vyžaduje identifikaci a použití některých speciálních modulů.



Obrázek 8-6: PenNetInreface

- IFuzzer.cs – Rozhraní pro fuzzer, tedy modul, který generuje posloupnosti slov nebo čísel pro testování vstupů aplikací. Ukázka zdrojového kódu na 8-3.
- IHTTPGenerator.cs – Rozhraní pro hlavní modul, bez kterého nelze aplikaci penNet spustit, generátor HTTP požadavků.
- IHTTPGeneratorReceiver.cs – Rozhraní pro moduly, které přijímají HTTP odpovědi získané HTTP generátorem.

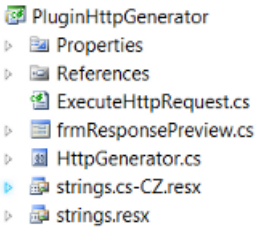
```
/// <summary>
/// Fuzzer plugin interface
/// </summary>
public interface IFuzzer
{
    string NextValue(string key);
    IFuzzer GetFactoryGenerator();
    long Count { get; }
}
```

Zdrojový kód 8-3: IFuzzer

8.2.5 Projekt PluginHTTPGenerator

Tento projekt implementuje rozhraní IHTTPGenerator. Umožňuje posílat HTTP požadavky a přijímat HTTP odpovědi s funkčností, která byla definována v kapitole Návrh. Každému modulu, který je definovaný jako IHTTPGeneratorReceiver posílá přijaté HTTP odpovědi pro další zpracování.

Modul PluginHTTPGenerator umožňuje tvorbu HTTP požadavku dvěma způsoby. První způsob je tvorba v grafickém uživatelském rozraní a druhý je přímý zápis v textové podobě. Nezávisle na způsobu implementace je následně HTTP požadavek reprezentován vnitřní strukturou viz následující kapitola 8.2.6 a může být poslán objektem HTTPWepRequst[14]. Aplikace penNet umožňuje nastavení několika parametrů pro posílání HTTP požadavků, hlavním je počet odeslaných požadavků, dále zpoždění mezi každým odesláním požadavku a také počet vláken v kterých budou HTTP požadavky paralelně odesílány. Po spuštění testu lze sledovat průběh testu a dobu jeho trvání. Každou sekundu se také zobrazují nově příchozí HTTP odpovědi v tabulce. Dále je možné v dialogovém okně každou jednotlivou HTTP odpověď celou zobrazit. Je možno ukládat a opět načítat vytvořené HTTP požadavky do textového souboru. Metadata HTTP odpovědí lze ukládat ve formátu CSV[15] nebo XML[16]. Lze využít autentizace typu Basic, Digest nebo NTML.

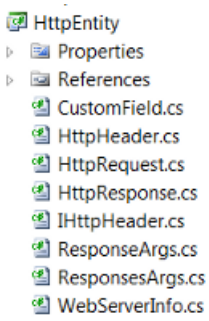


Obrázek 8-7: PluginHTTPGenerator

[14] Třída frameworku .NET pro odesílání HTTP požadavků.
[15] CSV je formát textového souboru, který data odděluje středníky a novými řádky. Lze ho zobrazit např. v aplikace MS Excel.
[16] XML je textový, strojově čitelný formát podobný HTML pro výměnu dat.

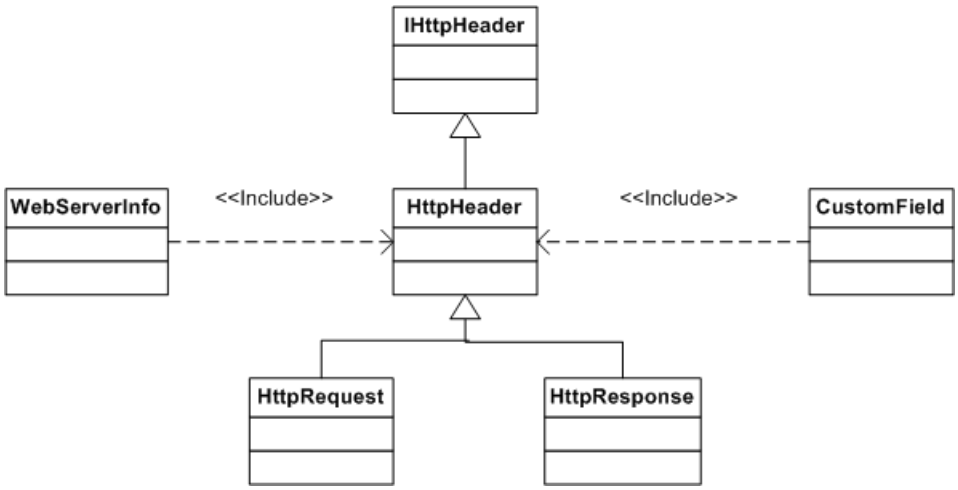
8.2.6 Projekt HTTPEntity

Projekt HTTPEntity obsahuje třídy pro manipulaci s HTTP požadavky a odpověďmi. Těmito třídami jsou vnitřně reprezentovány HTTP požadavky a odpovědi.



Obrázek 8-8: HTTPEntity

Obrázek 8-9 znázorňuje závislosti mezi třídami. Rozhraní IHTTPHeader definuje položky, které obsahuje každý HTTP požadavek/odpověď. Třída HTTPHeader proto implementuje toto rozhraní a přidává metody pro manipulaci s danými daty a dále obsahuje třídy WebServerInfo pro přidané informace o serveru (příjemci nebo odesílateli) a CustomField, která slouží pro uchování uživatelsky definovaných položek.

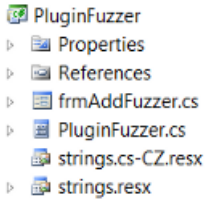


Obrázek 8-9: UML diagram tříd projektu HTTPEntity

Třídy ResponseArgs a ResponsesArgs slouží pro předávání odpovědi respektive skupině odpovědí pomocí události, která oznamuje, že jsou k dispozici nové HTTP odpovědi.

8.2.7 Projekt PluginFuzzer

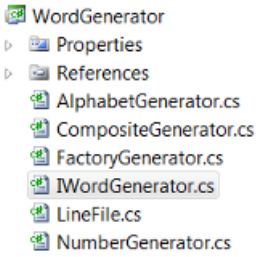
Tento modul implementuje rozhraní IFuzzer a slouží jako zdroj dat HTTP generátoru požadavků. Uživateli slouží jako prostředek definice dat, která budou nahrazována v HTTP požadavku. Lze definovat hodnoty formulářů, cookie, atd. Jako zdroj dat lze využít soubor, ve kterém je na každém řádku požadovaná hodnota nebo lze využít jednoduchého jazyka, který umožňuje definovat řetězce, které mají být generované. Princip použití vytvořeného jazyka je popsán v Příloze A a v následující kapitole. Modul fuzzer využívá knihovny WordGenerator, která tvoří celou logiku tohoto modulu.



Obrázek 8-10: PluginFuzzer

8.2.8 Projekt WordGenerator

Generování řetězců je klíčovou komponentou při testování webových aplikací a právě tím se zabývá projekt WordGenerator.



Obrázek 8-11: WordGenerator

Projekt WordGenerator definuje tři hlavní generátory. Jedná se o generátor číselné posloupnosti, generátor řetězců (variace s opakováním) a generátor, který pouze čte řádky ze souboru. Všechny tyto třídy implementují rozhraní IWordGenerator. Třída CompositGenerator pomocí tří základních tříd vytváří složené komplikovanější generátory. Třída FactoryGenerator je „správcem“ všech používaných generátorů, jak jednoduchých, tak složených. FactoryGenerator slouží jako zdroj hodnot pro modul PluginFuzzer. Následující zdrojový kód 8-4 předvádí rozhraní generátoru.

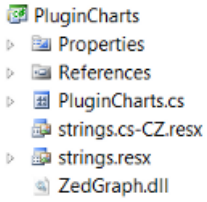
```
public interface IWordGenerator
{
    long Count { get; }
    string NextValue();
}
```

Zdrojový kód 8-4: IWordGenerator

Pro zápis generátorů je použit jednoduchý jazyk, který jsem pro daný účel implementoval. Generování číselných řad se zapisuje pomocí výrazu (d:minimun, maximu, krok), kde např. (d:0,9,1) generuje číselnou řadu 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Dále je možno generovat variace s opakováním pomocí zápisu (w:abeceda)délka_řetězce, např. (w:abc)2 bude generovat aa, ab, ac, ba, bb, bc, ca, cb, cc. Lze zadat i konstantní řetězce, bude vrácen opět tento řetězec. Důležité je, že tyto generátory lze skládat pomocí operátoru „+“ v komplikovanější generátory, např. x+(w:ab)2+(d:0,1,1) bude generovat řetězce xaa0, xaa1, xab0, xab1, xba0, xba1, xbb0, xbb1.

8.2.9 Projekt PluginCharts

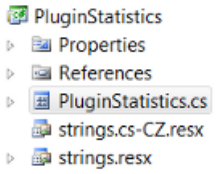
Modul PluginCharts implementuje rozhraní IHTTPGeneratorReciver, což znamená, že přijímá data od HTTP generátoru. Tyto data vynáší do grafické podoby pomocí open source knihovny ZedGraph. Jedná se o časovou odezvu, velikost a status HTTP odpovědi. Tyto grafy je dále možné ukládat do souboru v grafických formátech JPEG (Joint Photographic Experts Group), PNG (Portable Network Graphics) a GIF (Graphics Interchange Format). S grafy je možné manipulovat. Točením kolečka na myši je možné graf přibližovat nebo oddalovat. Držením prostředního tlačítka a pohybem myši je možno grafem posouvat.



Obrázek 8-12: PluginCharts

8.2.10 Projekt PluginStatistics

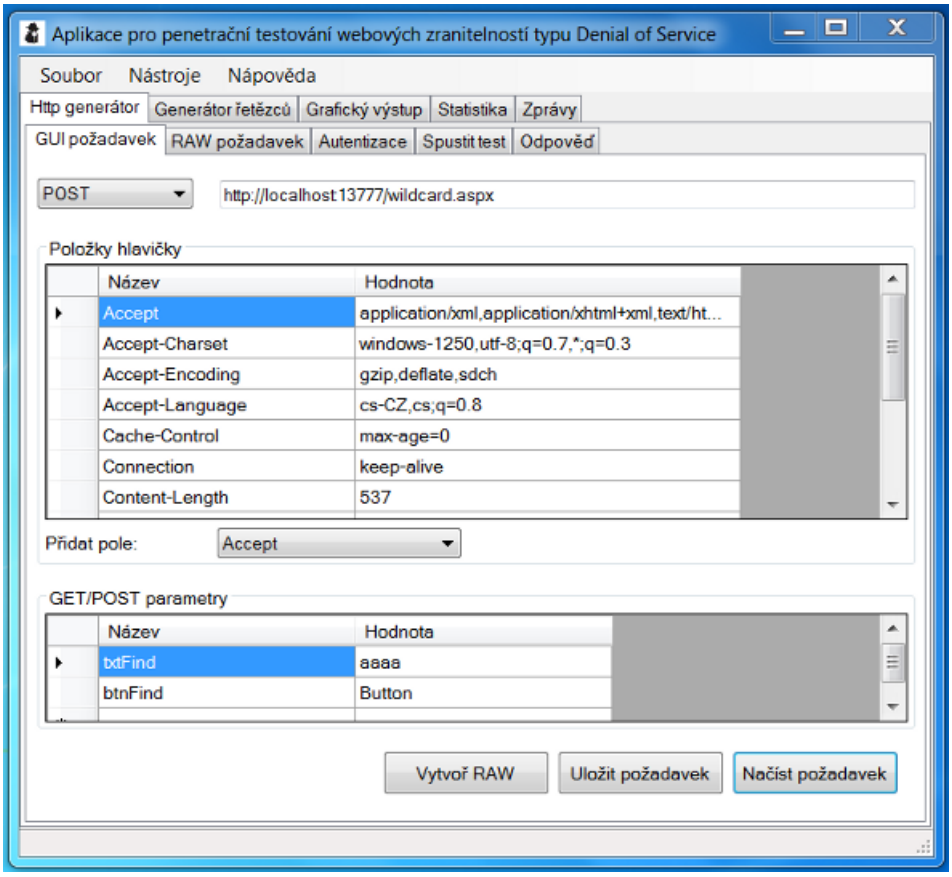
Modul PluginStatistics také implementuje rozhraní IHTTPGeneratorReciver. Jedná se o jednoduchou ukázku modulu. Modul zobrazuje průměrnou, minimální a maximální hodnotu doby odezvy a velikosti HTTP odpovědí přijatých HTTP generátorem.



Obrázek 8-13: PluginStatistics

8.3 Uživatelské prostředí aplikace

Tato kapitola představí uživatelské rozhraní implementované aplikace. Bližší seznámení s ovládáním aplikace provádí Příloha A. Následující obrázek 8-14 zobrazuje aplikaci po spuštění a vytvoření HTTP požadavku. Na obrázku jsou načteny moduly HTTP generátor, Generátor řetězců, Grafický výstup a Statistika HTTP odpovědí. Záložka Zprávy je součástí hlavní aplikace, slouží pro zaznamenávání událostí.

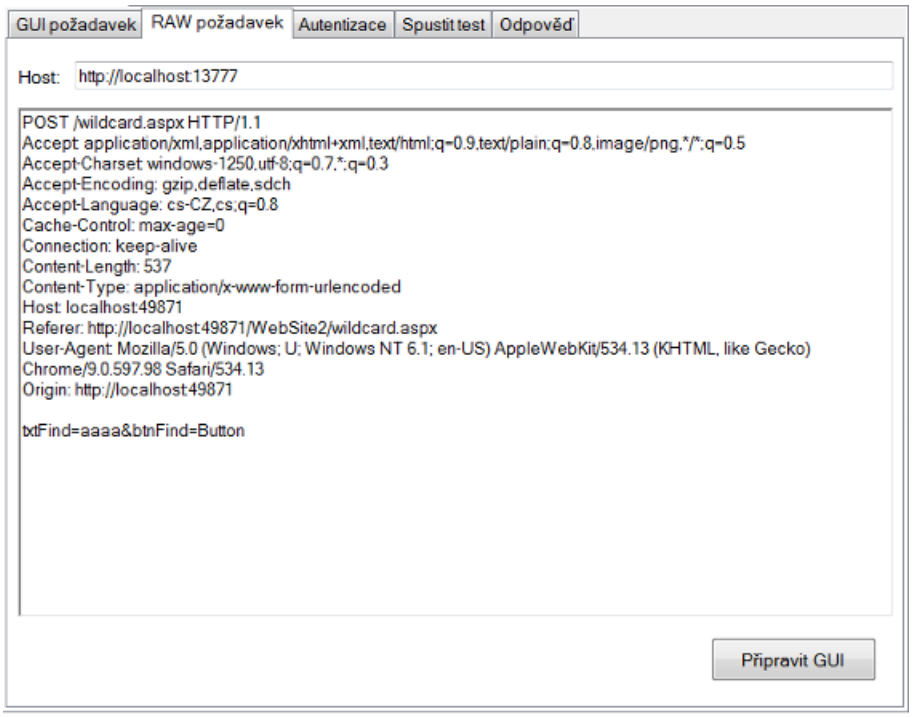


Obrázek 8-14: Aplikace po spuštění a vytvořeni HTTP dotazu

8.3.1 Modul HTTP generátor

Modul HTTP generátor je první modul, který uživatel vidí po spuštění aplikace, viz obrázek 8-14. Modul umožňuje tvorbu HTTP požadavku v grafickém prostředí, viz předchozí obrázek. Uživatel si v něm vybírá, jakou metodou bude HTTP požadavek posílán (GET/POST), následuje URL adresa s portem a cestou. Dále je okno rozděleno na dvě části. V první části (Položky hlavičky) se definuje HTTP hlavička (viz kapitola 5.1). Druhá část (GET/POST parametry) slouží pro definici hodnot formulářových prvků v případě metody POST a pro parametry v URL v případě metody GET. Výsledný HTTP požadavek je možno převést do textové podoby (RAW[17] formátu) nebo uložit do textového souboru.

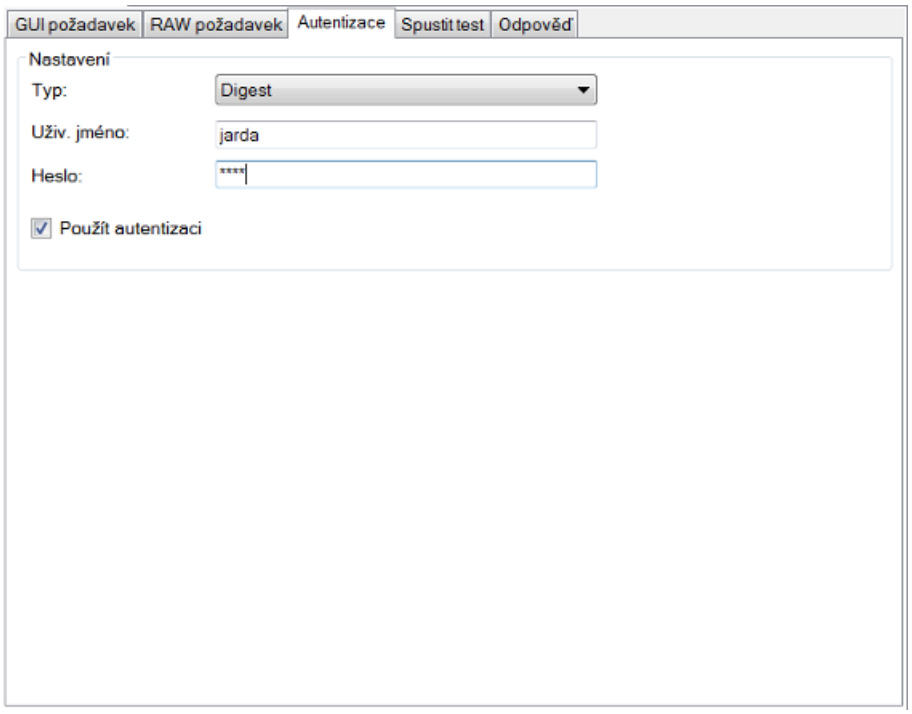
Další možností, jak vytvořit HTTP požadavek, je napsat ho přímo celý v textové podobě, k čemuž slouží záložka RAW požadavek. Vytvořený požadavek znázorňuje obrázek 8-15. Opět se zadává název serveru a port, zde se neuvádí cesta, která je uvedena za použitou metodou v HTTP požadavku. HTTP požadavky mohou využívat i šifrovaného přenosu pomocí HTTPS.



Obrázek 8-15: RAW požadavek

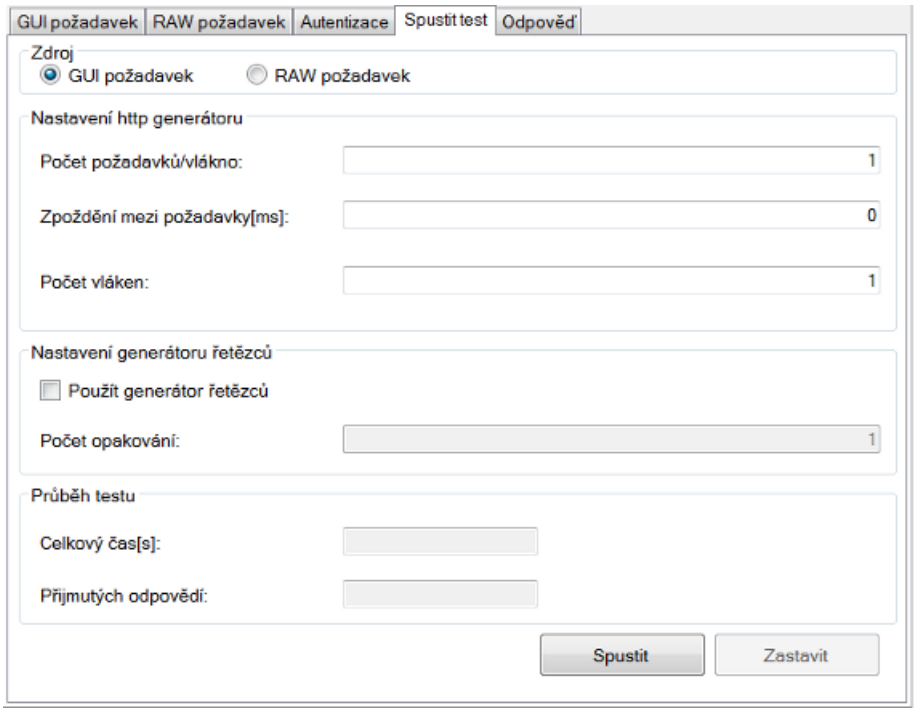
V případě, že webová aplikace využívá autentizace, poskytuje modul HTTP generátor možnost autentizace pomocí jména a hesla mechanismy Basic, Digest a NTML. Záložka autentizace je představena na obrázek 8-16.

[17] RAW z anglického slova raw, což je surový, neopracovaný. Představuje nezpracovaná data, v tomto případě v čisté textové podobě.



Obrázek 8-16: Autentizace

Nejvíce zajímavé pravděpodobně bude, jak spustit požadované testy. Nastavení testování a jeho samotné spuštění se nachází na záložce Spustit test. Zde je možno nastavit počet poslaných požadavků, počet spuštěných vláken a zpoždění mezi posíláním jednotlivých požadavků. Je možno použít Generátor řetězců (viz níže), případně zvolit jeho počet opakování. Záložka Spustit test je znázorněna na obrázku 8-17. Stav celkového průběhu testu je prezentován počtem již přijatých odpovědí a celkovou dobou trvání daného testu.

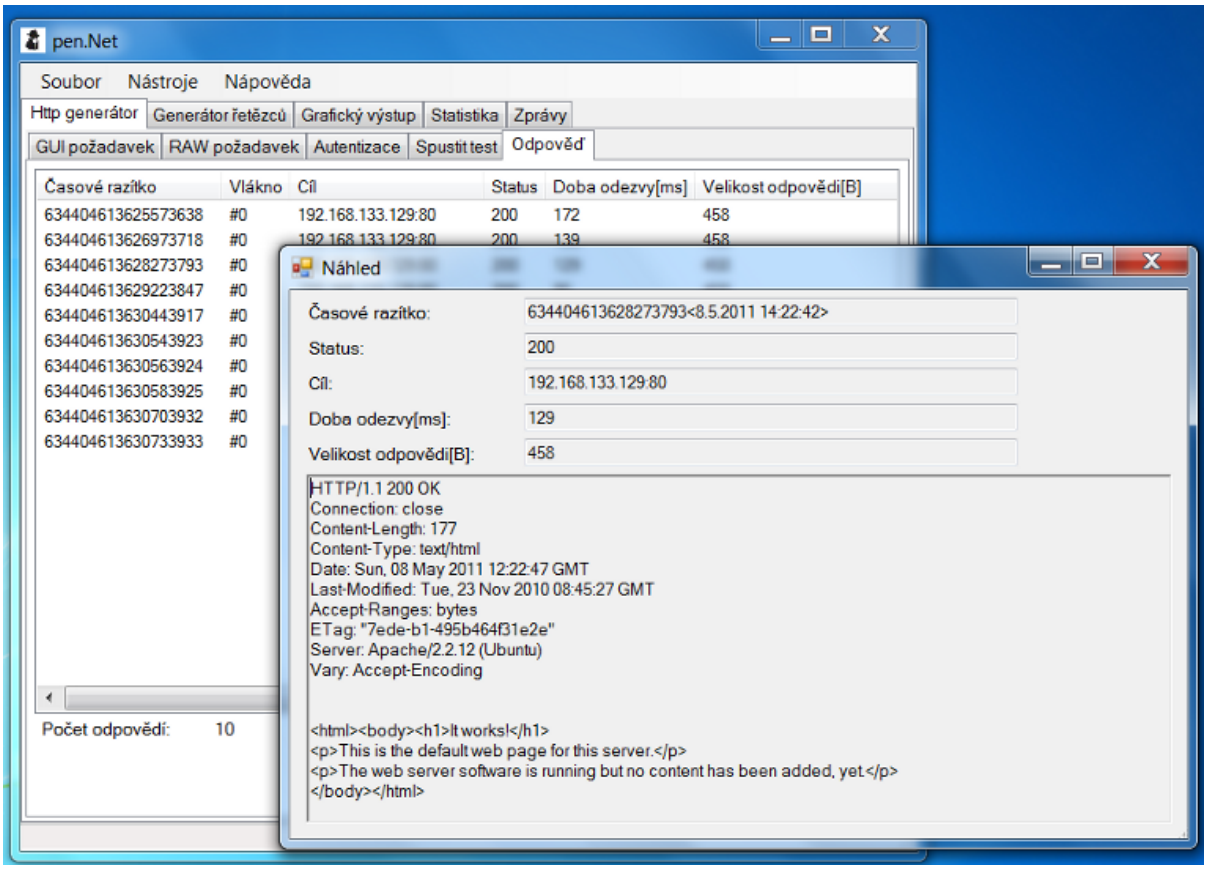


Obrázek 8-17: Spustit test

Po proběhnutí testu je zajímavá záložka Odpověď (Obrázek 8-18), která uchovává informace o přijatých HTTP odpovědích. Metadata zobrazovaná o HTTP odpovědi jsou:

- Časové razítko – Doba, kdy byl HTTP požadavek přijat s přesností na milisekundy.
- Vlákno – Číslo vlákna, které posílalo daný HTTP požadavek.
- Status – Status HTTP odpovědi (např. 200).
- Server – Název webového serveru.
- Doba odezvy – Doba potřebná pro přijetí HTTP odpovědi od odeslání HTTP požadavku.
- Velikost – Velikost dané HTTP odpovědi.

Také je možné, poklepáním na řádek HTTP odpovědi, zobrazit detailní informace o zvolené odpovědi. Tento detail zobrazuje nejen metadata, ale také samotnou HTTP odpověď.



Obrázek 8-18: Odpověď

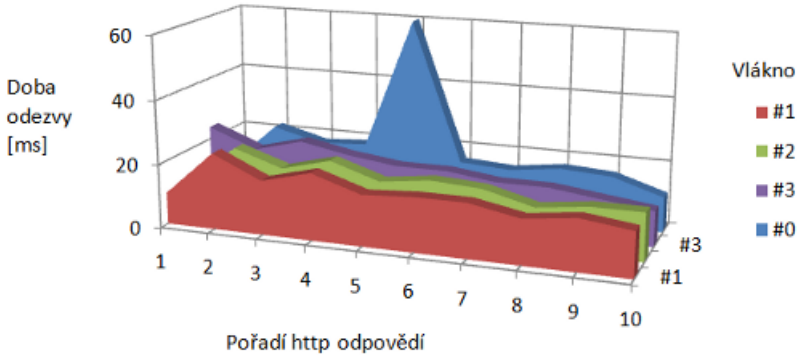
Uživatel může metadata HTTP odpovědi ukládat ve formátu CSV a metadata spolu s celou HTTP odpovědí ve formátu XML. Tyto formáty nemusí být pro člověka dobře čitelné, ale nasbíraná data je možné dále analyzovat v externích nástrojích a tvořit vlastní nejen grafické výstupy. Může se jednat například o Microsoft Excel, OpenOffice Calc nebo například skvělý nástroj GNUPlot[18]. Text 8-1 zobrazuje obsah CSV souboru exportovaného pomocí aplikace penNet.

```
timestamp;thread_id;hostname;status;response_time;response_size
634407059946855599;#0;192.168.133.134:80;200;109;458
634407059947275623;#0;192.168.133.134:80;200;14;458
634407059947335626;#0;192.168.133.134:80;200;5;458
634407059947395630;#0;192.168.133.134:80;200;6;458
634407059947445633;#0;192.168.133.134:80;200;5;458
634407059947495636;#0;192.168.133.134:80;200;5;458
634407059947555639;#0;192.168.133.134:80;200;6;458
634407059947605642;#0;192.168.133.134:80;200;5;458
634407059947675646;#0;192.168.133.134:80;200;7;458
634407059947735649;#0;192.168.133.134:80;200;6;458
```

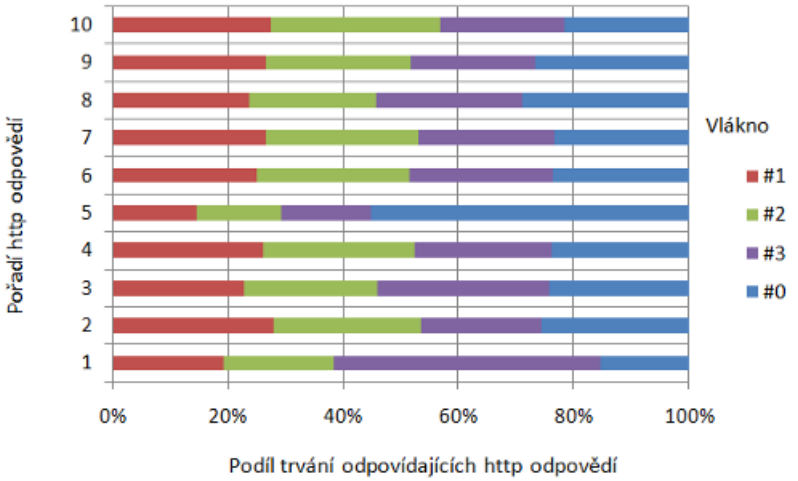
Text 8-1: penNet - CSV výstup

[18] GNUPlot je nástroj pro vykreslování grafů. Program se ovládá z příkazové řádky, proto se dá snadno použít ve skriptech atd. Je veřejný pod licencí GNU, viz [33].

Příklady zpracovaných výstupů pomocí MS Excelu jsou uvedeny Graf 8-1 a Graf 8-2, jedná se o prostorový graf průběhu jednotlivých vláken a poměr odpovídajících dob odezvy paralelních HTTP požadavků. Tyto výstupy předvádí variabilitu, kterou umožňuje export do externích aplikací.



Graf 8-1: 3D vizualizace průběhu stress testu



Graf 8-2: 2D vizualizace průběhu stress testu

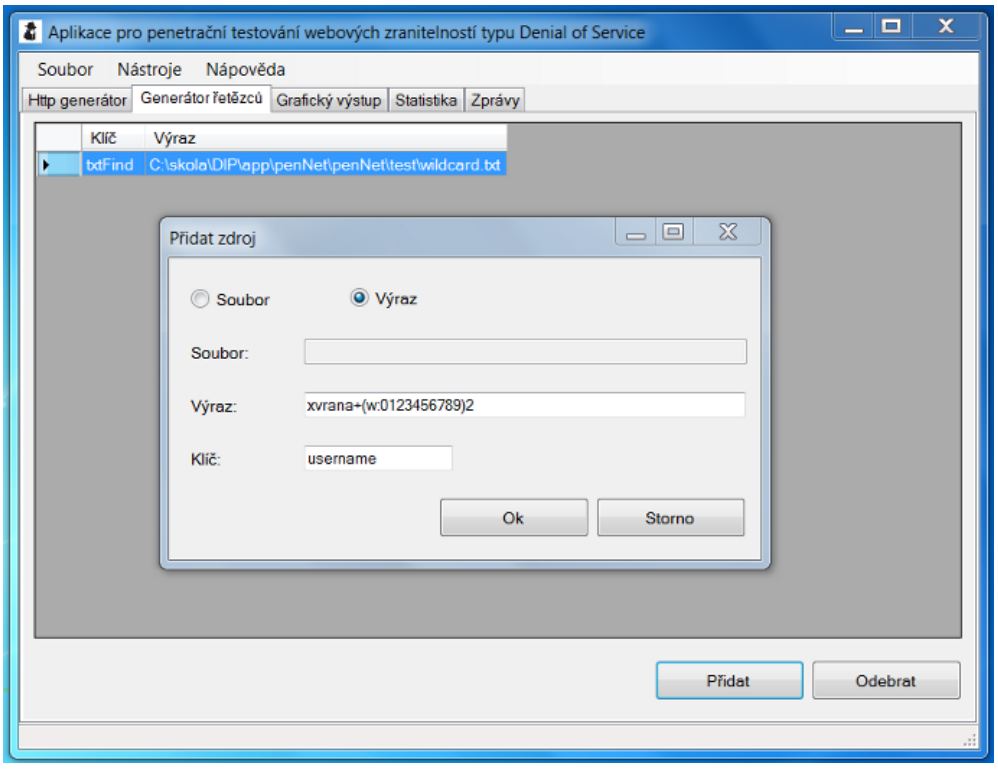
Obrázek 8-19 zobrazuje XML výstup aplikace penNet, který je zobrazen pomocí aplikace Internet Explorer 9.

```
<?xml version="1.0" encoding="UTF-8"?>
- <responses>
  - <response>
      <timestamp>634407059946855599</timestamp>
      <thread>#0</thread>
      <host>192.168.133.13480</host>
      <status>200</status>
      <time>109</time>
      <size>458</size>
      <result>HTTP/1.1 200 OK Connection: close Content-Length: 177 Content-Type: text/html Date: Wed, 11
        May 2011 08:25:10 GMT Last-Modified: Tue, 23 Nov 2010 08:45:27 GMT Accept-Ranges: bytes ETag:
        "7ede-b1-495b464f31e2e" Server: Apache/2.2.12 (Ubuntu) Vary: Accept-Encoding
        <html><body><h1>It works!</h1> <p>This is the default web page for this server.</p> <p>The
        web server software is running but no content has been added, yet.</p> </body></html> </result>
    </response>
</responses>
```

Obrázek 8-19: penNet - XML výstup

8.3.2 Modul Generátor řetězců

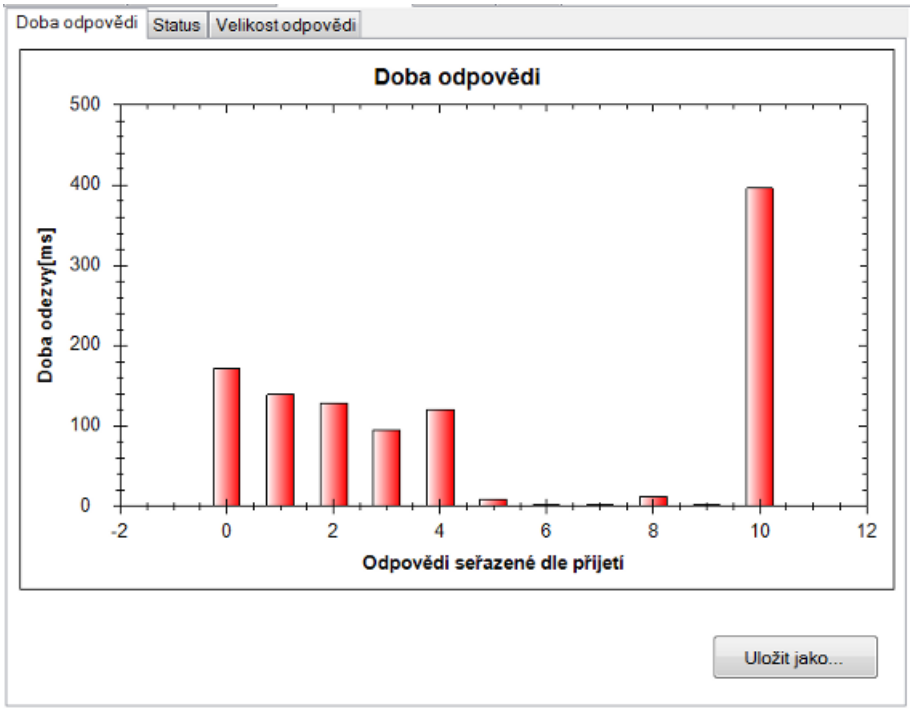
Na obrázku 8-20 je znázorněn modul, kde je definován jeden generátor jako soubor obsahující testovací hodnoty a zobrazeno dialogové okno pro tvorbu nového generátoru - v tomto případě generátoru definovaného z výrazu. Tvorba výrazů je detailně popsána v kapitole 8.2.8 nebo v Příloha A. Lze si vybrat, zda data číst ze souboru nebo z výrazu, dále se již definuje pouze za jaký klíč v HTTP požadavku se budou data dosazovat.



Obrázek 8-20: Modul Generátor řetězců

8.3.3 Modul Grafický výstup

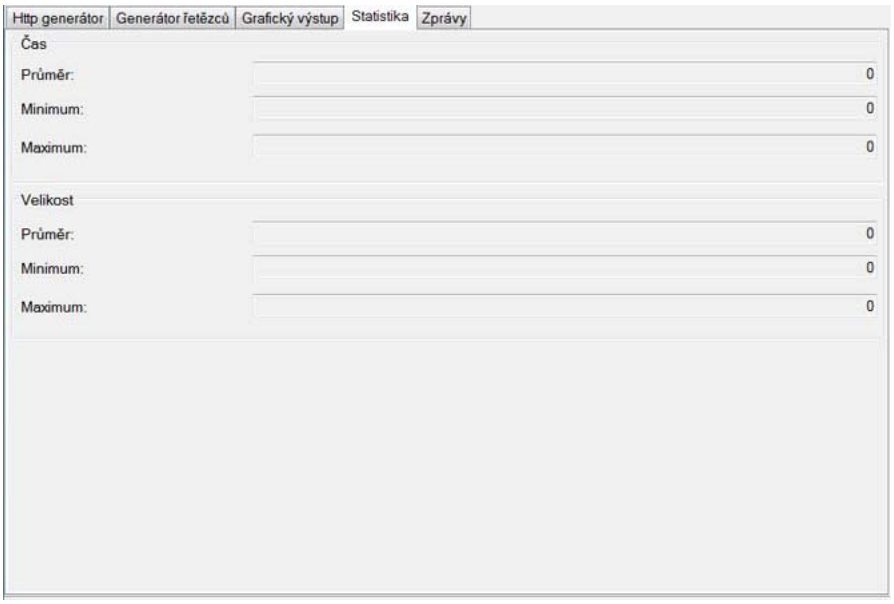
Modul grafický výstup umožňuje zobrazit statistiky přijatých odpovědí v grafické podobě a následně uložit v požadovaném grafickém formátu (JPEG, GIF a PNG). Je možné s grafy různým způsobem manipulovat (přibližovat, oddalovat a posouvat), bližší informace jsou uvedeny v kapitole 8.2.9 nebo v Příloha A. Obrázek 8-21 znázorňuje grafy Doba odezvy, Status a Velikost odpovědi. HTTP požadavky jsou řazeny dle doby příchodu.



Obrázek 8-21: Grafický výstup

8.3.4 Modul Statistika HTTP odpovědí

Modul Statistika HTTP odpovědí vznikl jako jednoduchá ukázka implementace rozhraní IHTTPGeneratorReciever. Modul přijme informace o všech HTTP odpovědích od HTTP generátoru a vytvoří statistiku o době odezvy a velikosti HTTP odpovědi. Obrázek 8-22 vykresluje modul Statistika HTTP odpovědí.



Obrázek 8-22: Statistika

9 Testování aplikace

V této kapitole jsou popsány některé funkční testy implementované aplikace (penNet). První kapitola představuje porovnání s konkurenčním nástrojem a ověřuje, zda penNet poskytuje kvalitní HTTP požadavky a následné HTTP odpovědi. Následují tři testy z množiny testů OWASP Testing Guide v3, buffer overflow, zamykání uživatelských účtů a SQL Wildcards. Nakonec je uvedena možnost využití penNetu pro stress testování spolu s ukázkou zpracování výstupu aplikace penNet v externím nástroji.

9.1 Odeslání jednoho HTTP požadavku

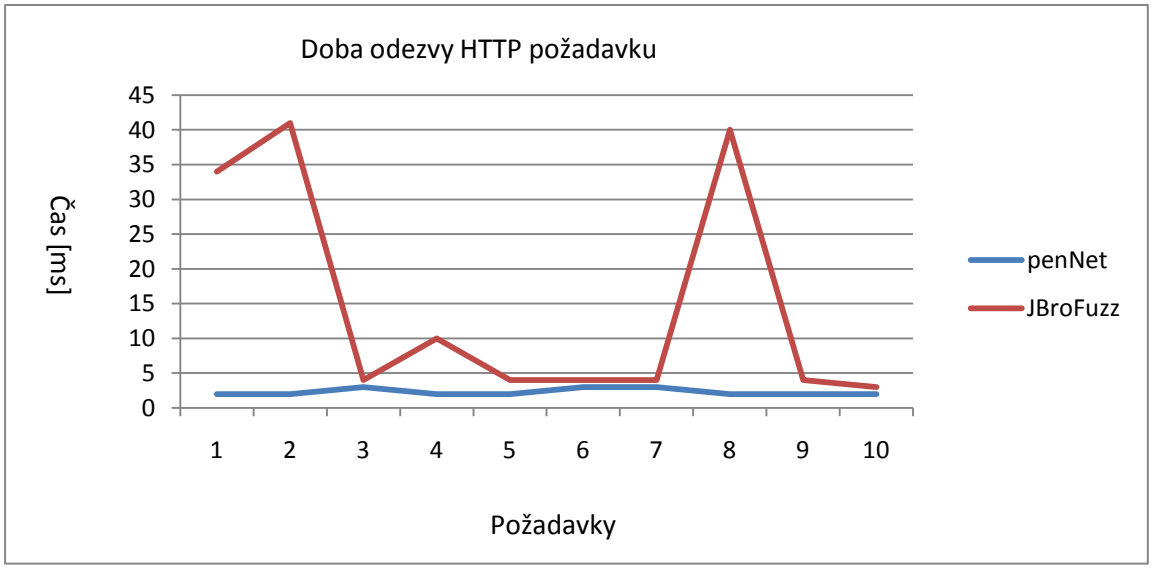
Základním prvkem aplikace je HTTP generátor (kapitola 8.3.1). Proto se první test týká právě ověření jeho správné funkčnosti. Cílem testu je odeslat HTTP požadavek webovému serveru a přijmout několik HTTP odpovědí. Totéž se provede s již existující aplikací, např. již zmíněným JBroFuzz a provést porovnání výsledku, tzn. porovnat formát a obsah HTTP odpovědi a dobu odezvy HTTP odpovědí. Test probíhal na lokální instalaci webového serveru Apache2 běžícího ve virtualizovaném operačním systému Ubuntu (konfigurace popsána v kapitole 5.2 Testovací systém).

Testovací HTTP požadavek Text 9-1 používá metodu GET a dotazuje se na objekt index.html pomocí prokolu HTTP 1.1. Následuje položka Accept, která definuje, že požadovaný objekt očekává v textové podobě ve formátu html. Položka Accept-Charset se říká, že příchozí zpráva by měla být kódována pomocí UTF8 nebo Windows 1250. Accept-Language definuje jazykové prostředí klienta. Poslední položka Host definuje IP adresu serveru, kterému je určen HTTP požadavek.

```
GET /index.html HTTP/1.1
Accept: text/html
Accept-Charset: windows-1250,utf-8;q=0.7,*;q=0.3
Accept-Language: cs-CZ,cs;q=0.8
Host: 192.168.133.180
```

Text 9-1: HTTP požadavek

Graf 9-1 zobrazuje porovnání doby odezvy HTTP požadavků aplikace penNet a JBroFuzz. Jak je z grafu patrné, tak aplikace penNet má kratší odezvy a také stabilnější průběh než aplikace JBroFuzz. Další výhodou aplikace penNet je, že umožňuje export dat. Z aplikace JBroFuzz jsem musel výsledky testu přepsat ručně do externí aplikace pro tvorbu grafu.



Graf 9-1: Porovnání doby odezvy

Následující výpisy HTTP odpovědí 9-2 (JBroFuzz) 9-3 (penNet) také dokazují, že odpovědi jsou totožné, vyjímaje uspořádání HTTP hlavičky, takže aplikace penNet přijímá a zobrazuje HTTP odpovědi správně.

```
HTTP/1.1 200 OK
Date: Sun, 24 Apr 2011 12:01:20 GMT
Server: Apache/2.2.12 (Ubuntu)
Last-Modified: Tue, 23 Nov 2010 08:45:27 GMT
Accept-Ranges: bytes
Content-Length: 177
ETag: "7ede-b1-495b464f31e2e"
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

<html><body><h1>It works!</h1>
      <p>This is the default web page for this server.</p>
      <p>The web server software is running but no content has been added, yet.</p>
</body></html>
```

Text 9-2: JBrofuzz - Detail HTTP odpovědi

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 177
Content-Type: text/html
Date: Sun, 24 Apr 2011 12:00:24 GMT
Last-Modified: Tue, 23 Nov 2010 08:45:27 GMT
Accept-Ranges: bytes
ETag: "7ede-b1-495b464f31e2e"
Server: Apache/2.2.12 (Ubuntu)
Vary: Accept-Encoding

<html><body><h1>It works!</h1>
      <p>This is the default web page for this server.</p>
      <p>The web server software is running but no content has been added, yet.</p>
</body></html>
```

Text 9-3: penNet - Detail HTTP odpovědi

Modul HTTP generátor svojí základní funkci plní správně, proto v dalších kapitolách proběhnou testy na DoS zranitelnosti uvedené v OWASP Testing Guide v3 z kapitoly 6, čímž se otestuje funkčnost dalších modulů, ale i další funkce modulu HTTP generátor.

9.2 Uzamykání uživatelských účtů (Wordpress)

Uzamykání uživatelských účtů je dle OWASP Testing Guide popsáno v kapitole 6.2.2. Na tomto testu jsem provedl test generátoru HTTP požadavků ve spojení s modulem Generátor slov. Tuto „zranitelnost" jsem testoval aplikací penNet na redakčním systému Wordpress v konfiguraci, jak jsem popsal v kapitole 5.2. Cílem tohoto testu bude provést brute force[19] útok na daný redakční systém Wordpress a zjisti, jestli se nechová rozdílně při zadávání smyšleného a skutečného uživatelského jména ve spojení se smyšleným heslem, jak udává OWASP Testing Guide v3. Pokud následně dokážeme identifikovat existující uživatelská jména, tak nebude problém je uzamknout opakovaným zadáním chybného hesla.

Nejdříve jsem použil aplikaci Fiddler2, která funguje jako proxy server[20], abych zachytil existující komunikaci internetového prohlížeče a aplikace Wordpress. Tímto jsem získal HTTP požadavek 9-4, který použiji v aplikaci penNet. Tento požadavek má parametr log, pomocí kterého webové aplikaci penNet posíláme uživatelské jméno. Právě na tento parametr jsem provedl brute force.

[19] Brute force útok je útok hrubou silou, kde se systematicky prohledává celý prostor možných řešení.
[20] Proxy server je mezičlánek mezi klientem a serverem. HTTP Požadavky a odpovědi „tečou" přes něj.

```
POST /wordpress/wp-login.php HTTP/1.1
Accept:
application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Charset: windows-1250,utf-8;q=0.7,*;q=0.3
Accept-Language: cs-CZ,cs;q=0.8
Cache-Control: max-age=0
Connection: keep-alive
Content-Length: 115
Content-Type: application/x-www-form-urlencoded
Host: 192.168.133.182
Referer: HTTP://192.168.133.182/wordpress/wp-login.php
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.16 (KHTML, like
Gecko) Chrome/10.0.648.205 Safari/534.16
Origin: HTTP://192.168.133.182

Log=bbbb&pwd=aaaaa&wp-
submit=Log+In&redirect_to=HTTP%3A%2F%2F192.168.133.182%2Fwordpress%2Fwp-
admin%2F&testcookie=1
```

Text 9-4: Ukázka HTTP požadavku na webovou aplikaci Wordpress

Použil jsem modul Generátor slov aplikace penNet. Hledal všechna uživatelská jména dlouhá pět znaků obsahující pro zjednodušení znaky jard, výraz odpovídající tomuto požadavku zobrazuje text 9-5. Výraz generuje variace s opakováním o délce pět znaků, tzn., jeden test pošle $4^5 = 1024$ HTTP požadavků. Výsledná slova jsou postupně posílána generátorem HTTP požadavků aplikace penNet webové aplikaci Wordpress se smyšleným heslem viz požadavek text 9-4. Generované řetězce postupně nahrazují hodnotu proměnné log.

```
(w:jard)5
```

Text 9-5: Výraz generující variace s opakováním (modul Generátor slov)

Následně je třeba identifikovat existující uživatelská jména. Redakční systém Wordpress tento přístup zjednodušuje, protože zobrazuje odlišná chybová hlášení pro neexistující uživatelská jména a pro existující uživatelská jména s nesprávným heslem. Výňatek z HTTP odpovědi 9-6 zobrazuje hlášení, kdy uživatelské jméno neexistuje, na rozdíl od 9-7, kdy uživatelské jméno jarda existuje. Rozdílné HTTP odpovědi lze identifikovat pomocí aplikace penNet například podle rozdílné velikost HTTP odpovědí.

```
<div id="login"><h1><a href="HTTP://wordpress.org/" title="Powered by WordPress">My test
wordpress</a></h1>
<div id="login_error"><strong>ERROR</strong>: Invalid username. <a
href="HTTP://192.168.133.182/wordpress/wp-login.php?action=lostpassword" title="Password
Lost and Found">Lost your password</a>?<br />
</div>
```

Text 9-6: Wordpress - neexistující uživatel

```
<div id="login"><h1><a href="HTTP://wordpress.org/" title="Powered by WordPress">My test
wordpress</a></h1>
<div id="login_error"><strong>ERROR</strong>: The password you entered for the username
<strong>jarda</strong> is incorrect. <a href="HTTP://192.168.133.182/wordpress/wp-
login.php?action=lostpassword" title="Password Lost and Found">Lost your password</a>?<br />
</div>
```

Text 9-7: Wordpress - existující uživatel, chybné heslo

Nyní je verifikována existence uživatelského jména jarda. Následuje poslání několika HTTP požadavků s tímto uživatelským jménem a smyšleným heslem. Tím zjistíme, zda lze uzamknout uživatelský účet. Aplikace penNet umožňuje nastavit počet opakování testů generovaných generátorem slov i nastavit opakované přeposlání jednoho požadavku, proto jsou úkony zcela automatizovány. Po třech opakovaných posláních HTTP požadavku s chybným heslem aplikace Wordpress nahlásila uzamčením účtu jarda, viz výňatek z HTTP odpovědi Text 9-8. V aplikaci penNet lze detekovat tuto událost změnou velikostí HTTP odpovědi.

```
<div id="login"><h1><a href="HTTP://wordpress.org/" title="Powered by WordPress">My test
wordpress</a></h1>
<div id="login_error"><strong>ERROR</strong>: The password you entered for the username
<strong>jarda</strong> is incorrect. <a href="HTTP://192.168.133.182/wordpress/wp-
login.php?action=lostpassword" title="Password Lost and Found">Lost your password</a>?<br />
<strong>ERROR</strong>: This user account has been locked for security reasons. Please use
Lost Password option to unlock it.<br />
</div>
```

Text 9-8: Wordpress - uzamčení uživatelského účtu

Testování pomocí dalších nástrojů zmíněných v této práci by bylo obdobné. Aplikace penNet pro tento účel nepřináší vylepšení. Ovšem ověřil jsem funkčnost modulu Generátor slov.

9.3 Buffer overflow

Zranitelnost typu buffer overflow byla popsána v kapitole 6.2.3 dle OWASP Testing Guide v3. V této kapitole se otestuji cgi skript napsaný v jazyce C. Využívám zde modul Generátor slov, který bude poskytovat testovací data, která jsou uložena v textovém souboru a modul Grafický výstup. Tímto otestuji správnou funkčnost další části modulu Generátor slov a modulu Grafický výstup. Testovací aplikace opět běžela v prostředí Ubuntu popsaném v kapitole 5.2.

Zdrojový kód 9-1 představuje jednoduchý testovací skript. Skript opět používá příkaz strpcy() na statické pole.

```
int main(void)
{
	char n[10];
	char *data;

	printf("%s%c%c\n", "Content-Type:text/html;charset=iso-8859-1",13,10);
	printf("<TITLE>Buffer ovewflow</TITLE>\n");
	printf("<H3>Buffer overflow test</H3>\n");


	data = getenv("QUERY_STRING");

	if (data == NULL) {
		printf("<p>getenv() error</p>");
		return 1;
	}

	strcpy(n, data);
	printf("<p>Buffer lenght: %d</p>", strlen(n));

	free(data);

	return 0;
}
```

Zdrojový kód 9-1: Cgi skript na straně serveru

Aplikace penNet bude postupně posílat testovací data 9-9 uložená v souboru a budeme sledovat typ HTTP odpovědí a případná chybová hlášení na straně webového serveru. Posílaná data jsou postupně delší, čímž otestujeme, kdy nastane případné přetečení. Data jsou posílána pomocí metody GET jako parametr v URL, viz HTTP požadavek 9-10. Sledování chybových logů není při praktickém testování většinou možné, ale v našich laboratorních podmínkách je to vítané.

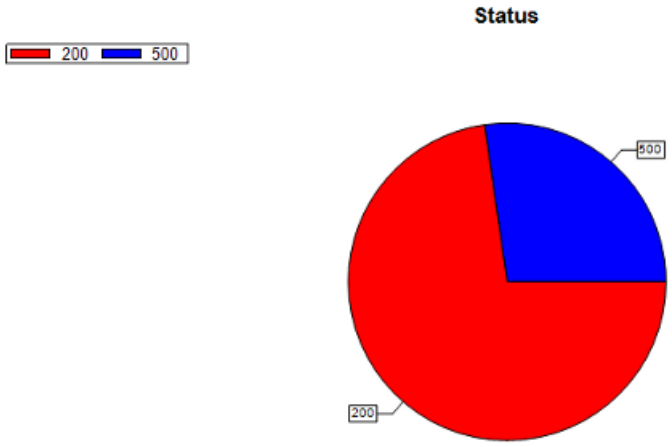
```
aaa
aaaaaa
aaaaaaaaa
aaaaaaaaaaaa
aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
… atd
```

Text 9-9: Testovací data

```
GET /cgi-bin/app.cgi?param=aa HTTP/1.1
Host: 192.168.133.134
```

Text 9-10: HTTP požadavek - bufferoverflow

Obrázek 9-1 je přímým grafickým výstupem aplikace penNet, vyplývá z něj, že více jak čtvrtina HTTP dotazů způsobila interní chybu webového serveru. Necelé tři čtvrtiny HTTP odpovědí vrátily kód statusu 200, ostatní 500, což označuje interní chybu serveru.



Obrázek 9-1: Status HTTP odpovědí při testování cgi skriptu

Nyní je zajímavé se podívat na logovací záznamy webového serveru Apache, které se nacházejí v GNU/Linux Ubuntu v adresáři /var/log/apache2/error.log. Část výpisu je zobrazena v Tabulka 9-1. Výpis skutečně zobrazuje chybnou manipulaci s pamětí, došlo k přepsání okolního paměťového místa, což je v našem případě uvolňování paměti pomocí free(), což způsobilo pokus o chybné uvolnění paměti.

```
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] *** glibc detected ***
/usr/lib/cgi-bin/mult.cgi: free(): invalid pointer: 0xbfc64e3e ***
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] ======= Backtrace: =========
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1]
/lib/tls/i686/cmov/libc.so.6[0x66a0d1]
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1]
/lib/tls/i686/cmov/libc.so.6[0x66b7d2]
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1]
/lib/tls/i686/cmov/libc.so.6(cfree+0x6d)[0x66e8ad]
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] /usr/lib/cgi-
bin/mult.cgi[0x8048600]
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] [0x61616161]
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] ======= Memory map: ========
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 00493000-004af000 r-xp 00000000
08:01 9477       /lib/libgcc_s.so.1
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 004af000-004b0000 r--p 0001b000
08:01 9477       /lib/libgcc_s.so.1
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 004b0000-004b1000 rw-p 0001c000
08:01 9477       /lib/libgcc_s.so.1
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 005ff000-0073d000 r-xp 00000000
08:01 1183       /lib/tls/i686/cmov/libc-2.10.1.so
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 0073d000-0073e000 ---p 0013e000
08:01 1183       /lib/tls/i686/cmov/libc-2.10.1.so
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 0073e000-00740000 r--p 0013e000
08:01 1183       /lib/tls/i686/cmov/libc-2.10.1.so
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 00740000-00741000 rw-p 00140000
08:01 1183       /lib/tls/i686/cmov/libc-2.10.1.so
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 00741000-00744000 rw-p 00000000
00:00 0
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 007c0000-007db000 r-xp 00000000
08:01 3556       /lib/ld-2.10.1.so
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 007db000-007dc000 r--p 0001a000
08:01 3556       /lib/ld-2.10.1.so
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 007dc000-007dd000 rw-p 0001b000
08:01 3556       /lib/ld-2.10.1.so
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 00903000-00904000 r-xp 00000000
00:00 0          [vdso]
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 08048000-08049000 r-xp 00000000
```

```
08:01 4268       /usr/lib/cgi-bin/mult.cgi
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 08049000-0804a000 r--p 00000000
08:01 4268       /usr/lib/cgi-bin/mult.cgi
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 0804a000-0804b000 rw-p 00001000
08:01 4268       /usr/lib/cgi-bin/mult.cgi
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] 08239000-0825a000 rw-p 00000000
00:00 0          [heap]
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] b78d5000-b78d6000 rw-p 00000000
00:00 0
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] b78e6000-b78e9000 rw-p 00000000
00:00 0
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] bfc50000-bfc65000 rw-p 00000000
00:00 0          [stack]
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] Premature end of script headers:
mult.cgi
[Mon Apr 25 12:39:37 2011] [error] [client 192.168.133.1] Premature end of script headers:
mult.cgi
```

Tabulka 9-1: Výpis logovacího souboru apache2/error.log

9.4 Stress test

Aplikaci penNet lze využít také jako stress test. Podobně jako aplikace URLStress nebo Apache Benchmark (ab) můžeme stálým tokem požadavků vytěžovat webový server. Aplikace penNet obsahuje konfiguraci velikosti sériové sekvence HTTP požadavků a zároveň je možné spouštět sekvence HTTP požadavků paralelně pomocí více konkurentních vláken.

Test proběhl opět na webovém serveru Apache2, jehož konfigurace je popsána v kapitole 5.2 Testovací systém. Oba programy penNet a URLStress poslaly pomocí sta vláken deset tisíc požadavků na webový server. Sledoval jsem přitom průměrnou dobu odezvy, velikost aplikací v paměti RAM a vytížení procesoru při testu. Pro diagnostiku jsem použil nástroje Správcem systému Windows a Sledování výkonu.

Obrázek 9-2 a text 9-11 zobrazují výsledky průměrné doby odezvy pro aplikaci penNet, respektive pro URLStress. Oba programy tuto hodnotu zobrazují implicitně. Aplikace penNet prostřednictvím modulu Statistika HTTP odpovědi a URLStress pomocí textového výstupu v textovém poli, které slouží jako jediný výstupní bod aplikace. Rozdíl v tomto případě je sedmnáct milisekund ve prospěch aplikace penNet, což může být způsobeno nahodilým jevem, takže jsou si aplikace v podstatě v tomto hledisku rovnocenné.

Obrázek 9-2: penNet - průměrná doba odezvy

```
      …
      Thread 67       100     0       165
      Thread 24       100     0       167
      Thread 46       100     0       167
      Thread 42       100     0       168
      Thread 17       100     0       167

      Total Average Time: 113 Milliseconds

      End Time: Wed May 11 14:13:40 CEST 2011

*** Stress Completed ***
```

Text 9-11: URLStress - průměrná doba odezvy

Aplikace penNet po spuštění zabírá v paměti o přibližně jednu třetinu méně než aplikace URLStress. Situace se obrací po doběhnutí testu, kdy aplikace penNet zabírá o cca dvacet MB více paměti. Je to dáno především tím, že penNet ukládá všechny příchozí HTTP odpovědi spolu s jejich metadaty (vlákno, doba odezvy, velikost, status). Aplikace URLStress zobrazuje pouze časové statistiky.

	penNet	URLStress
Spotřeba paměti RAM po spuštění [MB]	24	35
Spotřeba paměti RAM po zkončení testu [MB]	69	47

Tabulka 9-2: Spotřeba paměti

Podobně jako využití paměti dopadl test využití procesoru. Aplikace penNet průměrně spotřebovala o čtyři procenta více procesorového času než aplikace URLStress. Podobné také bylo minimální a maximální využití.

	penNet	URLStress
Minimální vytížení CPU [%]	41	35
Maximální vytížení CPU [%]	53	51
Průměrné vytížení CPU [%]	47	43

Tabulka 9-3: Vytížení CPU

Celkově aplikace penNet spotřebuje v průměru více systémových prostředků než aplikace URLStress. Nejedná se o razantní rozdíly. V potaz je třeba vzít také to, že aplikace penNet uchovává a zpracovává podstatně větší množství dat. Ve výsledku se tedy nejedná o ztrátu pro penNet. Doba odezvy HTTP požadavku u obou aplikací je podobná.

9.5 Porovnání zvolených aplikací

V této kapitole jsou porovnány aplikace penNet URLStress a JBroFuzz. Porovnání vybraných vlastností je zobrazeno v tabulce 9-4. Z tabulky je patrné, že implementovaná aplikace penNet poskytuje funkčnost jako konkurenční aplikace a ještě přidává např. modulární stavbu, která umožňuje tvorbu vlastních modulů nebo náhradu stávajících.

Funkce	penNet	URLStress	JBroFuzz
Paměťové nároky[21]	24 MB	35 MB	69 MB
Průměrná doba odezvy[22] [ms]	3	3	6
Fuzzer (vstupem soubor/výraz)	Ano/Ano	Ne/Ne	Ne/Ano
Integrované testy (Předdefinované fuzzery)	Ne	Ne	Ano
Grafický výstup	Ano	Ne	Ano
Export dat CSV/XML	Ano/Ano	Ne/Ne	Ne/Ne
Podpora vláken	Ano	Ano	Ne
Přímá podpora webové autentizace (Basic/ Digest/ NTML)	Ano/Ano/Ano	Ne/Ne/Ne	Ne/Ne/Ne
Metadata jednotlivých požadavků (status/velikost odpovědi/doba odezvy)	Ano/Ano/Ano	Ne/Ne/Ano (v neformátované podobě)	Ano/Ano/Ano
Statistické informace (průměrná doba odezvy HTTP požadavku)	Ano	Ano	Ne
Možnost lokalizace	Ano	Ne	Ne
Modulární stavba	Ano	Ne	Ne

Tabulka 9-4: Porovnání vlastností

[21] Velikost aplikace po spuštění zobrazena Správcem systému Windows.
[22] Posláno sto identických požadavků v jednom vlákně na referenční webový server popsaný v kapitole 5.2, test třikrát opakován.

10 Závěr

10.1 Zhodnocení výsledků práce

V úvodu práce jsem se věnoval představení základních principů počítačové bezpečnosti. Navázal jsem přiblížením pojmů DoS a DDoS a představil některé útoky na systém. Především zajímavým je útok Slowloris, na který je náchylný webový server Apache. Dále jsem se věnoval penetračnímu testování obecně a z hlediska metodologie OWASP Testing Guide v3 a to především z hlediska testů zaměřených na DoS zranitelnosti, přičemž jsem se seznámil s aplikacemi potřebnými pro tuto oblast.

Na základě získaných zkušeností jsem navrhl a implementoval aplikaci, která integruje funkce, které jsou pro danou oblast testování DoS zranitelností potřebné nebo ulehčují práci. Způsob implementace, uživatelské rozhraní a ovládání jsem popsal v práci.

Funkčnost aplikace byla ověřena porovnáním s existujícími aplikacemi na zranitelnostech, které byly představeny v OWASP Testing Guide v3. PenNet byl testován nejen na syntetických webových aplikacích, které jsem sám implementoval, ale také např. na celosvětově rozšířeném redakčním systému Wordpress. Na konci v souhrnné tabulce porovnávám aplikaci penNet s dalšími aplikacemi (URLStress a JBroFuzz). Aplikace penNet dále pokračuje ve vývoji jako open source projekt (http://pennet.sourceforge.net/).

10.2 Budoucí vývoj

Zajímavá by mohla být verze aplikace penNet pro příkazovou řádku. Aplikace penNet je implementována obecně, takže by neměl být problém konverze aplikace. Tento krok také souvisí se snahou portace aplikace v rámci projektu MONO pro běh na platformě GNU/Linux.

V návaznosti na zveřejnění projektu jako open source bude následovat implementace dalších modulů, které se ukážou jako užitečné. Bude také následovat postupná oprava zatím nenalezených chyb ve stávajících modulech.

11 Literatura

[1] *Denial-of-service attack* [online]. [cit. 14. 12. 2010]. Dostupný z WWW: <HTTP://en.wikipedia.org/wiki/Denial-of-service_attack>.

[2] GREGG, Michael. Prep, *Certified Ethical Hacker Exam.* : Que Publishing, 2006. ISBN 978-0-7897-3531-7.

[3] HANÁČEK, Petr. *Slajdy k předmětu Bezpečnost informačních systémů*. Brno, 2007.

[4] ANDREU, Andres. *Professional Pen Testing for Web Applications*.: Wrox Press, 2006. ISBN 978-0-4717-8966-6.

[5] *DDoS útoky a obrana proti nim* [online]. [cit. 14. 12. 2010]. Dostupný z WWW: <HTTP://www.itpoint.cz/zprava/?i=ddos-utoky-a-obrana-proti-nim-6402>.

[6] MELL, Peter. NIST - *Guide to IPS and IDS systems* [online]. 2007. [cit. 14. 12. 2010]. Dostupný z WWW: < HTTP://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf >.

[7] GALETKA, Ing. Josef. Diplomová práce - *Analýza síťových útoků mocí Honeypotů*. Brno, 2010.

[8] FORSBERG, Dan. *SYN Flood DoS Attack Experiments* [online]. [cit 5. 10. 2010]. Dostupný z WWW: <HTTP://www.oocities.com/zuhair_ali/Misc/synattack.html>.

[9] HUOVINEN, Lasse. *Denial of Service Attacks: Teardrop and Land* [online]. [cit 6. 10. 2010]. Dostupný z WWW: <HTTP://users.tkk.fi/lhuovine/study/hacker98/dos.html>.

[10] *Perl skript Slowloris* [Online]. [cit 10. 5. 2011]. Dostupný z WWW: <HTTP://ha.ckers.org/slowloris/slowloris.pl>.

[11] *Apache documentation* [Online]. [cit 19. 10. 2010]. Dostupný z WWW: <HTTP://HTTPd.apache.org/docs-project/>.

[12] *Specifikace webového serveru GoAhead* [Online]. [cit 10. 5. 2011]. Dostupný z WWW: <HTTP://www.goahead.com/products/webserver/specifications.aspx>.

[13] ROBBINS, Daniel. *Obrana proti Slowloris* [Online]. [cit. 9. 5. 2011] Dostupný z WWW: <HTTP://www.funtoo.org/en/security/slowloris/>.

[14] MITNICK, Kevin. *Umění klamu*. : Helion, 2003.

[15] LONG, Johnny. Google´s hacker guide v1.0. [cit. 12. 1. 2011].

[16] W3. *HTTP fields definitiv* [online]. [cit. 7. 5. 2011]. Dostupný z WWW: <*HTTP://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html* HTTP://www.w3.org/>.

[17] *Popis Ubuntu* [online]. [cit. 9. 11. 2011]. Dostupný z WWW: <HTTP://www.ubuntu.com/ubuntu>.

[18] *Popis VMWare Player* [online]. [cit. 9. 11. 2011]. Dostupný z WWW: <HTTP://www.vmware.com/products/player/overview.html>.

[19] *VirtualBox manual* [online]. [cit 9. 11. 2011]. Dostupný z WWW: <HTTP://www.virtualbox.org/manual/ch01.html>.

[20] *Instalace Wordpress* [online]. [cit. 8. 11. 2011]. Dostupný z WWW: <HTTP://codex.wordpress.org/Installing_WordPress>.

[21] MILLE, B.P. *Fuzz Testing of Application Reliability* [online]. [cit. 13. 11. 2011]. Dostupný z WWW: <HTTP://pages.cs.wisc.edu/~bart/fuzz/fuzz.html>.

[22] *Popis URLStress* [online]. Dostupný z WWW: <HTTP://sandark.com/URLStress.php>.

[23] MAVITUNA, Ferruh. *DoS Attacks Using SQL Wildcards* [online]. [cit. 11. 11. 2011]. Dostupný z WWW: <HTTP://www.slideshare.net/fmavituna/dos-attacks-using-sql-wildcards>.

[24] FOUNDATION, *OWASP*. *OWASP TESTING GUIDE* [online]. 2008. [cit. 11. 11. 2011]. Dostupný z WWW: <HTTPS://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf>.

[25] Wikipedia.org. *Garbage collector* [online]. [cit 9. 5. 2011]. Dostupný z WWW: <HTTP://en.wikipedia.org/wiki/Garbage_collection_(computer_science)>.

[26] W3. *HTTP Status* [online]. [cit 11. 5. 2011]. Dostupný z WWW <HTTP://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>.

[27] ZÁVODSKÝ, Petr. *OWASP: za webové aplikace bezpečnější* [online]. [cit. 13. 11. 2011]. Dostupný z WWW: <HTTP://www.root.cz/clanky/owasp-za-webove-aplikace-bezpecnejsi/>.

[28] KLEVINSKY, T. J. *Hack IT: security through penetration testing*. : Pearson Education, 2002.

[29] ROSIGNOLI , Simone. *How botnets work* [online]. [cit. 10. 5. 2011]. Dostupný z WWW: <HTTP://www.windowsecurity.com/articles/Robot-Wars-How-Botnets-Work.html>.

[30] KUBA, Martin. *Tutorial SOAP* [online]. [cit. 27. 4. 2011]. Dostupný z WWW: <HTTP://www.ics.muni.cz/~makub/soap/tutorial.html>.

[31] *PHP tutoriál* [online]. [cit. 20. 4. 2011]. Dostupný z WWW: <HTTP://php.net/manual/en/tutorial.php>.

[32] FOUNDATION, OWASP. *Popis WebGoat* [online]. [cit. 10. 5. 2011]. Dostupný z WWW: <HTTP://code.google.com/p/webgoat/>.

[33] *GNUPlot příklady* [online]. [cit. 10. 5. 2011], Dostupný z WWW: <HTTP://www.gnuplot.info/screenshots/index.html#demos>.

[34] *Russian Bussines Network* [online]. [cit. 11. 5. 2011]. Dostupný z WWW: <HTTP://en.wikipedia.org/wiki/Russian_Business_Network#cite_note-6>.

[35] ČERNÝ, Michal. *Kybernetická válka je reálnou hrozbou* [online]. [cit. 11. 5. 2011]. Dostupný z WWW: <HTTP://www.lupa.cz/clanky/kyberneticka-valka-je-realnou-hrozbou/>.

Seznam příloh

Příloha A - Příručka ovládání aplikace penNet

Příloha B - CD (aplikace, manuál, DP v elektronické podobě)

Příloha A

Uživatelská příručka

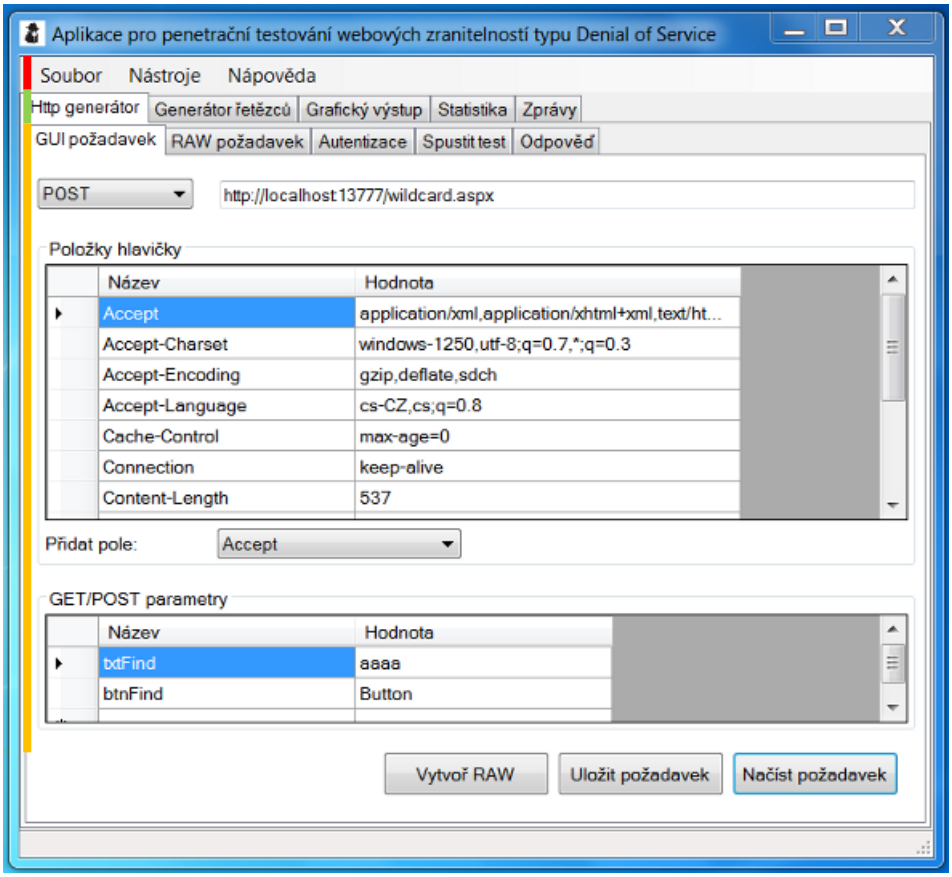
APLIKACE PRO PENETRAČNÍ TESTOVÁNÍ WEBOVÝCH ZRANITELNOSTÍ TYPU DENIAL OF SERVICE

Autor: Bc. Jaroslav Vrána

Obsah

1. Obecný popis aplikace

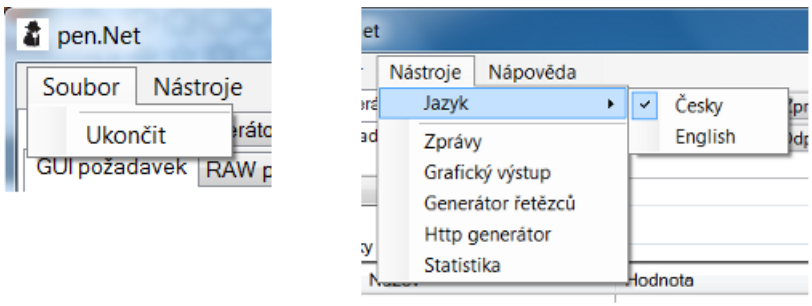
Aplikace s připraveným http požadavkem je znázorněna na obrázku 1-1. Aplikace se skládá ze tří částí. První částí je obecné menu, viz červený region. Druhá část je označena zeleně a obsahuje záložky načtených modulů. V posledním regionu je zobrazen samotný modul a na obrázku je zobrazen žlutou barvou.



1-1: Aplikace – první pohled

1.1 Obecné menu

Položky menu jsou Soubor, Nástroje a Nápověda, viz obrázek 1-1 a detail na obrázku 1-2. Aplikaci lze z menu Soubor ukončit. Menu Nástroje umožňuje změnit jazykové prostředí a přepínat se mezi moduly. Pomocí menu Nápověda lze zobrazit stručný popis aplikace.



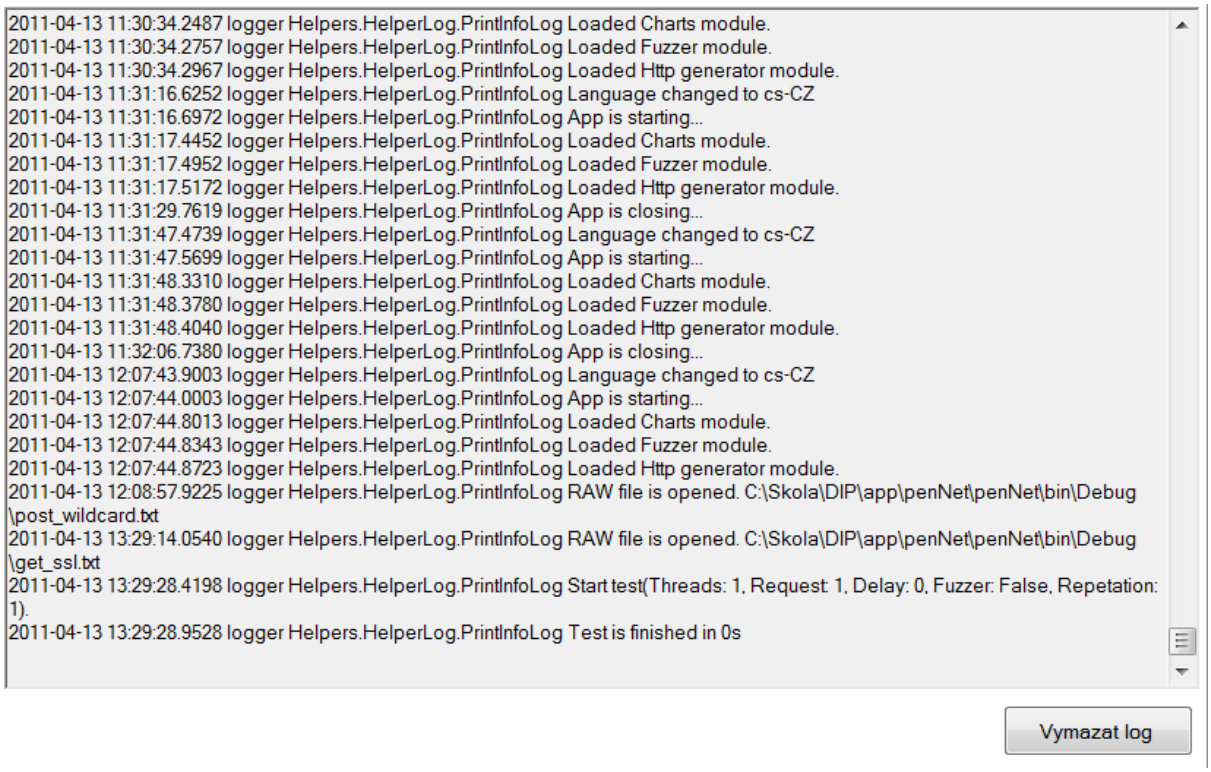
1-2: Položky obecného menu

1.2 Zprávy

Okno zprávy zobrazuje záznamy o aktivitách uživatele a chybová hlášení. Tyto záznamy jsou uchovány v souboru *log.txt*.

Tlačítka:

- *Vymazat log* - Smaže permanentně všechny záznamy.

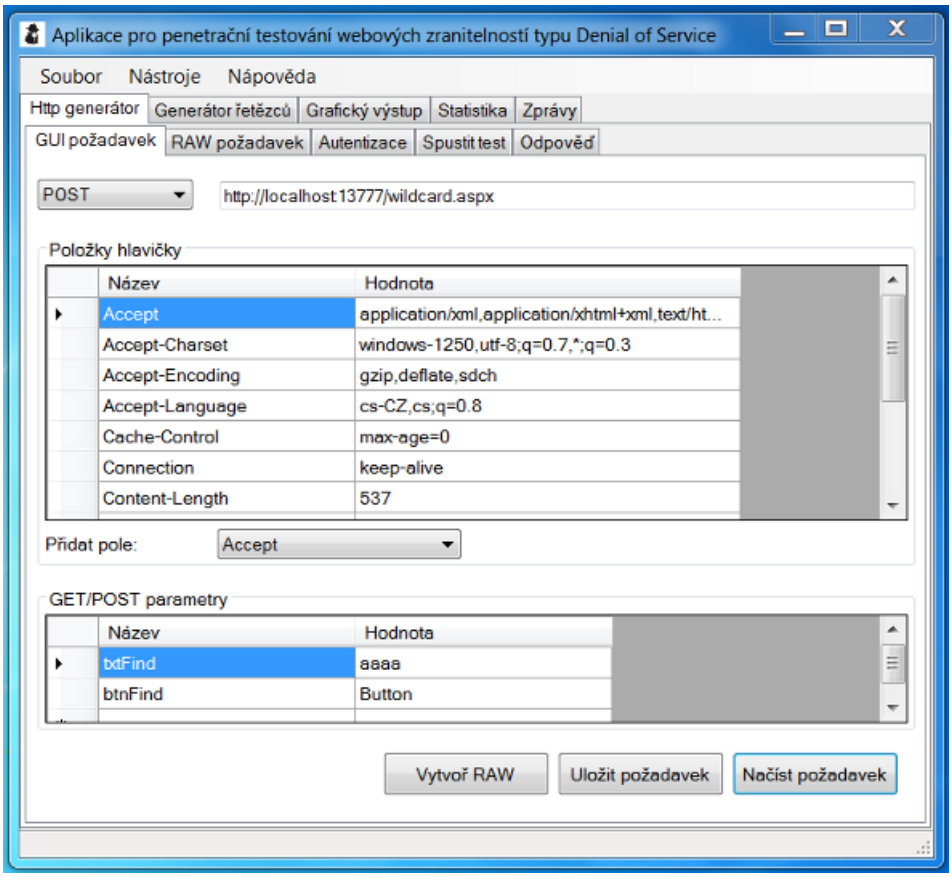


1-3: Zprávy

2 Http Generátor

Modul http Generátor se skládá ze čtyř částí (viz obrázek 2-1):

- *GUI Požadavek* – Tvorba http požadavku v GUI prostředí.
- *RAW požadavek* – Tvorba http požadavku v textové podobě.
- *Autentizace* – Nastavení způsobu autentizace.
- *Star test* – Nastavení způsobu testování.
- *Odpověď* – Seznam doručených http odpovědí s jejich metadaty (velikost, doba odezvy, …).



2-1: Http generátor

2.1 GUI Požadavek

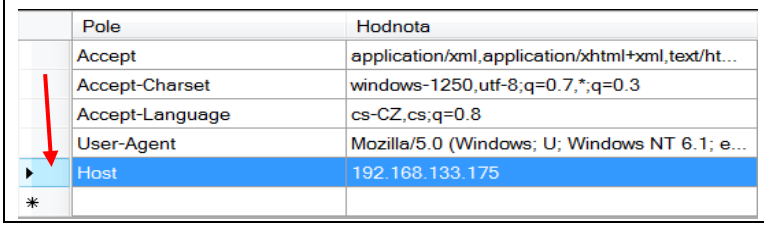
Záložka požadavek umožňuje tvorbu http požadavku v GUI prostředí. Tvorba http požadavku je podmíněna vyplněním *URL*, viz obrázek 2-1. Musí být zadáno plné jméno serveru (např. https://www.google.com), port (např. 443) a cesta v rámci serveru. Další částí je tvorba http hlavičky. Do tabulky je možné zadávat pole ručně nebo pomocí seznamu *Přidat pole*, které na konec tabulky přidá požadované pole a uživatel již musí pouze vyplnit požadovanou hodnotu. Jsou podporovány i další typy polí, které nejsou uvedeny v seznamu *Přidat pole* (např. pole Cookie). Uživatel je musí zadávat pomocí klávesnice. Uživatel si vybírá metodu pro odesílání http požadavku (GET/POST). Následuje tabulka *GET/POST parametry*, která umožňuje zadávat hodnoty formulářových položek, parametrů atd.

Tlačítka:

- *Vytvoř RAW* – Generuje RAW tvar http požadavku. (viz další kapitola).
- *Uložit požadavek* – Ukládá http požadavek, který může být opětovně použit.
- *Načíst požadavek* – Načítá http požadavek pro opětovné použití.

TIP

V celé aplikaci lze smazat řádek z tabulky kliknutím na zvolený řádek v levém nultém sloupci a následným stiskem klávesy *Delete*.

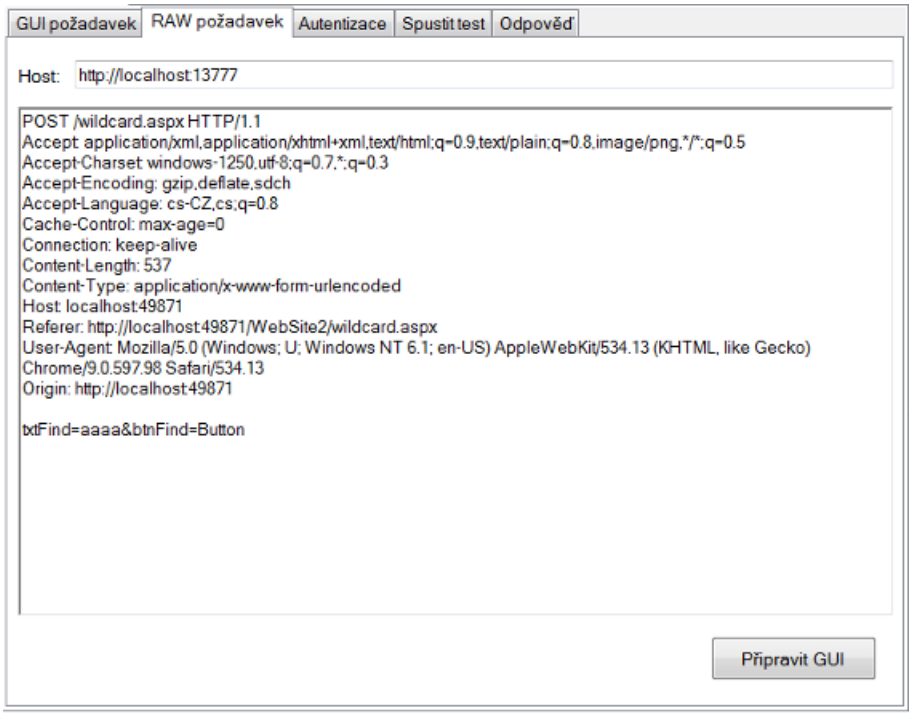


2.2 RAW požadavek

Záložka *RAW požadavek* umožňuje tvorbu http požadavku v textové podobě. Obrázek 2-2 znázorňuje RAW požadavek vytvořený v předchozí kapitole a následně vygenerovaný pomocí tlačítka *Vytvořit RAW*.

Tlačítka:

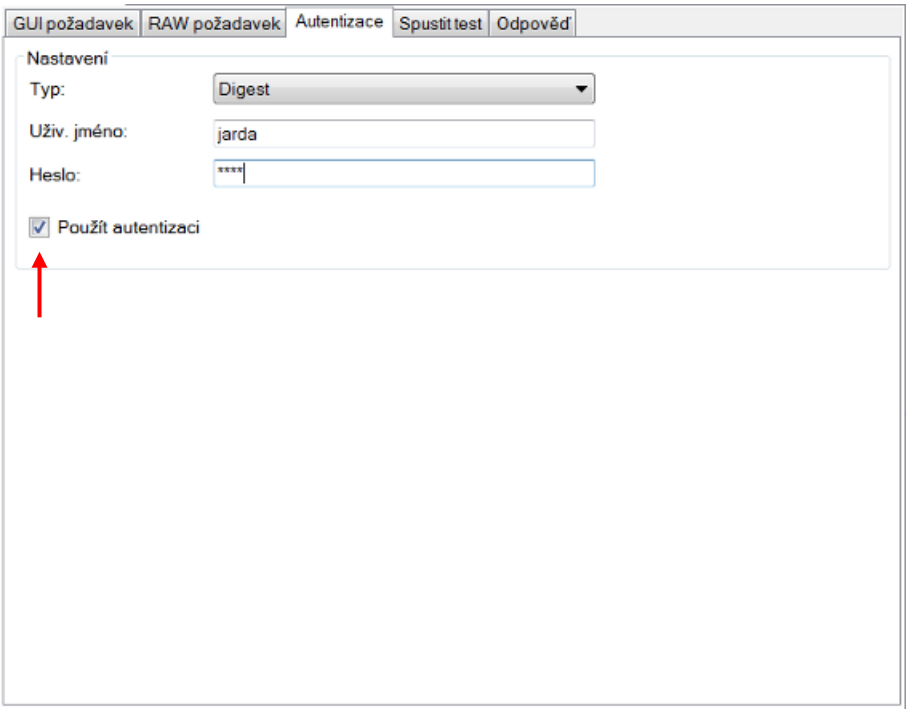
- *Připravit GUI* – Převede požadavek do záložky GUI požadavek.



2-2: Http generátor – RAW požadavek

2.3 Autentizace

Aplikace umožňuje autentizaci pomocí tří mechanismů (Basic, Digest a NTML) a právě tato nastavení se provádějí v záložce *Autentizace*. Obrázek 2-3 zobrazuje příklad nastavení autentizace.



2-3: Http generátor – Autentizace

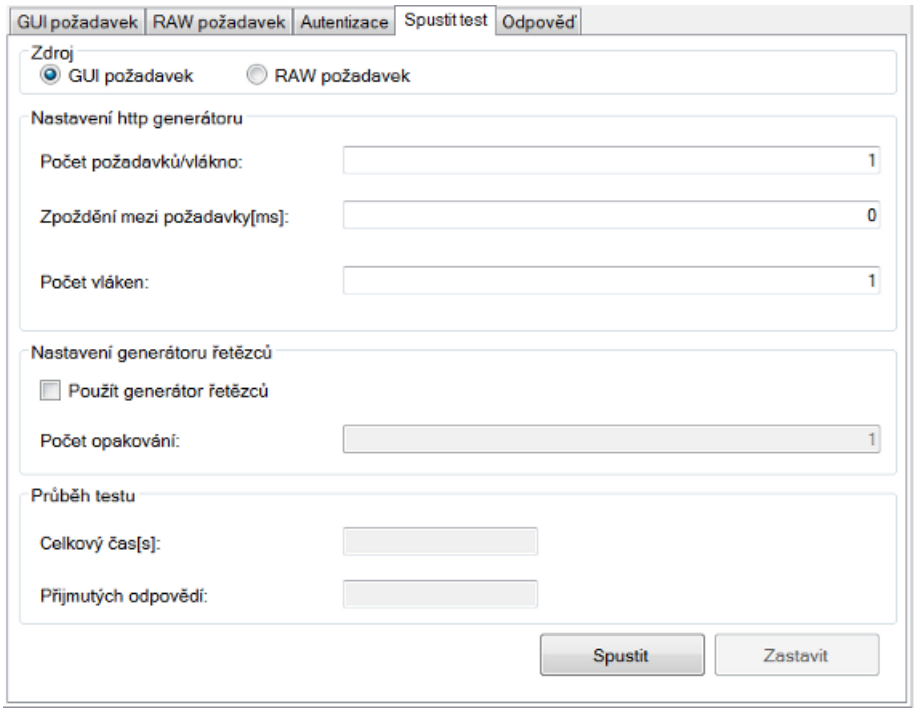
2.4 Start test

Záložka Start test umožňuje konfigurovat způsob odeslání požadavku, případně skupiny požadavků. Záložka se skládá ze tří hlavních částí:

- *Zdroj* – Přepínání mezi GUI a RAW zdrojem požadavků.
- *Nastavení http generátoru* – Umožňuje nastavit počet odeslaných požadavků, zpoždění mezi nimi a počet vláken, ze kterých se bude požadavek odesílat.
- *Nastavení generátoru řetězců* – Umožňuje navíc nastavit počet opakování testu při použití generátoru řetězců (viz kapitola 3). Při použití generátoru řetězců a více vláken se bude v každém vlákně samostatně generovat nezávislé vlastní test od začátku do konce.
- *Průběh testu* – Zobrazuje průběh testu.

Tlačítka:

- *Spustit* – Spustí odesílání http požadavku/požadavků.
- *Zastavit* – Ukončí probíhající test.



2-4: Http Generátor – Test

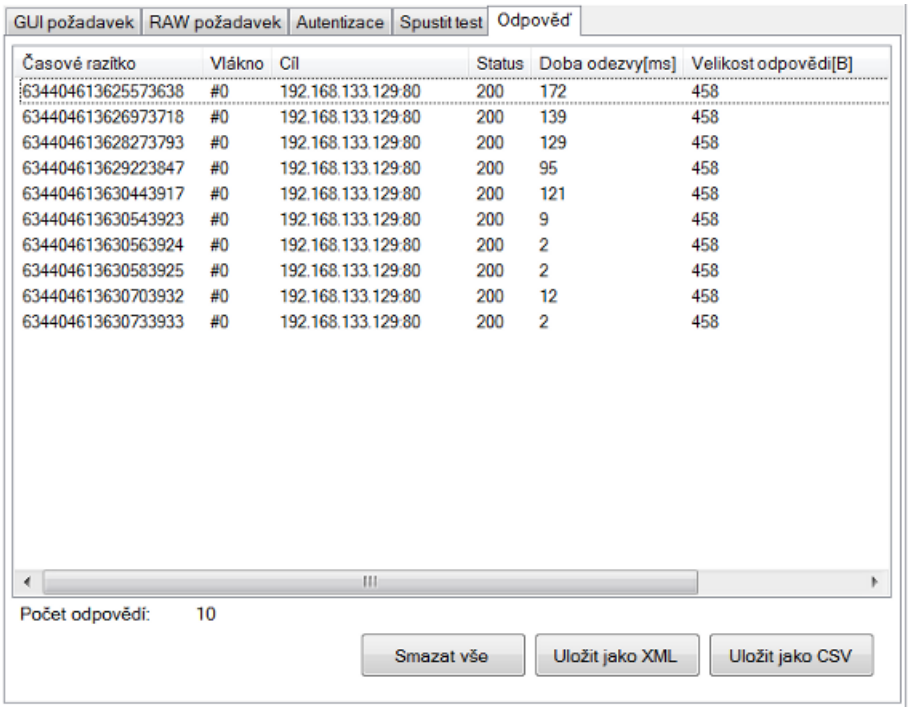
Poznámka
Maximální počet poslaných požadavků, vláken a doby zpoždění není z experimentálních důvodů omezen, z čehož vyplývá, že při některých konfiguracích může docházet k chybovým stavům.

2.5 Odpověď

Tato záložka obsahuje seznam přijatých http odpovědí serveru a případně výpis chybových stavů (označených statusem -1) aplikace při odesílání http požadavků (např. server nenalezen). Formát výstupu je zobrazen na obrázku 2-5.

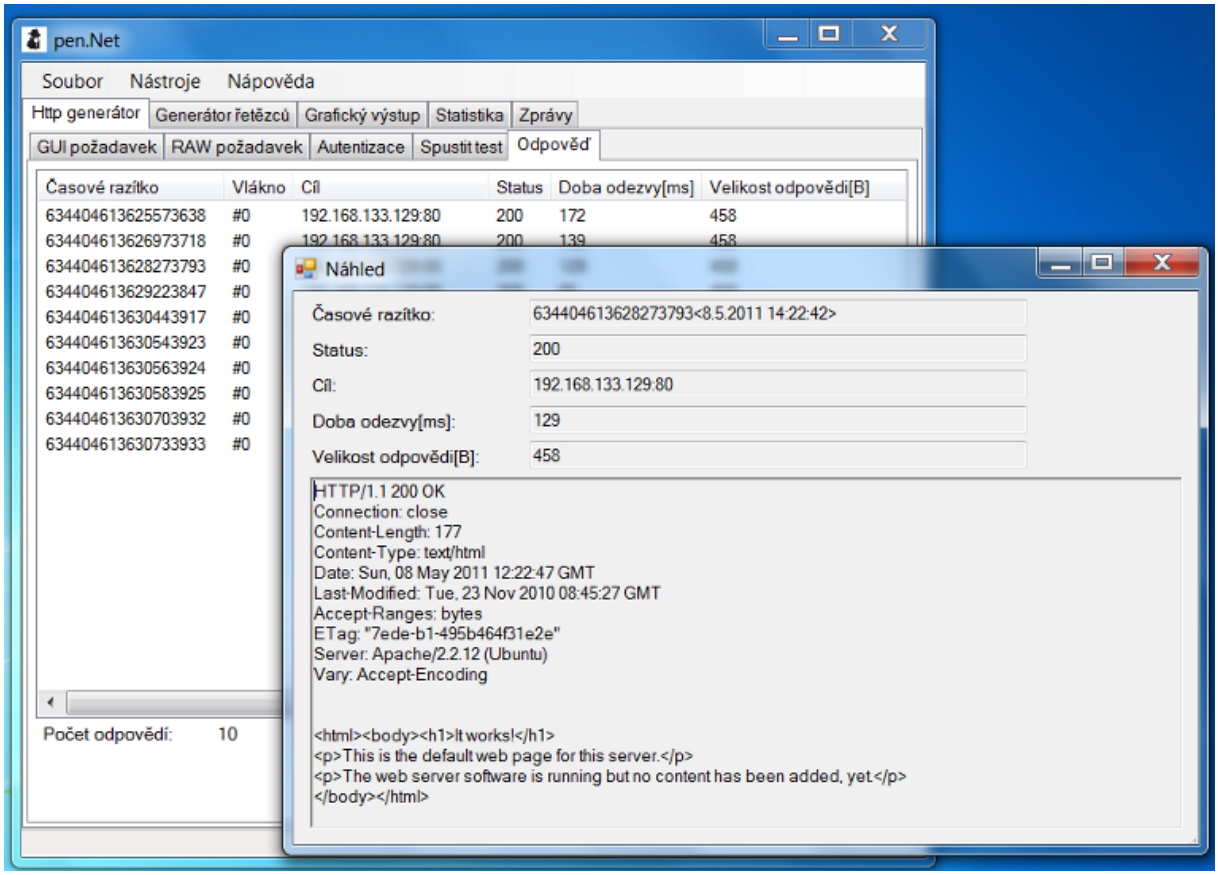
Tlačítka:

- *Smazat vše* – Smaže celý seznam http odpovědí.
- *Uložit jako XML* – Uloží kompletní seznam http odpovědí včetně metadat do XML souboru.
- *Uložit jako CSV* – Uloží metadata http odpovědí do souboru ve formátu CSV pro zpracování v některém z externích nástrojů (GNUPlot, OpenOffice, MS Excel, …).



2-5: Http Generátor – Odpověď

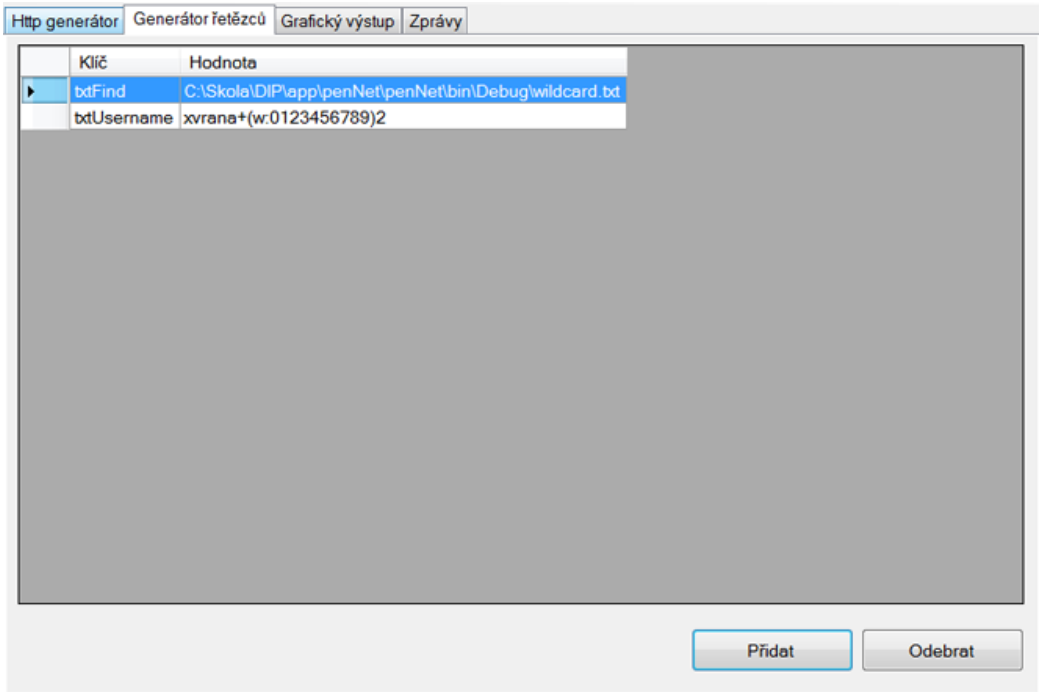
Dvojitým poklepáním na požadovaný řádek se zobrazí detailní náhled http odpovědi, viz obrázek 2-6.



2-6: Http Generátor - Odpověď – Detail

3 Generátor řetězců

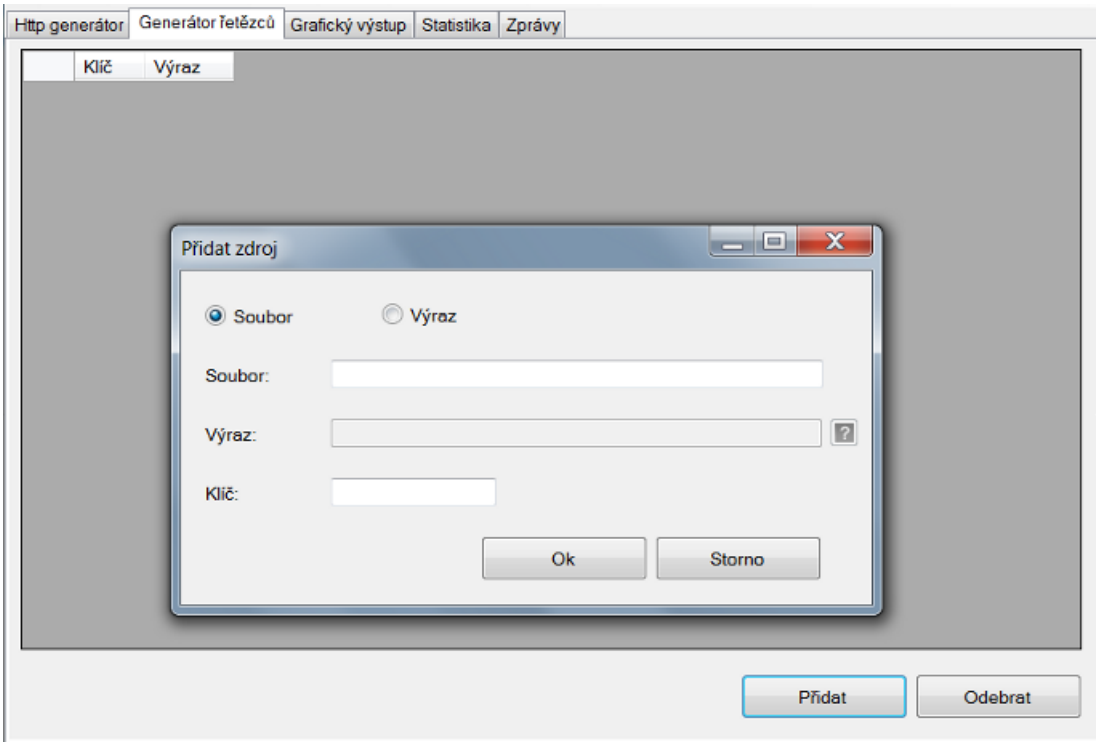
Modul *Generátor řetězců* funguje jako zdroj testovacích hodnot pro položky http požadavků (např. cookies, atd). Zdrojem hodnot mohou být soubory nebo výrazy, které uživatel může volně zapisovat. Obrázek 3-1 zobrazuje záložku *Generátor řetězců* se dvěma generátory. První používá jako zdroj soubor a druhý uvedený výraz. První generátor bude číst řádky zadaného souboru a postupně je přiřazovat klíči txtFind, který je definovaný v http požadavku, viz obrázek 1-1. Druhý generátor bude postupně klíči txtUsername přiřazovat hodnoty generované zadaným výrazem, tj. xvrana00, xvrana01 , …, xvrana99. Obdobně, pokud bude definováno pole Cookie: hodnota1=aaa; hodnota2=bbb, mohou být pro klíče hodnota1 a hodnota2 definovány generátory.



3-1: Generátor řetězců

Tlačítka:

- *Přidat* – Přidává nový generátor, viz obrázek 3-2.
- *Odebrat* – Odebírá zvolený generátor.



3-2: Generátor řetězců - přidat zdroj generátoru

3.1 Způsob vytváření generátorů pomocí výrazů

Výrazy se mohou skládat ze čtyř typů generátorů:

- Konstantní řetězec – Generátor vždy vrátí nezměněný řetězec, např. řetězec xvrana, vždy vrátí řetězec xvrana.
- Generátor čísel – Generátor postupně vrací další číslo v řadě. Formát generátoru je (d: min, max, krok), konkrétně např. výraz (d: 0, 9, 1) bude vracet řetězce 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Krok musí být kladné nenulové číslo.
- Generátor slov – Generátor postupně vrací další slovo v řadě. Formát generátoru je (w:abeceda)délka_slova, konkrétně např. výraz (w:abc)2 bude vracet řetězce aa, ab, ac, ba, bb, bc, ca, cb, cc. Jedná se o variace s opakováním, tedy počet variací je $V_k'(n) = n^k$, kde *n* je počet písmen a *k* je délka slova.
- Složený generátor – Výše zmíněné generátory lze konkatenovat pomocí operátoru „+“, např. výraz x+(w:ab)2+(d:0,1,1) bude generovat řetězce xaa0, xaa1, xab0, xab1, xba0, xba1, xbb0, xbb1.

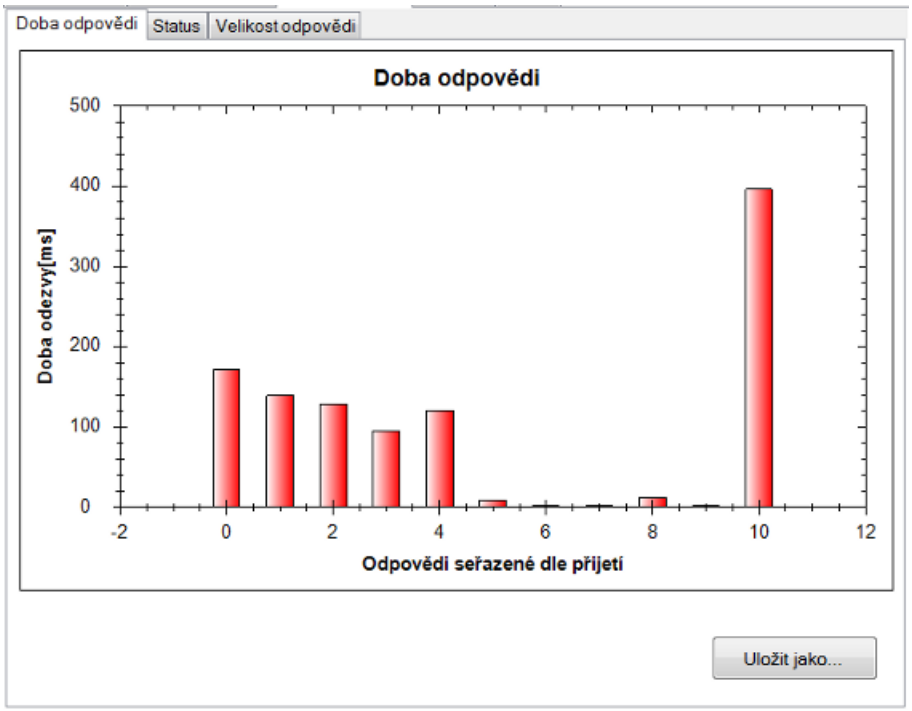
4 Grafický výstup

Modul *Grafický výstup* zobrazuje vybrané vlastnosti http odpovědi v grafické podobě. Díky možnosti exportovat http odpovědi do XML nebo CSV formátu si může uživatel graficky znázornit i další vlastnosti http odpovědi v jím požadované grafické podobě. Modul *Grafický výstup* umožňuje sledovat statistiku velikosti http odpovědi, dobu odezvy http odpovědi a hodnotu statusu http odpovědi, viz následující obrázky, kde je graficky znázorněn test na wildcard zranitelnost.

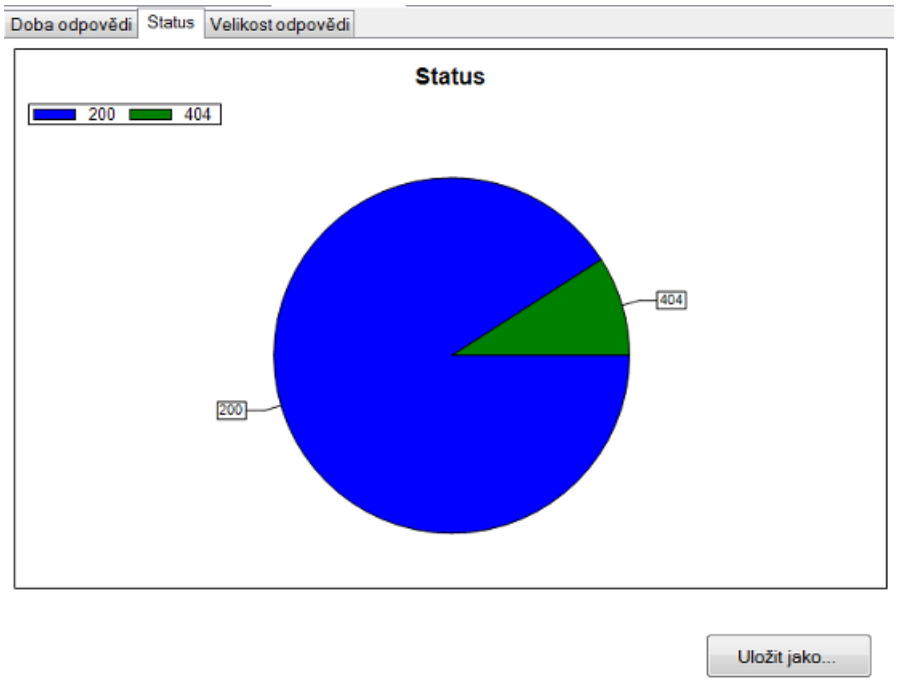
Omezením modulu Grafický výstup je, že maximálně zobrazí sto odpovědí. Je možné zobrazit větší množství odpovědí, ale vykreslení nemusí být přehledné a je potřeba ruční manipulace, což mohlo vést k dojmu, že vykreslování nefunguje správně. Tudíž byl maximální počet omezen.

Tlačítka:

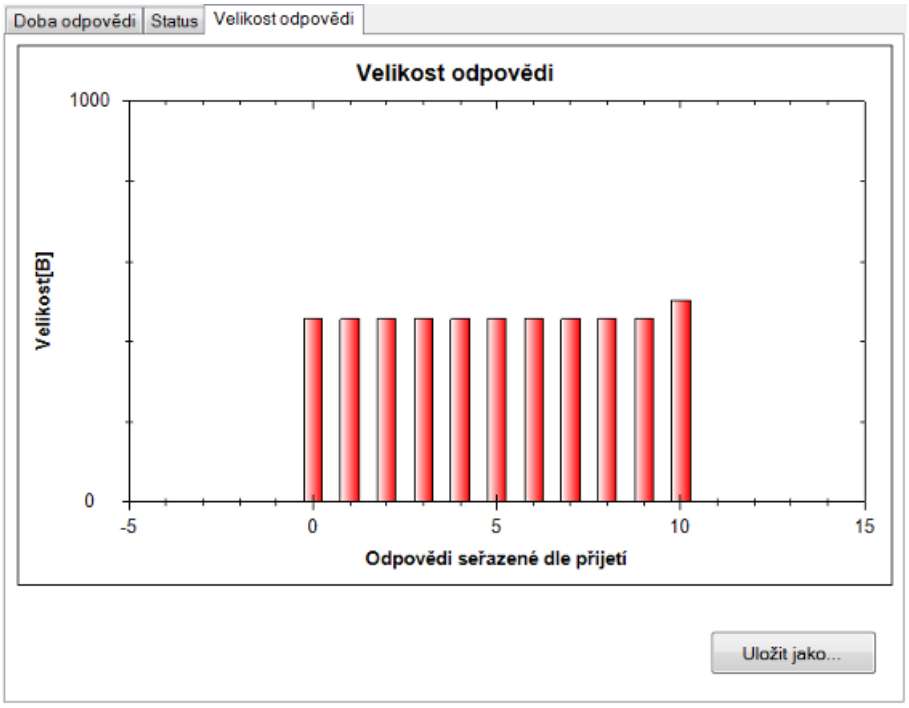
- *Uložit jako …* - Ukládá graf do zvoleného obrazového formátu.
-



4-1: Doba odezvy



4-2: Status http odpovědi



4-3: Velikost http odpovědi

TIP
S výslednými grafy je možné dále manipulovat. Na graf je třeba kliknout. Držením prostředního tlačítka myši a tahem ruky lze grafem posouvat a rolováním kolečka myši lze graf přibližovat nebo oddalovat. Tyto techniky jsou vhodné při větším počtu naměřených hodnot a hledání vhodné perspektivy.

5 Statistika

Tento modul je implementován jako jednoduchá ukázka tvorby modulů. Zobrazuje jednoduchou statistiku o velikostech a dobách odezvy přijatých odpovědí.

5-1: Statistika