VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INTERAKTIVNÍ WEBOVÉ ROZHRANÍ PRO ZOBRAZENÍ IP FLOW DAT

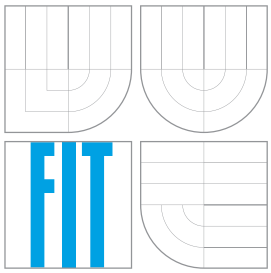
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. RADEK SALAČ

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INTERAKTIVNÍ WEBOVÉ ROZHRANÍ PRO ZOBRAZENÍ IP FLOW DAT

INTERACTIVE WEB INTERFACE FOR IP FLOW DATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. RADEK SALAČ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR MATOUŠEK, Ph.D.

BRNO 2012

Abstrakt

Tato práce popisuje vytvoření aplikace pro práci s IP flow daty. Provádí srovnání již existujících protokolů a nástrojů a snaží se najít jejich silné a slabé stránky. Na základě jejich analýzy a sběru požadavků je pak vytvořena vlastní aplikace, která si klade za cíl poskytnout interaktivní a uživatelsky přívětivé prostředí pro práci s IP flow daty.

Abstract

This thesis describes development of application for analyzing IP flow data. The author conducts relative comparison of already existing protocols and tools and studies theirs pro's and con's. Based on this comparison and features requested by users, author develops his own application primarly focused on interactive and user-friendly interface for working with IP flow data.

Klíčová slova

IP flow, NetFlow, IPFIX, nfdump, bezpečnost

Keywords

IP flow, NetFlow, IPFIX, nfdump, security

Citace

Radek Salač: Interaktivní webové rozhraní pro
zobrazení IP flow dat, diplomová práce, Brno, FIT VUT v Brně, 2012

Interaktivní webové rozhraní pro zobrazení IP flow dat

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Petra Matouška, Ph.D.

.....
Radek Salač
21. května 2012

Poděkování

Na tomto místě bych chtěl v prvé řadě poděkovat panu Ing. Petru Matouškovi, Ph.D. za jeho odborné rady a vedení v průběhu psaní této práce, stejně tak za jeho trpělivost v průběhu jejího opravování. Dále bych chtěl poděkovat panu Ing. Tomáši Podermańskému za připomínky a rady v průběhu implementace. Dále bych chtěl poděkovat celé skupině Sec6net zejména pak Ing. Matěji Grégrovi, Bc. Miroslavu Šoltésovi a Ing. Martinu Žádníkovi. V neposlední řadě bych chtěl poděkovat všem, bez nichž by tato práce nemohla vzniknout.

Obsah

Kapitola 1

Úvod

Stejně jako vzrůstá objem přenesených dat prostřednictvím počítačových sítí, vzrůstá i potřeba tyto sítě efektivně spravovat. Abychom tohoto byli schopni dosáhnout, musíme mít detailní znalosti ohledně charakteru síťového toku, který naší počítačovou sítí protéká. Bez této znalosti jsme schopni jen velmi obtížně detekovat problémové body naší síťové topologie a v případě problémů sjednat nápravu.

Dále je nutno si uvědomit narůstající vliv počítačové kriminality. Zde můžeme také využít nástroje pro analýzu síťového toku, kdy s jejich pomocí vystopujeme útočníka, který pomocí naší sítě prováděl kriminální činnost, nebo se naopak snažil na naši síť útočit.

Dalším zájemcem o vhodnou analýzu síťového toku mohou být různá hostingová centra, která potřebují na základě přenesených dat účtovat klientům poplatky.

Abychom mohli úspěšně vyřešit výše zmíněné, musíme mít metriku a data, jež toky v počítačových sítích rozumným způsobem popisují.

Data popisující síťový tok nazýváme *flow* daty. Existuje mnoho různých implementací těchto dat. Mezi nejznámější patří NetFlow [4] od firmy Cisco, dále pak IPFIX [6], který z NetFlow vychází, nebo sFlow [17].

Problematika práce s daty pro popis síťového toku je značně rozsáhlá. Zahrnuje jak samotný sběr dat, tak jejich uchování a následné vyhodnocení. Každá část procesu má své vlastní problémy a úskalí, z nichž některým se budeme dále věnovat v samostatných kapitolách.

Cílem práce je vytvořit interaktivní nástroj pro práci s daty pro popis síťového toku, který by nahradil nebo doplnil portfólio již existujících nástrojů. Abychom toho byli schopni dosáhnout, musíme se s těmito nástroji seznámit a nalézt jejich silné a slabé stránky. Zajímat nás bude také jejich rozšiřitelnost a licence, pod kterou jsou distribuovány.

Náš nástroj by měl nalézt praktické uplatnění v univerzitní síti Vysokého učení technického v Brně a mělo by být možné jej provozovat na linuxovém serveru, ideálně jako webovou aplikaci. Musí podporovat práci více uživatelů s možností určit přístupová práva k příslušným datům minimálně na úrovni jednotlivých podsítí. Zpracovaná data musí být možno dále exportovat v podobě vhodné pro další zpracování.

Samostatnou kapitolu budeme věnovat různým možnostem pro ukládání dat o síťovém toku. Vzhledem k jejich množství bude naším cílem najít způsob, jenž by nám umožnil uchovávat tato data hospodárným způsobem, který by přehnaně nezatěžoval kapacitu našeho úložného zařízení. Zároveň však musíme vyhovět požadavku na co možná nejrychlejší přístup k datům.

V této práci se zaměříme hlavně na již existující databázové systémy, které by v případě úspěšného použití mohly zásadním způsobem ovlivnit vývoj naší aplikace. Více se tomuto tématu věnujeme v kapitole 4.

Vzhledem k povaze údajů, se kterými bude aplikace pracovat, je potřeba dbát na důkladné zabezpečení. Bezpečnost webových aplikací, a nejen těch, je v poslední době poměrně velmi důležité téma. S tím, jak se firmy stávají neustále závislejšími na informačních systémech, jsou tyto systémy stále lákavějším cílem útoků.

Přestože problém správného zabezpečení je již velice dobře popsán v mnoha odborných publikacích, za příklad si můžeme uvést Hacking - umění exploitace[8] a Bezpečný kód[10], praxe ukazuje, že je stále mnoho, a to i velkých, společností a projektů, které nekladou dostatečný důraz na bezpečnost vyvíjených aplikací. Z poslední doby si můžeme vzít za příklad útok na společnost Sony z roku 2011. V průběhu tohoto útoku byla z databází ukradena čísla kreditních karet[1] a nezašifrovaná hesla v čitelné podobě[2]. Proto jsme se rozhodli tomuto problému věnovat samostatnou sekci 6.

Aplikace by také měla podporovat součinnost s policií a ministerstvem vnitra. Z tohoto důvodu bude potřeba nastudovat příslušnou vyhlášku[1] definující požadavky na uchovávání, prezentaci a způsob předávání těchto údajů příslušným orgánům.

V závěru práce provedeme srovnání námi vytvořené aplikace s již existujícími nástroji a provedeme diskuzi přínosů. Zároveň se pokusíme naznačit, jakým směrem by se měl následující vývoj naší aplikace ubírat a jaké by mohly být směry jeho dalšího rozšíření.

[1] http://www.mobiledia.com/news/111980.html
[2] http://www.theregister.co.uk/2011/06/08/password_re_use_survey/

Kapitola 2

Analýza síťového toku

Abychom byli schopni vytvořit kvalitní nástroj pro práci s daty pro popis síťového toku, musíme se o těchto datech nejdříve více dozvědět. Je proto nutné získat alespoň základní představu o tom, jakým způsobem se tato data sbírají, uchovávají a dále analyzují. Stejně tak se musíme seznámit s problémy, které při jejich zpracování nastávají.

V následující kapitole si proto důkladně vysvětlíme, co se rozumí pod pojmem síťový tok. Popíšeme si, jakým způsobem lze sbírat data o síťovém toku a jakým způsobem je lze dále zpracovávat. Nakonec se seznámíme s nejčastěji používanými protokoly, jež se v této oblasti používají, včetně jejich silných a slabých stránek.

Síťovým tokem rozumíme: Posloupnost paketů vycházejících z jednoho určitého zdroje do jednoho určitého cíle se nazývá tok (flow). Cíle mohou byt přímé (unicast) a vícesměrové (anycast, multicast). Tok může sestávat ze všech paketů náležících některému přenosovému spojení (transport connection) nebo vysílání multimediálního obsahu. Tok však nemusí byt přiřazen k jednomu přenosu právě v poměru 1:1[18].

Tradiční klasifikátory síťového toku byly založeny na pětici parametrů: zdrojová a cílová adresa, zdrojový a cílový port a typ transportního protokolu. Avšak některé z těchto údajů nemusí být vždy získatelné, například z důvodu fragmentace nebo šifrování.

Správný popis síťového toku by tedy měl splňovat tyto charakteristiky:

- Lze jej aplikovat na libovolný protokol pomocí atributů, jež popisují daný síťový tok. Atributy síťového toku jsou definovány takovým způsobem, aby byly nezávislé na protokolu, a musí správně fungovat v multiprotokolovém systému.
- Uživatelé jsou schopni definovat množinu pravidel, jež jim umožní získávat data popisující síťový tok, která je zajímají, a ignorovat data jiná.
- Snaha o efektivní implementaci vyžaduje mít možnost umístit sondu sbírající data pro popis síťového toku přímo na aktivní prvek.
- Základní jednotkou je flow, což odpovídá jednomu spojení, a naším cílem je toto spojení popsat. Typickými atributy jsou zdrojová adresa, cílová adresa, port, čas, protokol a jiné. Nicméně žádný z těchto atributů není povinný.
- Musí jít o druh metriky, který se snaží popsat chování počítačové sítě.

- Data popisující síťový tok jsou většinou získávána ze směrovačů (routerů), případně sond, odkud jsou přenášena na kolektor, kde jsou ukládána pro pozdější použití.
- Dále jsou definovány metriky pro měření a srovnání dat popisujících síťový tok. Tyto metriky jsou plně multiplatformní, umožňující snadnou přenositelnost.

Všechny výše zmíněné body jsou značně obecné, což dává prostor pro více možných implementací, z nichž jsme již některé jmenovali a některé si představíme důkladněji.

Základní model zpracování dat pro popis síťového toku je popsán v RFC 2722[2].

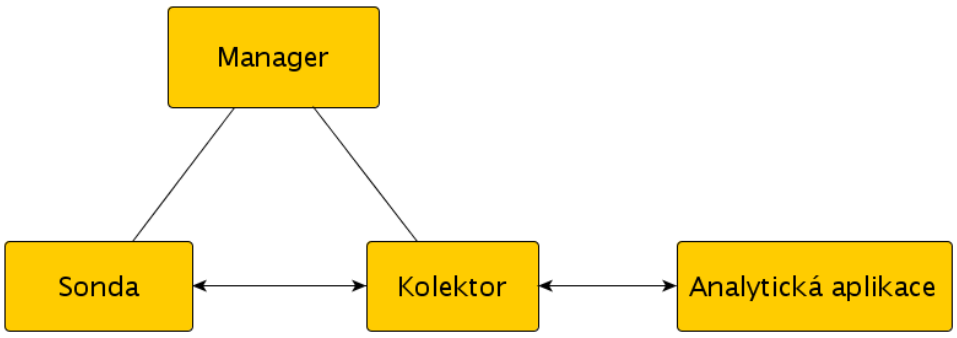
Obsahuje tyto prvky:

Manager (v originále MANAGER): Nástroj pro konfiguraci sondy a pro kontrolu kolektoru. Posílá konfigurační příkazy na jednu nebo více sond a monitoruje jejich funkčnost. Je možné spojit funkcionalitu managera a kolektoru do jednoho síťového prvku.

Sonda (v originále METER): Nástroj pro měření a generování dat o síťovém toku podle zvolených pravidel. Umožňuje nad těmito daty dělat základní předzpracování, mezi které patří například filtrace nebo transformace těchto dat. Výsledná data následně posílá do úložiště.

Kolektor (v originále METER READER): Čte a ukládá data o síťovém toku ze sondy. Tato data následně poskytuje analytické aplikaci k dalšímu zpracování.

Analytická aplikace (v originále ANALYSIS APPLICATION): Vlastní analytická aplikace, která poskytuje informace z nasbíraných dat a generuje z nich přehledy užitečné pro koncového uživatele. Může se jednat například o objem přenesených dat prostřednictvím konkrétního síťového prvku.



Obrázek 2.1: Základní schéma komunikace mezi prvky pro práci s daty popisujícími síťový tok. Sonda čte data o síťovém toku a posílá je na kolektor, který tato data uchovává pro další zpracování analytickou aplikací. Činnost sondy a kolektoru je řízena managerem.

Jak jsme již předeslali dříve, analýza síťového toku je velmi důležitá, pokud chceme svědomitě a zodpovědně spravovat rozsáhlejší počítačové sítě. Jen díky ní jsme schopni:

- zachytit anomálie v síťovém toku a potenciální bezpečnostní incidenty,
- vyhodnotit využití síťových prvků,
- nalézt nejproblémovější místa v síti,
- vyhodnotit dopad změn v síťové topologii.

Jak jsme se zmínili výše, IP flow nepředepisuje přesně žádné atributy, které jej definují. Je pouze na konkrétní implementaci, jaké atributy zvolí za natolik významné, že by měly být definicí flow.

Krom zmíněné pětice se jedná například o:

- začátek přenosu
- konec přenosu
- počet přenesených bitů
- počet přenesených paketů
- hodnota TOS[1]
- množina TCP příznaků[2] a mnohé další.

Některé implementace navíc umožňují volit si vlastní atributy definující flow.

V současnosti existuje několik různých nástrojů a protokolů pro tuto analýzu. Mezi nejrozšířenější protokoly používané v této oblasti patří:

NetFlow (poslední verze má číslo 9): V této práci budeme tento protokol brát jako referenční.

IPFIX: Vychází z NetFlow a přidává některé další funkce jako je například zabezpečený přenos dat mezi sondou a kolektorem.

sFlow: Cílem protokolu je dosáhnout co nejvyššího výkonu, k čemuž využívá vzorkování. Technika vzorkování spočívá v tom, že neuchováváme a nezaznamenáváme veškerá data o síťovém toku, ale jenom jejich vzorek. Statistickými metodami následně dopočítáváme údaje, abychom získali přehled o síťovém toku v naší počítačové síti. Vzorkování je možno samozřejmě využít i u jiných protokolů, nicméně zde se s ním počítá už v době návrhu.

Mimo výše zmíněné existují i další implementace, například Rflow[3], NetStream[4] a J–Flow[5]. Většinou se od sebe ale příliš neliší a základní princip fungování zůstává nezměněn.

[1] Type of service
[2] ACK, SYN, FIN ...
[3] http://www.dd-wrt.com/wiki/index.php/RFlow_Collector
[4] www.huawei.com/products/datacomm/pdf/view.do?f=65
[5] http://www.juniper.net/techpubs/software/erx/junose82/swconfig-ip-services/html/ip-jflow-stats-config2.html#57715

V naší práci se zaměříme výhradně na NetFlow a IPFIX. Důvod je ten, že protokol NetFlow je používán v univerzitní síti Vysokého učení technického v Brně (Dále jen VUT Brno) a my jej budeme intenzivně využívat. Protokol IPFIX z něj vychází a byl definován jako standard, který je popsán v RFC 5101 [6], občas se o něm hovoří jako NetFlow verze 10. Oba protokoly jsou spolu tedy velmi úzce spjaté.

2.1 Protokol NetFlow

NetFlow je otevřený, multiplatformní protokol vyvinutý firmou Cisco. Má několik verzí, z nichž poslední je verze 9, z které se však dále vyvinul IPFIX. Touto verzí se budeme zabývat i my. Protokol NetFlow vychází z definice flow a velká část terminologie je stejná nebo podobná. Specifikace protokolu je zdokumentovaná a volně dostupná v RFC 3954 [4].

Obecné pojmy architektury NetFlow jsou:

Sledovaný bod: Bod v síti, na kterém chceme monitorovat provoz. Může se jednat například o rozhraní směrovače.

Množina sledovaných bodů: Vzniká agregací více sledovaných bodů na jednom zařízení. Může se jednat například o směrovač, na němž chceme sledovat všechna rozhraní.

NetFlow záznam: Strukturovaná informace zachycující charakter síťového toku.

Exportér: Zařízení, jež je schopno generovat NetFlow záznamy na základě probíhajícího toku.

NetFlow kolektor: Zařízení sbírá informace z Exportéru a ukládá je do úložiště (pevný disk, databáze atd.). Navíc může provádět základní operace, jako je filtrování nebo agregace, předtím, než dojde k uložení.

Základní princip fungování je takový, že Exportér zachycuje probíhající tok a generuje NetFlow záznamy. V určité chvíli jsou tyto záznamy odeslány jako zpráva do NetFlow kolektoru. Do verze 9 byla struktura této zprávy značně neflexibilní, od verze 9 je však možné tvar zprávy ovlivnit pomocí šablon. Dané šablony umožňují definovat sbíraná data, a tak například omezit velikost přenášených záznamů z exportéru do kolektoru pouze na ty, jež jsou pro nás relevantní. Tyto optimalizace potom mají pozitivní vliv na výkon celého systému.

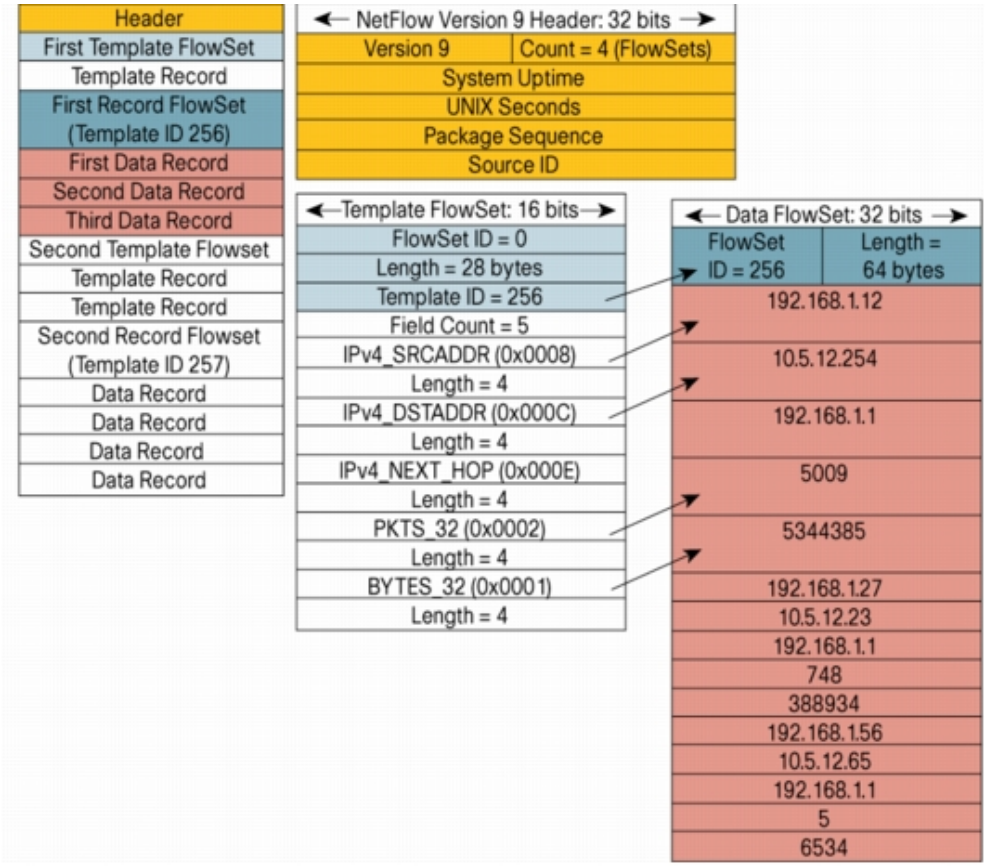
Podoba NetFlow paketu je zachycena na obrázku 2.2.

Paket se skládá z těchto části:

- Hlavička (Header) poskytuje základní údaje o daném paketu. Asi nejdůležitějším údajem je pole Version a Count.
- Šablony (Template FlowSet) pro popis přenášených údajů.
- Vlastní přenášené záznamy (Record FlowSet).

Hlavička obsahuje následující údaje:

- Version - Číslo verze NetFlow protokolu.
- Count - Počet přenášených FlowSet záznamů.



Obrázek 2.2: Podoba NetFlow paketu v 9 verzi. Obrázek byl převzat z [3].

- System Uptime - Počet milisekund od chvíle, kdy byl exportér spuštěn.
- UNIX Seconds - Unixový čas, jenž odpovídá počtu vteřin od 1.1.1970.
- Package Sequence - Počítadlo exportovaných paketů z Exportéru. Tato hodnota může být použita pro detekci ztráty daného exportu.
- Source ID - Identifikátor konkrétního exportéru.

Pokud se podíváme podrobněji na šablonu (Template FlowSet), máme možnost vidět tato pole:

- FlowSet ID - Toto pole se používá k rozeznání šablony od datové věty. V případě šablony má vždy hodnotu 0.
- Length - Délka v bytech následující šablony. Jeden FlowSet může obsahovat i více šablon.

- Template ID - Identifikátor konkrétní šablony.
  - Field Count - Počet datových polí v dané zprávě, následuje popis těchto polí v podobě dvojice:
    * Typ údaje uložený v tomto poli, například zdrojová IPv4 adresa.
    * Délka pole v bytech.

    Tato dvojice údajů se může opakovat.

  Za sebou může následovat více šablon.

Poslední část paketu, kterou si musíme popsat, jsou vlastní přenášená data. Struktura těchto dat je následující:

- FlowSet ID - Nenulová hodnota tohoto pole značí, že se nejedná o šablonu. Hodnota FlowSet ID pak odpovídá Template ID použité šablony.
- Length - Délka datové věty.
- Vlastní data.

Zajímavým problémem je, jak zachytit celý síťový tok. Jak vyplývá z definice, síťový tok je určitá sekvence paketů, jež mají jisté společné znaky. Jedním z důležitých atributů flow je dozajista čas začátku a čas jeho konce. Získat hodnotu prvního atributu je veskrze snadné. Druhý atribut je problematičtější. U TCP toku nám většinou stačí ukončit flow při obdržení FIN paketu. U UDP však žádnou takovou pomoc nemáme. Navíc i u TCP může například dojít k výpadku linky či jiné nepředpokládané události. Proto většinou pokud neobdržíme paket s danou charakteristikou po předem daný časový úsek, považujeme toto flow za ukončené.

Dalším problémem NetFlow protokolu je bezpečnost. V době návrhu se předpokládalo, že jak kolektor, tak exportér budou v jedné privátní síti, ideálně propojeny dedikovanou linkou. Přestože tomu tak ve většině případů pravděpodobně bude, nelze se na toto spolehnout a nic nebrání tomu, aby přenos probíhal po nezabezpečeném médiu nebo prostřednictvím veřejně dostupné sítě. Bohužel, data jsou mezi exportérem a kolektorem přenášena v nešifrované podobě. To umožňuje odhalení potenciálně citlivých údajů, případně podvržení falešných dat.

Jednou z možností, jak problém řešit, je využít zabezpečeného spojení, například pomocí SSH[6] nebo utility stunnel[7]. Další variantou je využít nástupce NetFlow v9 s názvem IPFIX, který výše zmíněné problémy řeší uspokojivě již ve svém návrhu.

2.2 Protokol IPFIX

IPFIX [6] vznikl evolucí protokolu NetFlow, a proto se také občas nazývá NetFlow v10. Mezi nejdůležitější prvky, v nichž se liší od NetFlow, patří podpora následujících vlastností:

- Kontrola integrity přenášených dat a zajištění, že případný útočník nebude schopen tato data změnit.

[6] http://www.ssh.com/
[7] http://www.stunnel.org/

- Zabezpečení přenášených dat proti odposlechu útočníkem a zabránění odhalení citlivých údajů.
- Autentizace mezi kolektorem a exportérem. Zde chceme zabránit útočníkovi číst citlivá data, případně odeslat data falešná.

Pro zajištění výše zmíněných bodů byly zvoleny dva prostředky, a to TLS [7] v případě, že je pro přenos flow dat zvolen TCP protokol, respektive DTLS [5] v případě, že je zvolen protokol UDP. Druhý zmíněný způsob však není příliš doporučován, jelikož bylo prokázáno, že bezpečnost v tomto případě není dostatečná [6, sekce 11.1].

Dalším rozšířením oproti NetFlow v9 je podpora vlastních polí v šabloně a polí s proměnnou délkou. Zatímco v NetFlow v9 jsme měli k dispozici pouze pevně danou množinu možných polí (zdrojová adresa, zdrojový port atd.), z nichž každé pole mělo přesně danou délku a sémantický význam, nyní jsme schopni si definovat pole vlastní, a to i pole proměnné délky. Více se lze o protokolu IPFIX dočíst v RFC 5101 [6].

2.3 Shrnutí

V této kapitole jsme si odpověděli na otázku, co rozumíme pod pojmem síťový tok. Dále jsme si stanovili vlastnosti, které musí splňovat metrika pro popis tohoto toku, aby byla užitečná koncovému uživateli a poskytla mu veškeré potřebné informace.

Seznámili jsme se způsoby, jakými je síťový tok zaznamenáván, včetně některých problémů, se kterými se můžeme setkat v souvislosti se sběrem dat, jež ho popisují.

V dalších kapitolách na toto navážeme a seznámíme se s aplikacemi pro vyhodnocení získaných informací. Více se také zaměříme na způsob uchování získaných dat.

Z protokolů pro popis síťového toku jsme se zaměřili převážně na NetFlow a IPFIX z důvodu dalšího směřování této práce. Nicméně vzhledem k tomu, že dané protokoly jsou pouze implementací obecného RFC 2722 [2], lze oprávněně předpokládat, že podstatná část implementace se nebude ve výrazných rysech lišit.

Kapitola 3

Analýza existujících nástrojů

V předchozích kapitolách jsme se se zaměřovali převážně na teoretickou stránku problematiky sběru a analýzy dat popisujících síťový tok. V této části se zaměříme na vyhodnocení nasbíraných údajů a představíme si nástroje, které s nasbíranými údaji dále pracují.

Budeme se snažit najít zejména takové nástroje, které by nám mohly pomoci v naší vlastní práci. Důraz budeme klást na vhodnou licenci a připravenost pro použití v IPv6 prostředí. Zvláštní důraz bude dále kladen na způsob uložení dat, škálovatelnost systému a jeho případnou rozšiřitelnost.

Jelikož jedním z největších problémů při práci s flow daty je jejich množství, je naší snahou, aby byla data ukládána úsporně, zároveň však, aby přístup k těmto datům byl pokud možno co nejrychlejší.

Při porovnávání následujících programů si tedy budeme pokládat převážně následující otázky:

- pod jakou licencí je program šířen,
- jaké podporuje operační systémy,
- jaké podporuje protokoly
- jak složité je systém upravit a integrovat do stávající infrastruktury
- jak je daný program výkonný a škálovatelný
- jakým způsobem program pracuje a uchovává flow data.

3.1 Program nfdump

Prvním nástrojem, kterému se budeme věnovat, je nfdump, který náleží do stejně pojmenovaného projektu [1]. Cílem tohoto projektu je vyvinout sadu open-source nástrojů pro práci s NetFlow daty. Nástroje jsou šířeny pod licencí BSD a umí pracovat s daty ve formátu NetFlow ve verzích 5, 7 a 9.

V současné chvíli bohužel nepodporují protokol IPFIX, nicméně na toto téma existují odborné práce, které se zabývají implementací této chybějící podpory. Za všechny jmenujme: Network Traffic Collection with IPFIX Protocol[11].

[1] Domovská stránka projektu je http://nfdump.sourceforge.net

Jak jsme si řekli, projekt nfdump je kolekcí utilit, z nichž každá má přesně dané určení. Tyto utility si popíšeme v následujícím seznamu:

nfcapd - NetFlow capture daemon: čte informace o síťovém toku ze sítě a zaznamenává je pro pozdější použití. Nfcapd tedy plní funkci kolektoru, který ukládá data ze sběrných míst v jednoduchém binárním formátu. Tento formát je vhodný pro jednoduchý sekvenční průchod. Indexace obsahu není podporována. Pro zvýšení výkonu je možné rozdělit soubory podle času. Výhodou řešení je značná jednoduchost implementace, administrace, provádění záloh a vysoký výkon při ukládání dat. Nevýhodou je pomalé filtrování a špatná možnost škálování (například rozložením zátěže na více serverů). Další značnou nevýhodou je nemožnost snadným způsobem rozšířit tento formát o další pole, která bychom chtěli v naší aplikaci využívat a zpracovávat.

nfdump – NetFlow dump: Čte soubory vytvořené utilitou nfcapd, umožňuje jejich filtrování a zobrazování statistiky. Jedná se tedy o samotnou aplikaci, která koncovému uživateli podává informace o síťovém toku jako takovém.

nfprofile – NetFlow profiler: Čte soubory vytvořené utilitou nfcapd a filtruje je dle daných profilů a ukládá pro další použití.

nfreplay – NetFlow replay: Čte soubory vytvořené utilitou nfcapd a získává z nich souhrnné informace. Přitom je možné provádět i filtraci. Podoba tohoto filtru je stejná jako u utility nfdump.

ft2nfdump: Konvertuje data z formátu "flow-tools"do formátu zpracovatelného nástrojem nfdump.

Nfdump je v tuto chvíli používaný v síti VUT Brno. Sám o sobě neplní úlohu analytické aplikace, jeho rozhraní je tvořeno čistě příkazovou řádkou, díky čemuž je možné jej snadno volat z jiných aplikací nebo kombinovat s jinými programy. Staví na něm projekty jako například NfSen 3.2.

Jelikož program nfdump budeme dále intenzivně využívat, seznámíme se s ním podrobněji. Jak již bylo napsáno, program se ovládá pouze z příkazové řádky. Data zpracovává dávkově, po načtení vstupu tedy program začne pracovat a není možné žádným způsobem jeho výsledek ovlivnit.

Přepínače, které je možno programu předat, lze rozdělit do několika skupin. Nejprve se seznámíme s přepínači, které určují jaké soubory se budou zpracovávat:

-r: Umožňuje zpracovat jeden konkrétní soubor, předáme jméno tohoto souboru přepínačem -r. Chceme-li tedy zpracovat jediný soubor /home/1/20111130.nfcapd můžeme tak učinit následujícím voláním:

```
nfdump -r /home/1/20111130.nfcapd
```

-R: Chceme-li zpracovat více souborů, předáme je přepínačem -R, za nímž následuje úplná cesta k prvnímu souboru, a název posledního souboru. Za předpokladu, že bychom tedy potřebovali zpracovat soubory 20111130.nfcapd až 20111230.nfcapd seřazené vzestupně a umístěné ve složce /home/1, spustíme nfdump takto:

```
nfdump -r /home/1/20111130.nfcapd:20111230.nfcapd
```

-M: Chceme-li číst jeden stejně nazvaný soubor z více adresářů, využijeme přepínač -M v kombinaci s dříve popsaným přepínačem -r. Potřebujeme-li například číst soubor 20111230.nfcapd v adresářích /home/1, /home/2, /home/3, ..., /home/10, provedeme to voláním:

```
nfdump -M /home/1:10 -r 20111230.nfcapd
```

Pokud nechceme zpracovávat všechny adresáře v dané sekvenci, ale pouze jejich výčet, můžeme využít variantu příkazu, kdy jsou jednotlivé zpracovávané adresáře oddělené dvojtečkou, jak uvádí následující příklad:

```
nfdump -M /home/1:5:10 -r 20111230.nfcapd
```

Takto spuštěný příkaz bude číst soubor 20111230.nfcapd postupně v adresářích /home/1, /home/5 a /home/10. Kromě varianty s *-r* je možno použít i variantu s *-R*, kdy program čte postupně sekvenci souborů z výčtu daných adresářů. Pokud bychom například potřebovali zpracovat posloupnost souborů /home/1/1.nfcapd, /home/1/2.nfcapd, /home/2/1.nfcapd a /home/2/2.nfcapd, můžeme tak provést následujícím příkazem:

```
nfdump -M /home/1:2 -R 1.nfcapd:2.nfcapd
```

Není-li specifikován žádný soubor, je čten a zpracováván standardní vstup. To může být užitečné v případě, kdy chceme nfdump použít jako filtr.

Další důležitý parametr je přepínač pro formátování výstupu *-o* následovaný jednou z těchto možností:

raw: Vypisuje všechny známé údaje pro daný záznam na více řádků. Pro další zpracování se tento formát příliš nehodí. Výpis při použití parametru *raw* bude vypadat následujícím způsobem:

```
Flow Record:
  Flags        =              0x01 Unsampled
  size         =               176
  first        =        1335002725 [2012-04-21 12:05:25]
  last         =        1335002725 [2012-04-21 12:05:25]
  msec_first   =               870
  msec_last    =               870
  src addr     =  2001:67a:1220:c1a1:95bc:9597:68a8:74da
```

```
dst addr     =  2001:0:5cf5:79fb:cc:e09:abfd:9b06
src port     =                    0
dst port     =                33024
fwd status   =                    0
tcp flags    =                 0x00 ......
proto        =                   58
(src)tos     =                    0
(in)packets  =                    1
(in)bytes    =                   52
input        =                    3
output       =                    2
src as       =               2481312910
dst as       =               1409443065
out packets  =                    1
out bytes    =                   52
ip router    =        192.255.71.11
engine type  =                    0
engine ID    =                    0
in src mac   = 00:00:00:00:00:00
out dst mac  = 00:00:00:00:00:00
in dst mac   = 00:00:00:00:00:00
out src mac  = 00:00:00:00:00:00
MPLS Lbl  1  =               0-0-0
MPLS Lbl  2  =               0-0-0
MPLS Lbl  3  =               0-0-0
MPLS Lbl  4  =               0-0-0
MPLS Lbl  5  =               0-0-0
MPLS Lbl  6  =               0-0-0
MPLS Lbl  7  =               0-0-0
MPLS Lbl  8  =               0-0-0
MPLS Lbl  9  =               0-0-0
MPLS Lbl 10  =               0-0-0
```

line: Vypisuje základní údaje o daném toku. Jedná se o čas začátku a trvání přenosu v milisekundách, protokol, zdrojovou adresu v kombinaci s portem, cílovou adresu v kombinaci s portem, počet paketů a počet bytů. Dále je zde uveden údaj o počtu agregovaných záznamů do tohoto jednoho. Pokud není výsledek agregován, je zde uvedena hodnota 1. Výpis při použití parametru *line* bude vypadat následujícím způsobem:

```
Date flow start          Duration Proto     Src IP Addr:Port
2012-04-21 12:05:25.419     0.231 TCP       197.229.217.20:53729 ->

Dst IP Addr:Port   Packets    Bytes Flows
29.21.111.139:80           4      620     1
```

long: Stejné údaje jako u varianty line, pouze přidává navíc Tos a TCP příznaky. Výpis při použití parametru *long* bude vypadat následujícím způsobem:

```
Date flow start          Duration Proto  Src IP Addr:Port
2012-04-21 12:05:24.116     0.000 TCP    192.50.135.1:80 ->

Dst IP Addr:Port   Flags Tos  Packets    Bytes Flows
197.229.151.75:14043 .A..S.   0        1       44     1
```

extended: Stejné jako long. Opět přidává další údaje. Konkrétně jsou to dopočítané hodnoty bity za sekundu, pakety za sekundu a byty na paket. Příklad v tomto případě uvádět nebudeme a necháme laskavého čtenáře, ať si jej domyslí.

Kromě výše zmíněných je možné si vytvořit i vlastní formát pomocí příkazu:

```
-o fmt: ...
```

za *fmt:* následuje výčet všech sloupců, jenž chceme mít ve výsledku. Seznam přípustných hodnot shrnuje tabulka 3.1.

Za normálních okolností budou IPv6 adresy vypsány ve zkrácené podobě například takto:

```
2005:62..2c:9c10
```

Pokud chceme vypsat IPv6 adresu nezkrácenou, můžeme přidat přepínač *-6*. Následně již uvidíme celou adresu:

```
2005:620:0:8:203:baff:fe2c:9c10
```

V některých případech potřebujeme informaci o celkovém počtu přenesených dat mezi dvojicí IP adres, IP adresy a podsítě nebo dvou podsítí. K tomu slouží *agregace*.

Agregace se v programu nfdump provádí přepínačem *-a*, případně *-a -A*. U první jmenované varianty jsou agregovány záznamy se stejným protokolem, zdrojovou adresou, zdrojovým portem, cílovou adresou a cílovým portem. V případě druhé varianty je možné druh agregace upřesnit zadáním libovolné kombinace příznaků uvedených v tabulce 3.2 oddělených čárkou [2].

Pokud tedy chceme agregovat podle protokolu, zdrojové a cílové IP adresy (bez ohledu na port), provedeme to voláním:

```
nfudmp -a -A proto,srcip, dstip
```

Pokud nechceme agregovat podle konkrétní IP adresy, použijeme mírně upravenou variantu. V prvé řadě je nutno uvést verzi IP protokolu u příznaků srcip,dstip 4 v případě IPv4 nebo 6 v případě IPv6. Následovat bude lomítko a délka masky podsítě. Tedy například:

```
nfudmp -a -A srcip, dstip4/24
```

Často také stojíme před úkolem zobrazit pouze záznamy, jež splňují konkrétní kritéria. K tomu v prostředí utility nfdump využíváme takzvané filtry, které se zadávají jako poslední parametr příkazové řádky.

[2] Oficiální manuál v tomto bohužel není úplný, ucelenější seznam všech příznaků jsme nalezli v manuálové stránce http://www.linuxcertif.com/man/1/nfdump/

Značka	význam
%ts	Začátek přenosu
%te	Konec přenosu
%td	Trvání přenosu (%te - %ts)
%pr	Protokol
%sa	Zdrojová adresa
%da	Cílová adresa
%sp	Zdrojový port
%dp	Cílový port
%sap	Zdrojová adresa v kombinaci se zdrojovým portem
%dap	Cílová adresa v kombinaci s cílovým portem
%sap	Cílová adresa v kombinaci s cílovým portem
%dap	Cílová adresa v kombinaci s cílovým portem
%sas	Číslo zdrojového autonomního systému
%das	Číslo cílového autonomního systému
%in	Číslo vstupního rozhraní
%out	Číslo výstupního rozhraní
%pkt	Počet paketů
%byt	Počet bytů
%fl	Počet agregovaných záznamů
%flg	TCP flagy
%tos	ToS údaj z IPv4 hlavičky
%bps	bitů za sekundu
%pps	paketů za sekundu
%bps	bytů na paket
%mpls1 až %mpls10	MPLS label

Tabulka 3.1: Seznam značek pro uživatelem definovaný výstupní formát.

Filtrovat je možné podle několika kriterií. Nejdůležitější pro nás budou tato:

Protokol: TCP, UDP, ICMP, ale i některé další.

Verze protokolu: inet a ipv4 pro IPv4, inet6 a IPv6 pro IPv6.d

IP adresa: například: *IP 192.168.0.1* tomuto může předcházet příznak SRC,DST, SRC and DST nebo SRC or DST pro určení, zda se jedná o zdrojovou, případně cílovou adresu. V případě, že není uvedeno, předpokládá se SRC or DST.

Síť: například: *NET 192.168.0.1 255.255.255.0* nebo *NET 192.168.0.1 /24* stejně jako v případě IP adresy je možné použít příznak SRC,DST, SRC and DST nebo SRC or DST. Pokud bychom tak chtěli vyfiltrovat veškerý provoz pocházející ze sítě *NET 192.168.0.1/24*, použili bychom variantu *SRC NET 192.168.0.1/24*

Toto je výčet asi nejdůležitějších parametrů, podle kterých je možno filtrování provádět. Ucelenější seznam lze najít na stránkách projektu [3] . Všechny filtrovací výrazy je navíc

[3] http://nfdump.sourceforge.net/

Značka	význam
proto	IP protokol
srcip	Zdrojová IP adresa
dstip	Cílová IP adresa
srcip4/net	Zdrojová IPv4 adresa s maskou podsítě
srcip6/net	Zdrojová IPv6 adresa s maskou podsítě
dstip4/net	Cílová IPv4 adresa s maskou podsítě
dstip6/net	Cílová IPv6 adresa s maskou podsítě
srcport	Zdrojový port
dstport	Cílový port

Tabulka 3.2: Seznam nejdůležitějších atributů pro agregaci NetFlow dat.

možno spojovat operátorem *and* a *or*. Dále je možno použít závorky pro úpravu priority, případně operátor *not* pro negaci následujícího výrazu.

Posledním parametrem přepínačem, který si představíme, je přepínač pro omezení časového rozmezí, ve kterém se mají nacházet požadované záznamy. Tímto přepínačem je *-t*. Po tomto přepínači následuje čas začátku filtrování ve formátu yyyy/MM/dd.hh:mm:ss[4], případně je možno definovat i čas konce ve stejném formátu.

V takovém případě má příznak podobu: yyyy/MM/dd.hh:mm:ss-yyyy/MM/dd.hh:mm:ss.

Ještě stojí za to poznamenat, že v naší práci jsme nevyužili originální verzi nfdumpu dostupnou ze stránek projektu, ale mírně upravenou verzi vyvíjenou na Fakultě informačních technologií VUT v Brně. Tomuto se však budeme více věnovat v kapitole 5.

3.2 Program NfSen

Zatímco nfdump plní primárně funkci kolektoru NetFlow dat, NfSen [5] již lze považovat za plně analytickou aplikaci. Jedná se o projekt vyvíjený pod BSD licencí, který je s nfdumpem úzce svázán, jelikož jej používá jako svůj backend. V tuto chvíli je aktuální verze 1.3.6. Skládá se ze dvou částí. První je démon, který průběžně generuje statistiky. Tato část je psaná v Perlu. Druhá část je psaná v PHP a jedná se o webový frontend pro procházení těchto statistik. Jak již bylo napsáno dříve, statistiky jsou počítány průběžně. Pokud tedy jsme schopni definovat dopředu, jaká data nás zajímají, je možné dosahovat poměrně dobrých výsledků. V opačném případě je však práce velice pomalá, a to z toho důvodu, že všechna data jsou zpracovávána sekvenčně, což je základní omezení nfdumpu. Rozšiřitelnost nástroje je poměrně špatná, jelikož se jedná o kombinaci několika technologií a programovacích jazyků.

Konkrétní systémové nároky jsou následující:

OS: Doporučuje se *NIXový operační systém

PHP: minimálně ve verzi 4.1 s rozšířením pro regulární výrazy kompatibilní s Perlem a podporou socketů.

Perl: minimálně ve verzi 5.6.0, s moduly Mail::Header, Mail::Internet, RRDTool

[4] rok/měsíc/den.hodina.minuta.vteřina

[5] Domovská stránka projektu http://sourceforge.net/projects/nfsen/

RRD: balík nástrojů RRD tools [6]

nfdump: minimálně ve verzi 1.5.8

Použití takto pestré palety nástrojů ztěžuje vývojářům možnost upravovat tento produkt, jelikož by měli mít alespoň nějaké povědomí o všech použitých technologiích. Navíc kvalita kódu (přestože je to pojem poměrně subjektivní) není příliš vysoká, o čemž vypovídá kompatibilita s PHP ve verzi 4.1, kde byla velmi malá, až téměř žádná podpora pro objektové programování. Ve verzi 5, která vyšla v roce 2004, byl objektový model zcela přepracován. Navíc, jak si ukážeme dále, s novější verzí PHP začíná aplikace generovat nemalé množství chyb.

V našem případě bohužel NfSen nebyl v žádném repositáři softwaru pro náš operační systém, proto jsme jej museli nainstalovat ručně následujícím způsobem (návod pro *nixový operační systém):

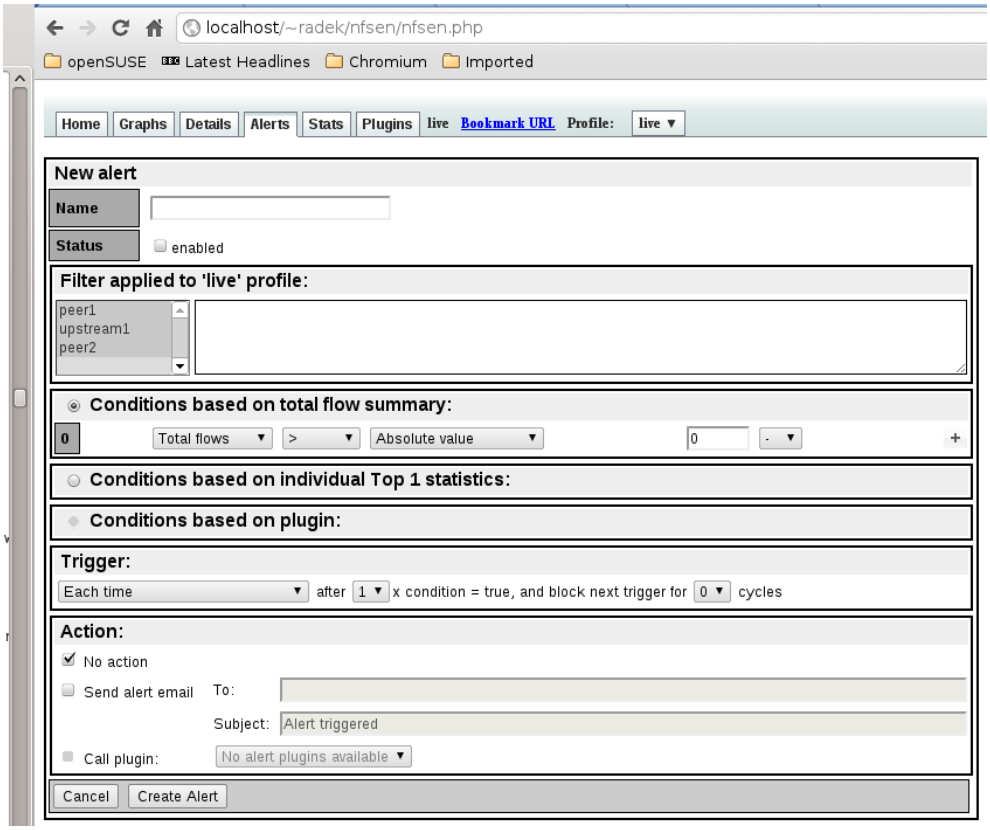
1. Ze stránek projektu jsme stáhli potřebné zdrojové soubory a umístili je do složky /home/radek/nfsen.
2. Vytvořili jsme složku /home/radek/pubic_html/nfsen (Apache server má zaveden modul mod_userdir).
3. Následně jsme vytvořili uživatele a skupinu s názvem nfsen, uživatele nfsen jsme přidali do stejně nazvané skupiny.
4. Ve složce /home/radek/nfsen/etc/ najdeme soubor nfsen-dist.conf, přejmenujeme jej na nfsen.conf a zeditujeme podle následujícího klíče:
   (a) $BASEDIR=”/home/radek/nfsen”
   (b) $HTMLDIR=”/home/radek/pubic_html/nfsen”
   (c) $PREFIX=”/usr/local/bin”
   (d) $USER=”nfsen”
   (e) $WWWUSER=”nfsen”
   (f) $WWWGROUP=”nfsen”
5. Následně vejdeme do složky /home/radek/nfsen a spustíme příkaz ./install.pl etc/nfsen.conf
6. Nyní můžeme spustit vlastní NfSen příkazem /home/radek/nfsen/bin/nfsen start

V tuto chvíli bychom měli vidět na adrese *http://localhost/~radek/nfsen/nfsen.php* fungující NfSen. Pro lepší představu přikládámé obrázek 3.1

Jak jsme se již dříve zmínili, po instalaci jsme byli zaskočení poměrně velkým množstvím chyb, které tato aplikace generuje. Většinou se naštěstí jedná pouze o varování typu notice, tedy pouze o upozornění ve smyslu čtení neinicializované proměnné. Tyto chyby opravíme většinou příkazem:

```
if( !isset($variable) ){
  $variable = null;
}
```

[6] http://oss.oetiker.ch/rrdtool/



Obrázek 3.1: Na obrázku se nachází ukázka uživatelského rozhraní programu nfsen.

Druhou variantou je snížení úrovně chybových hlášení generovaných PHP interpretem. Toho docílíme nastavením konfigurační direktivy:

```
error_reporting = E_ALL ^ E_NOTICE
```

, čím ale problém pouze schováme.

Při testování jsme narazili na několik nedostatků. Systém v základní instalaci není chráněn žádným jménem ani heslem. Řešení, jak se tohoto problému zbavit, je dvojí:

- Implementace vlastní přihlašovací stránky a editace zdrojového kódu.
- Implementace zabezpečení na úrovni webového serveru klientským certifikátem, nebo HTTP(S) autentizací.

Žádné z daných řešení však nepovažujeme za dostatečné, jelikož jediný problém, který tímto vyřešíme, je autentizace. Dalším bodem by měla být autorizace, kdy ověřujeme, že konkrétní uživatel má právo disponovat konkrétním prostředkem. Toho však s tímto nástrojem nejsme schopni dosáhnout.

Další nedostatek, který vyplývá z chybějící podpory více uživatelů je, že systém nezaznamenává žádné údaje, které by v případě bezpečnostního incidentu poskytly informaci o tom, který uživatel se systémem prováděl jakou operaci.

Při hlubší analýze jsme zjistili, že systém je náchylný na útoky typu CSRF, jež jsou více popsány v 6.5.

Celkově lze tento systém označit jako funkční s výhradami. Zaměřuje se převážně na souhrnné statistiky, kdy uživateli poskytne kupříkladu informaci o celkovém objemu přenesených dat v síti. Na tyto statistiky je možno navázat událost, která se spustí například ve chvíli, kdy se objem přenesených dat skokově například zdvojnásobí. V takovém případě může systém upozornit administrátora například e-mailem.

V tomto je systém poměrně značně propracovaný a umožňuje velké možnosti nastavení. Na druhou stranu systém neposkytuje žádnou podporu pro dotazy, kdy potřebujeme vybrat množinu záznamů, které odpovídají některému z kriterií.

Navíc je celý systém naprosto nezabezpečený, což považujeme za velkou slabinu u libovolného webového projektu. Zcela jistě lze argumentovat tím, že systém bude přístupný pouze z intranetu nebo konkrétní IP adresy. Nicméně tento postup by dle našeho názoru měl být pouze doplňkem k běžným zabezpečovacím procedurám. V opačném případě se jedná o takzvané *Security through obscurity*[7]. Tento princip je obecně považován za špatný, jelikož poskytuje pouze iluzi bezpečnosti. Například pokud bychom zpřístupnili fungující instanci NfSenu pro administrátora, který se bude připojovat z konkrétní IP, je neustále možné napadnout systém odesláním podvržené stránky právě administrátorovi. Daná stránka pak vykoná všechny potřebné úkony bez vědomí napadeného administrátora.

3.3 Program Ntop

Ntop[8] je další z aplikací pro práci s daty popisujícími síťový tok. V době psaní této práce byla aktuální verze 4.1, jež vyšla 15.8.2010. Na rozdíl od aplikace NfSen, která plní roli čistě analytickou, Ntop plní i roli kolektoru, přestože data neukládá, pouze nad nimi počítá statistiky. To by se dalo chápat jako nevýhoda, na druhou stranu, záleží na druhu použití. V případě, že nemáme potřebu poskytovat uživateli historická data, je i zbytečné je ukládat.

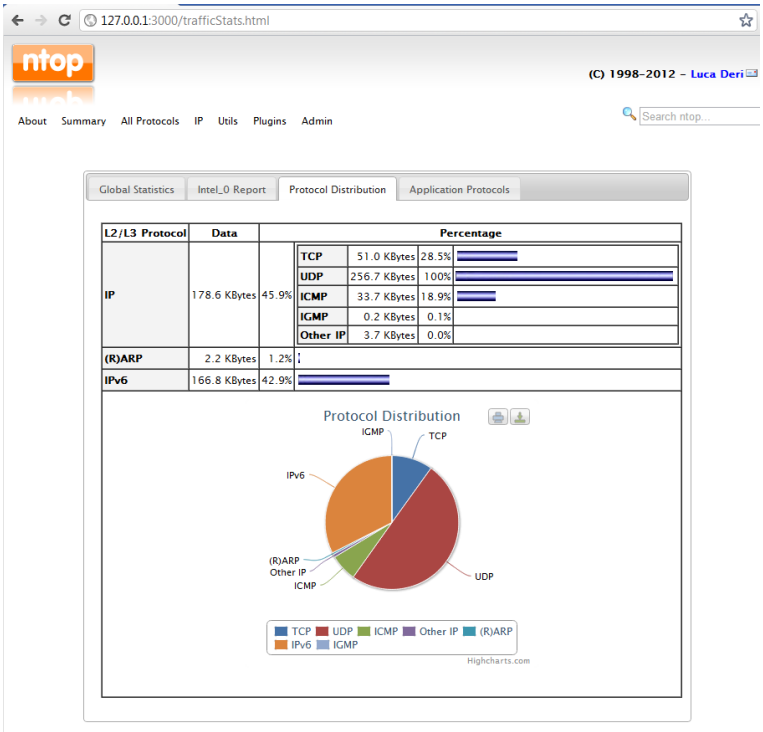
V aktuální verzi podporuje tato aplikace jak protokoly NetFlow a IPFIX, tak sFlow. Obsahuje vlastní vestavěný webový server, s jehož pomocí jsou prezentovány veškeré statistiky uživateli. Ukázku je možné si prohlédnout na obrázku 3.2. Celá webová aplikace je napsána v programovacím jazyku *C*. To na jednu stranu umožňuje velmi těsnou kooperaci modulu pro prezentaci dat s modulem pro jejich zpracování, který je také napsán v jazyce *C*. Na druhou stranu to znesnadňuje snadnou úpravu prezentační vrstvy případně další přizpůsobení aplikace našim potřebám. Možností, jak tento nedostatek obejít, je využití REST[9] rozhraní, které ntop nabízí. Díky tomuto rozhraní je možné napojit na ntop další aplikace, které můžou lépe odpovídat našim potřebám.

Oblast, ve které je Ntop vyzrálejší než NfSen, je jednoznačně Autorizace a řízení přístupu. Do systému je možno zanést libovolné množství uživatelů a těm následně přidělit práva ke zdrojům. Pod zdrojem se zde rozumí URL adresa, kdy uživateli nemusíme dávat právo na konkrétní URL, ale můžeme použít metaznaku *, jenž zastupuje libovolné množství libovolných dalších znaků.

Přestože tento systém není dokonalý, jelikož neumožňuje zachytit dědičnost a další pokročilé vlastnosti řízení přístupu, je možné jej považovat za dostatečný, jelikož s vynaložením jistého úsilí jsme schopni s ním dosáhnou všech možností nastavení.

Potěšující fakt je i ten, že ntop je schopen generovat záznamy o tom, kdo, kdy, z jaké IP provedl jaký požadavek na tuto aplikaci. Užitečnost těchto údajů bychom ocenili v případě

[7] bezpečnost skrze utajení
[8] http://www.ntop.org/products/ntop/
[9] Representational state transfer



Obrázek 3.2: Ukázka uživatelského rozhraní aplikace ntop.

bezpečnostního incidentu, kdy jsme schopni snáze dohledat útočníka, případně jsme schopni prokázat, že si daný uživatel vyžádal konkrétní data.

Pro svou práci využívá stejně jako NfSen utilitu RRD tools pro vykreslování průběžných grafů.

Systémové požadavky na tuto aplikaci jsou dle dokumentace následující:

OS: Unix, Linux, Solaris, MacOS X, *BSD, Windows 95 a novější

RAM: 100Mb pro WAN síť

3.4 Program SiLK

Bezplatně dostupná skupina aplikací, jež je šířená pod GNU licencí [10]. Aplikace jsou psány kombinací C, Perlu a Pythonu a měly by bez větších problémů fungovat na většině Unixových operačních systémů. Jsou ovládány z příkazové řádky a princip použití je podobný jako u utility nfdump.

Nejdůležitější utility z tohoto projektu:

FloCap: Tato utilita plní funkci kolektoru.

rwfilter: Čte data a provádí na nich filtrování času, protokolu a jiných atributů.

rwstats: Počítá statistiky ze vstupních dat, například celkový počet přenesených paketů a jejich distribuci mezi protokoly.

[10] Internetová adresa stránek projektu je http://tools.netsa.cert.org/silk/

rwcount: Počítá některé statistiky agregované dle časových bloků.

rwcut: Čte a formátuje do čitelné podoby binární podobu dat o síťovích tocích, kterou interně používají ostatní utility.

rwsort: Řadí údaje o síťovém toku dle parametrů na vstupu.

rwuniq: Utilita pro počítání výsledků agregovaných dle jednotlivých atributů.

Utility je možné v prostředí příkazové řádky řetězit tak, jak je běžně zvykem.

3.5 Další aplikace

Při naší práci jsme narazili na mnoho projektů, jež si kladly za cíl nějakým způsobem pracovat s daty pro popis síťového toku. Bohužel, značná část z nich je již delší dobu neudržovaná. S ohledem na rostoucí potřebu zpracovávat IPv6 data jsme se rozhodli vyřadit ty projekty, které tuto podporu nedeklarovaly, nebo které byly delší čas neudržované a kde lze tedy oprávněně předpokládat, že tato podpora nebude dostatečná (uvědomíme-li si, že některá doporučení k IPv6 se v průběhu času mění).

Další skupinou aplikací jsou aplikace komerční, ke kterým jsme neměli přístup. Zde jsme vycházeli pouze z informací, jež se nám podařilo vyčíst z manuálů a stránek projektů.

PRTG Network Monitor: Komerční produkt[11], k dispozici je však i bezplatná verze. Aplikace je k dispozici pouze pro platformu Windows od verze XP. Plní funkci analytické aplikace a kolektoru. Je značně podobný projektu NfSen. Umožňuje však krom monitoringu sítě i monitoring serverů, služeb a dalších internetových aplikací.

Scrutinizer: Komerční aplikace[12] plnící funkci kolektoru. Dostupná je opět i v bezplatné variantě, která má však mnoho omezení (například ukládá pouze 1 poslední den). Analytický nástroj k tomuto se jmenuje flow Analytics. Tato aplikace je stejně jako PRTG Network monitor dostupná pouze pro Windows.

Softflowd Aplikace[13] plnící funkci monitoru dostupná pod New BSD License. S kolektorem komunikuje pomocí NetFlow protokolu, který podporuje ve verzích 1, 5 a 9. Je kompatibilní s IPv6. Zároveň integruje některé funkce analytických aplikací, jelikož umožňuje průběžné získávat některé základní statistické informace, například počet přenesených paketů a podobně. Množství a možnosti poskytovaných funkcí však nejsou nikterak rozsáhlé.

3.6 Shrnutí

V této kapitole jsme se pokusili seznámit s již existujícími nástroji pro práci s daty, která popisují síťový tok.

Zaměřili jsme se převážně na oblast analytických aplikací a zvýšenou pozornost jsme zde věnovali aplikacím, jež pracují s protokoly NetFlow a IPFIX. Tato analýza nám měla umožnit lépe pochopit potřeby uživatelů a inspirovat se při tvorbě naší vlastní aplikace.

[11] Stránky produktu http://www.paessler.com/prtg
[12] Stránky produktu http://www.plixer.com
[13] Stránky produktu http://www.mindrot.org/projects/softflowd/

Přestože jsme čekali, že nabídka již hotových nástrojů bude širší, nemyslíme si, že by současný stav byl nedostatečný. Existující nástroje své poslání plní a je zde i jistá možnost výběru.

Nicméně i zde existují oblasti, kde by bylo možno dosáhnout jistého pokroku. Nemyslíme si, že oblastí, kterou bychom se měli zabývat, jsou agregované dotazy a statistika nad nasbíranými údaji, jelikož tato oblast je již poměrně dobře pokrytá existujícími nástroji a uživatel zde může najít aplikace, jež mu pomohou dosáhnout svých cílů. Více se zaměříme na oblast, jež zatím pokrytá není.

Touto oblastí je zodpovídání konkrétnějších neagregovaných dotazů, export dat a jejich transformace pro snadnou integraci dalších aplikací. V neposlední řadě pak podpora práce více uživatelů a správa privilegií. Toto tedy budou hlavní body, na které se naše aplikace zaměří a v nichž by měla doplnit portfolio již existujících nástrojů.

Kapitola 4

Ukládání dat o síťovém toku

Cílem následující kapitoly je prostudovat možnosti ukládání dat popisujících strukturu síťového toku. Jednat se nám bude zejména o NetFlow data, nicméně většinu závěrů lze zobecnit i na IPFIX data, případně libovolná jiná data, která potřebujeme logovat, a která mají pevnou strukturu. Charakter logovacích dat napovídá, že našim záměrem je najít systém, který bude dostatečně výkonný pro ukládání dat a jejich následné čtení. Úprava dat je operace, ke které docházet nebude. K mazání bude docházet také jen velice zřídka a bude se jednat vždy o odmazání nejstarších dat, technicky se tedy bude jednat svým druhem o RRD[1] databázi.

Ukládání flow dat a obecně všech logovacích údajů má svá specifika. Jak jsme psali dříve, formát dat je dopředu znám a mění se jen zřídka. Nedochází zde k úpravě již uložených dat a mazána jsou vždy nejstarší data. Naproti tomu množství ukládaných dat je značné.

My si jako základní a referenční produkt zvolíme nfdump, což je sada utilit, která obsahuje jak kolektor, tak nástroj pro dotazování se nad uloženými údaji. Našim cílem je najít nástroj, který by při dotazování dosahoval srovnatelných nebo lepších výsledků jak v čase dotazování, tak v komfortu při kladení dotazů. Přitom se primárně zaměříme na již existující databázové systémy.

Od tohoto řešení si slibujeme možnost využití pokročilých možností databázových systémů jako je schopnost opatřit data indexem a tím urychlit jejich prohledávání, replikace nebo možnost rozdělit jednu databázovou tabulku přes více disků, takzvaný *partitioning*. Těmito a dalšími technikami bychom pak mohli zvětšit výkon našeho systému a získat více komfortu pro koncového uživatele.

Typické dotazy, které chceme provádět nad NetFlow daty, jsou následující:

1. **Základní dotaz na komunikaci IP adresy.** Následující dotaz zobrazí komunikaci IP adresy 2001:67c:1220:e000::93e5:30a.

   ```
   nfdump "host 2001:67c:1220:e000::93e5:30a"
   ```

2. **Filtrace dle prefixu.** Následující dotaz zobrazí komunikaci, kde zdrojová nebo cílová IP adresa náleží do podsítě 2001:67c:1220:e000::/56.

   ```
   nfdump "net 2001:67c:1220:e000::/56"
   ```

[1] RRD = Round-Robin Database

3. **Filtrace prefixu + agregace podle zdrojové adresy.** Následující dotaz zobrazí komunikaci, kde zdrojová nebo cílová IP adresa náleží do podsítě 2001:67c:1220:e000::/56. Výsledek je následně agregován dle zdrojové IP adresy.

```
nfdump -a -A srcip "net 2001:67c:1220:e000::/56"
```

4. **Komunikace z vybraného prefixu do vybrané cílové sítě.** Následující dotaz zobrazí komunikaci, kde zdrojová IP adresa náleží do podsítě 2001:67c:1220:e000::/56 a cílová adresa náleží do podsítě 2001:718::/32. Výsledek je následně agregován dle zdrojové a cílové IP adresy.

```
nfdump -a -A srcip,dstip  "src net 2001:67c:1220:e000::/56
and dst net 2001:718::/32"
```

5. **Komunikace vybraných adres na cílový port 80 (http).** Následující dotaz zobrazí komunikaci, kde zdrojová IP adresa náleží do podsítě 2001:67c:1220:e000::/56 a cílový port má hodnotu 80. Výsledek je následně agregován dle zdrojové IP adresy.

```
nfdump -a -A srcip  "src net 2001:67c:1220:e000::/56
and dst port 80"
```

6. **Struktura protokolu na jednom rozhraní.** Následující dotaz zobrazí komunikaci na síťovém rozhraní s ID 1. Výsledek je následně agregován dle použitého protokolu.

```
nfdump -a -A proto  "in if 1"
```

7. **Objemy odchozích dat dle prefixu pro vybranou síť.** Následující dotaz zobrazí komunikaci, kde zdrojová IP adresa náleží do podsítě 2001:67c:1220::/48. Výsledek je následně agregován dle prefixu podsítě.

```
nfdump -a -A srcip6/64  "src net 2001:67c:1220::/48"
```

8. **Objemy odchozích dat dle prefixu pro vybranou síť a následné seřazení jednotlivých prefixů podle objemu přenesených dat.** Stejný dotaz jako v předchozím příkladu, výsledek je však seřazen dle objemu přenesených dat.

```
nfdump -a -A srcip6/64  -w -  "src net 2001:67c:1220::/48"
|  nfdump -n 0 -s srcip/bytes
```

Jako vhodné indexy proto zvolíme:

- začátek přenosu,

- konec přenosu,
- zdrojová IP v kombinaci se zdrojovým portem,
- cílová IP v kombinaci s cílovým portem.

Velice zajímavé se v tomto případě jeví využití NoSQL, případně Column oriented databází. Tyto databáze jsou sice ochuzeny o některé možnosti klasických SQL databází, výměnou za to je zvýšení výkonu při úkolech jiných. Jako představitele NoSQL databází zvolíme MongoDB a CouchDB jako dvě nejznámější.

Nevýhodou databázových systémů je na druhou stranu zvýšení nároků na administraci, zálohy a výpočetní výkon. Proto se budeme zabývat i databází SQLite a FastBit, které nemají vlastní server, což je z hlediska administrace činí velmi jednoduchými.

4.1 Relační databáze

Relační databáze jsou v současnosti nejrozšířenějším typem všech používaných databází. Data jsou zde ukládána do tabulek s pevnou strukturou. Mezi nejznámější a nejrozšířenější zástupce patří MySQL, PostgreSQL, Oracle a MSSQL. Výhodou zmíněných systémů je velká řada nabízených funkcí, podpora partitioningu, referenční integrity a transakčního zpracování. Nutno však zdůraznit, že pro naše použití můžou být tyto funkce spíše na škodu, budou-li implementovány na úkor rychlosti jednoduchých dotazů.

4.1.1 Databáze MySQL

Jeden z nejrozšířenějších databázových serverů. Zajímavostí je, že pro ukládání dat je možné zvolit z celé řady dostupných backendů (MyISAM, InnoDB, Archive ...). Díky tomu našel uplatnění i při ukládání velkého množství dat a jejich indexace za využití speciálních backendů. Nicméně momentálně není v naších silách tohoto využít, a proto použijeme jeden ze standardních dodávaných, a to InnoDB. Verzi MySQL jsme použili 5.1.59.

Před samotným testováním jsme se také snažili upravit konfiguraci serveru, a to takovým způsobem, aby nám v našich podmínkách poskytla co možná největší výkon. Pro dosažení maximálního výkonu při zápisu velkého množství dat jsme zvolili následující postup. Netflow data jsme transformovali do podoby CSV[2] záznamu, který jsme pak do databáze nahrávali pomocí příkazu LOAD DATA INFILE[3], který nahraje data ze souboru přímo do cílové tabulky.

Dále jsme provedli nastavení těchto proměnných:

innodb_flush_log_at_trx_commit = 0: Touto volbou omezíme počet zápisů transakčního logu na disk. Nevýhodou je, že v případě havárie ztratíme některá data.

innodb_buffer_pool_size = 1G: Velikost jsme nastavili na 50% fyzické paměti.

innodb_log_file_size = 256M: Velikost jsme nastavili na 25% innodb_buffer_pool_size. Zvětšením této hodnoty dosáhneme zrychlení ukládání nových údajů, na druhou stranu se prodlouží doba obnovy po výpadku serveru.

[2] Comma-separated values neboli hodnoty oddělené čárkami
[3] http://dev.mysql.com/doc/refman/5.1/en/load-data.html

4.1.2 Databáze SQLite

Z hlediska administrace se jeví zajímavě využití databází bez vlastního serveru. Lze očekávat, že dané databáze budou pomalejší, jelikož nejsou schopny zachovat hodnoty dlouhodobě v operační paměti, na druhou stranu jejich použití a nasazení bývá mnohem jednodušší. Jako zástupce jsme zde zvolili SQLite ve verzi 3.7.5

I zde jsme se snažili dosáhnout maximálního výkonu, a to nastavením těchto proměnných:

PRAGMA synchronous = OFF: Touto volbou zrychlíme ukládání nových dat, v případě výpadku však můžeme ztratit poslední data, jež ještě nebyla fyzicky uložena na disku.

PRAGMA journal_mode=MEMORY: Touto volbou zrychlíme ukládání nových dat, v případě výpadku však můžeme ztratit poslední data, jež ještě nebyla fyzicky uložena na disku.

PRAGMA temp_store=MEMORY: Touto volbou zrychlíme některé operace za cenu větších paměťových nároků.

Více o daných proměnných lze najít přímo v dokumentaci projektu[4].

Následné nahrání dat jsme provedli pomocí příkazu import, který má podobnou funkci jako LOAD DATA INFILE v případě MySQL.

4.2 NoSQL databáze

NoSQL databáze, občas též dokumentové databáze, se vyznačují tím, že většinou nemají pevné schéma a nabídka datových typů je také omezenější. K datům se nepřistupuje pomocí jazyka SQL, ale každá databáze má většinou svůj vlastní způsob tvorby dotazů. Neposkytují obecně takovou množinu operací jako databáze relační, na druhou stranu se prezentují jako výkonnější, lépe škálovatelné a chybějící pevná struktura může být v některých případech také výhodou.

Jako zástupce těchto databází jsme si vybrali MongoDB a CouchDB.

4.2.1 Databáze MongoDB

Je dokumentová databáze napsaná v C++ pod licencí GNU AGPL v3.0, případně je možné koupit i komerční licenci. Konektory jsou pak pod svobodnější Apache License v2.0, což nám umožňuje tuto databázi poměrně snadno nasadit v komerční sféře. Dle manuálu je cílem databáze co možná nejvyšší rychlost a propustnost i za cenu menší stability. Dokumenty ukládá ve formátu BSON, což je binární serializace JSON[5] objektu. Má dobrou podporu indexace. Kromě základních indexů umožňuje tvořit i složené indexy a určovat jejich řazení. Do dokumentu je navíc možné vkládat i binární data.

Zde jsme se pouze ujistili o správném nastavení této proměnné:

journal = OFF: touto volbou zrychlíme ukládání dat, ale v případě výpadku prodloužíme čas nutný pro jejich obnovu.

MongoDB bohužel nemá žádnou alternativu pro dávkový import dat tak, jako SQLite nebo MySQL. V naší práci jsme používali verzi 1.8.4.

[4] http://www.sqlite.org/pragma.html
[5] Javascript object notation

4.2.2 Databáze CouchDB

Je zástupce key-value databází. Tato databáze je napsaná v jazyku Erlang a dle manuálu se snaží o maximální stabilitu. Každý dokument je automaticky identifikovatelný unikátní hodnotou, klíčem. Pokud tuto hodnotu neposkytneme my, systém ji vygeneruje za nás automaticky. Další vlastností, která stojí za zmínku, je fakt, že daná databáze komunikuje výhradně pomocí REST API. Veškeré operace, vkládání, úprava, mazání a selekce dat, probíhají pomocí HTTP protokolu. To umožňuje relativně snadné použití v libovolném programovacím jazyce. Nevýhodou však je o něco menší výkon způsobený další vrstvou abstrakce.

Konfigurační soubor této databáze neobsahuje příliš mnoho možností jak zvětšit výkon, neprovedli jsme proto žádné úpravy nad rámec základního nastavení.

Databáze poskytuje možnost provést dávkový import zasláním více dokumentů pro uložení v jednom HTTP požadavku, čehož jsme následně využili. V této práci byla použita verze 1.1.

4.2.3 Databáze FastBit

Technicky tato databáze spadá pravděpodobně pod NoSQL, přestože umožňuje pokládat i velice omezené SQL dotazy. Stejně jako SQLite nemá vlastní server. Má velmi dobrou podporu indexace pomocí bitmapových indexů a díky možnosti rozložit velká data do více adresářů je pro nás poměrně zajímavá. Navíc pro ni existuje podpora v projektu nmap a nprobe, což značí, že pro danou úlohu by mohla být použitelná.

Na rozdíl od ostatních databází, v této nebylo možné upravit žádnou konfigurační direktivu pro zrychlení ukládání dat. Obsahuje však mnoho možností jak ovlivnit druh vytvořeného indexu. Pro naše testování jsme zvolili verzi 1.2.4.

4.2.4 Srovnání databází

Pro srovnání jednotlivých databází byl použit testovací stroj:

Operační systém	Linux 2.6.38-gentoo-r4
CPU	x86_64 Intel(R) Atom(TM) CPU D525 @ 1.80GHz
RAM	2Gb
HD	3 * WESTERN DIGITAL Caviar Green 2000GB 64MB cache s Advanced Format, 5400 - 7200 RPM (IntelliPower) v Raid 5

Tabulka 4.1: Konfigurace testovacího serveru.

Test byl proveden s devadesáti minutami anonymizovaných NetFlow dat získaných v době 2011-03-01 00:00 až 2011-03-01 01:30 v univerzitní síti CESNET.

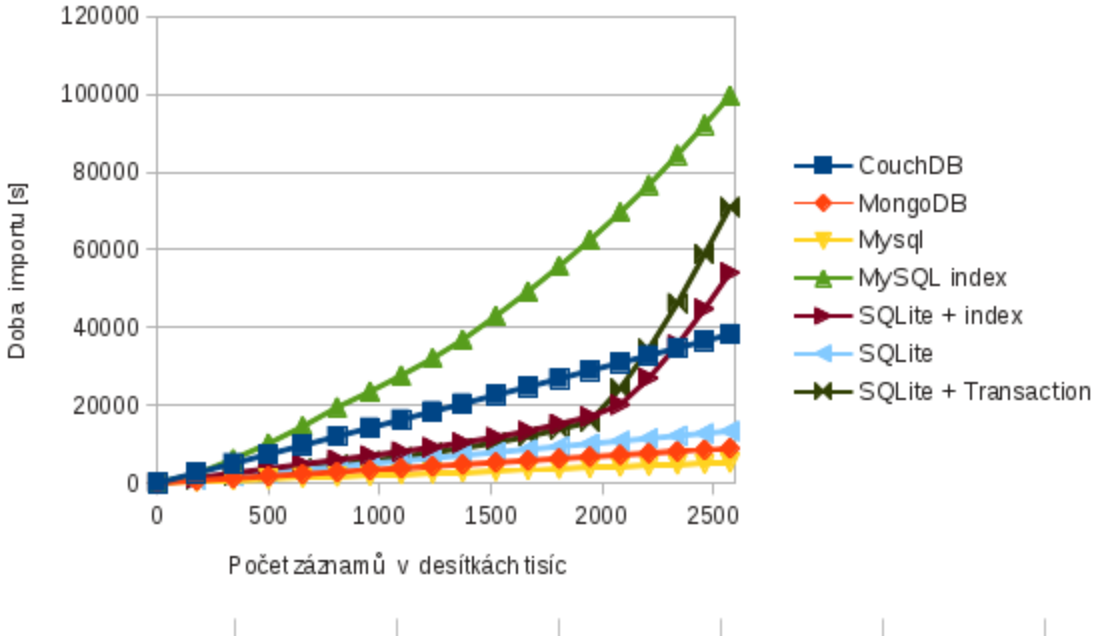
Důležité údaje jsou pro nás:

- rychlost vytvoření databáze a případné výtvoření indexu nad daty,
- rychlost následných dotazů.

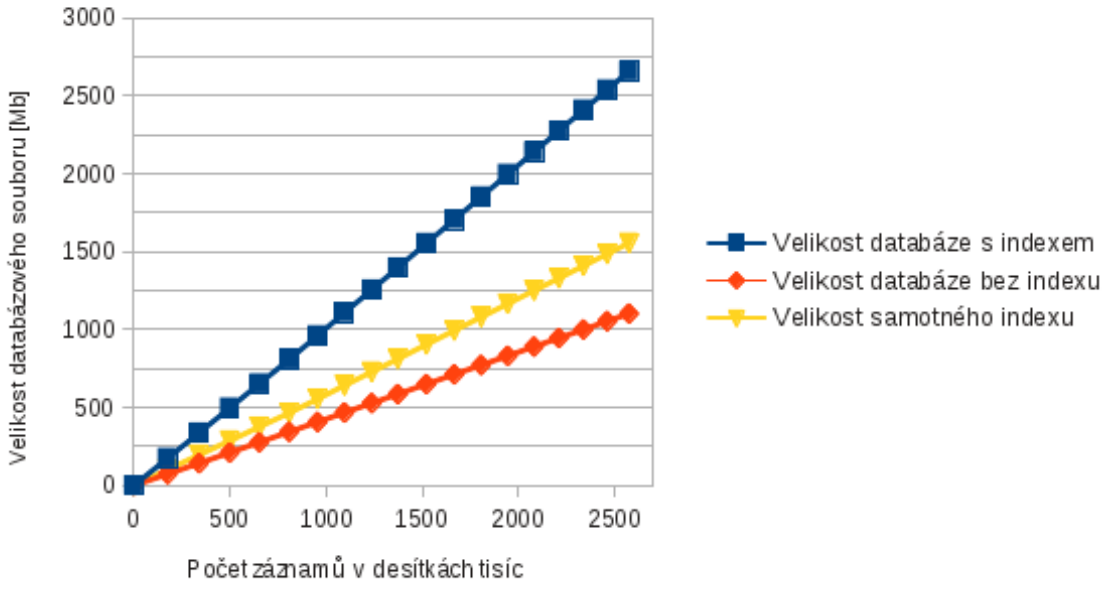
Rychlost vytvoření databáze shrnuje tabulka 4.2. Obecně platí, že pro většinu databází, jakmile velikost indexu překročila velikost operační paměti, došlo k prudké degradaci výkonu, některé testy nebylo možné z důvodu velké časové náročnosti ani dokončit, to se týká databází MongoDB a CouchDB.

DB/počet záznamů v tisících	956	1238	1522	1807	2082	2339	2576
CouchDB	14181 s	18379 s	22602 s	26809 s	30911 s	34703 s	38226 s
MongoDB	3321 s	4289 s	5267 s	6242 s	7199 s	8089 s	8894 s
MySQL	1925 s	2489 s	3071 s	3655 s	4214 s	4735 s	5215 s
MySQL + index	23490 s	32032 s	42938 s	55709 s	69575 s	84407 s	99591 s
SQLite + index	7029 s	9184 s	11745 s	15138 s	20112 s	35508 s	54137 s
SQLite	4943 s	6399 s	7888 s	9361 s	10777 s	12116 s	13355 s
SQLite + index + Transaction	5623 s	7788 s	10377 s	13759 s	24371 s	46226 s	70979 s
FastBit + index	2616 s	3390 s	4165 s	4938 s	5682 s	6376 s	7018 s

Tabulka 4.2: Trvání importu v závislosti na množství importovaných dat. Za povšimnutí stojí hlavně chování databází při využití indexu, kdy dochází k prudkému prodloužení doby se vzrůstajícím množstvím dat. Za index jsme zvolili čas začátku, čas konce, zdrojovou IP adresu v kombinaci se zdrojovým portem a cílovou IP adresu v kombinaci s cílovým portem.



Obrázek 4.1: Graf zachycující rychlost importu a případné indexace dat v závislosti na počtu importovaných záznamů.



Obrázek 4.2: Graf zachycující velikost SQLite databáze v závislosti na počtu importovaných záznamů.

V grafu 4.1 je pozorovatelné prudké zhoršení výkonu u databáze SQLite při použití indexu v okolí 20 000 000 vložených záznamů. Pokud se podíváme do grafu 4.2 vidíme, že velikost SQLite při použití indexu se v okolí 20 000 000 vložených záznamů pohybuje v okolí 2GB, což je i velikost operační paměti našeho testovacího serveru.

V tabulce 4.3 jsme zachytili čas potřebný k vykonání jednotlivých dotazů.

	NFDump	Mysql*	Mysql	Sqlite*	Sqlite	CouchDB	MongoDB	FastBit
dotaz č.1	15.080 s	203 s	153 s	0.01s	78 s	NA	113 s	14.350 s
dotaz č.2	15.553 s	171 s	165 s	0.02s	85 s	NA	151 s	0.289 s
dotaz č.3	15.565 s	175 s	103 s	0.04s	84 s	NA	127 s	0.099 s
dotaz č.4	15.565 s	93 s	110 s	0.02s	89 s	NA	127 s	0.100 s
dotaz č.5	15.565 s	NA	115 s	0.04s	85 s	NA	134 s	0.118 s
dotaz č.6	15.565 s	158 s	134 s	358.00 s	351 s	NA	481 s	1.118 s
dotaz č.7	15.565 s	NA	124 s	0.04 s	86 s	NA	135 s	0.098 s
dotaz č.8	15.565 s	NA	159 s	0.04 s	84 s	NA	NA	2.350 s

Tabulka 4.3: Čas dotazu pro jednotlivé způsoby ukládání dat. NA značí, že dotaz nedoběhl. * značí využití indexu

K testování jsme použili dotazy ze začátku této kapitoly, tedy konkrétně:

1. **Základní dotaz na komunikaci IP adresy.**
2. **Filtrace dle prefixu.**
3. **Filtrace prefixu + agregace podle zdrojové adresy.**

4. **Komunikace z vybraného prefixu do vybrané cílové sítě.**
5. **Komunikace vybraných adres na cílový port 80 (http).**
6. **Struktura protokolu na jednom interface.**
7. **Objemy odchozích dle prefixu pro vybranou síť.**
8. **Objemy odchozích dle prefixu pro vybranou síť a následné seřazení jednotlivých prefixů podle objemu přenesených dat.**

Z výsledků vyplývá, že mnohé databázové systémy nebyly schopny daný dotaz vůbec provést, případně ho provedly až v čase, který je pro nás neakceptovatelný. V tabulce je vidět, že nfdump má ve všech případech čas konstantní, což vyplývá z jeho charakteristiky, kdy sekvenčně prochází bloky dat bez použití indexů. Překvapivě velmi dobrých výsledků dosáhlo SQLite alespoň v případě, že na daný dotaz šlo použít index. Vidíme, že v případě dotazu číslo 6, kde žádný index není k dispozici, došlo k sekvenčnímu čtení, které je značně pomalé.

Z tabulky by mohlo vyplynout, že velmi dobrých výsledků dosáhl FastBit, nicméně zde musíme upozornit na nestabilitu těchto výsledků. Vzhledem k tomu, že jsme pracovali s anonymizovanými bloky dat, tak testovací sada dotazů vracela nulový počet výsledků. Když jsme si toto uvědomili a upravili filtrovací podmínku, dotazy se zpomalily na hodnotu mírně přesahující hodnotu nfdumpu.

Dalším testováním vyšlo najevo, že čas dotazu nad FastBitem je přímo úměrný počtu tázaných sloupců. To vychází z faktu, že FastBit ukládá data z jednoho sloupce vždy do samostatného souboru. Pokud jsme tedy upravili dotaz a odstranili méně relevantní hodnoty, tento dotaz se značně zrychlil. Toto shrnuje tabulka 4.4.

Ukázka dotazů prováděných nad FastBit databází, jediným rozdílem je počet sloupců ve výsledném dotazu:

1. thula -d fastbit-full -s "Date_flow_start, Date_flow_end, Input, Output, Packets, Proto, Bytes, Src_IP, Src_Pt, Dst_IP, Dst_Pt, flows, Tos, TCP_Flags"-w "Src_IP = 1282317556"
2. thula -d fastbit-full -s "Date_flow_start, Date_flow_end, Src_IP, Src_Pt, Dst_IP, Dst_Pt, TCP_Flags"-w "Src_IP = 1282317556"

číslo dotazu	čas
1	13.940 s
2	1.374 s

Tabulka 4.4: Čas různých variant podobného dotazu.

Z výše provedených měření nám nejlépe dopadl FastBit a SQLite. Z důvodu dřívějších špatných zkušeností, kdy SQLite nebylo schopno pracovat se soubory, jež by obsahovaly řádově stamilióny řádků, jsme SQLite vyloučili. Oproti tomu při hledání zdrojů o databázi FastBit jsme našli zmínky ohledně experimentů právě pro ukládání NetFlow dat. Rozhodli jsme se proto toto úložiště podrobit dalším testům.

4.3 FastBit důkladněji

Pro důkladnější srovnání jsme se rozhodli použít blok dat ze sítě CESNET nasbíraný v rozmezí dat 01.07.2011 00:00 až 22.07.2011 00:00. Pro testy jsme využili výkonnější konfiguraci, jež je popsána v tabulce 4.5.

Operační systém	Linux coyote.cis.vutbr.cz 2.6.32-131.6.1.el6.x86_64
CPU	x86_64 Intel(R) Xeon(R) CPU X5650 @ 2.67GHz * 12
RAM	64Gb
HD	6 * DELL 600GB SAS 6Gbps 15k 3.5"v Raid 5

Tabulka 4.5: Konfigurace 2. testovacího serveru.

Na této konfiguraci byla provedena stejná sada testů jako v prvním případě. Výsledky zachycuje tabulka 4.6.

	NFDump	FastBit 5minut	FastBit 3h	FastBit 24h
dotaz č.1	1161.854s	NA	NA	NA
dotaz č.2	1171.223s	NA	NA	NA
dotaz č.3	1164.229s	NA	NA	NA
dotaz č.4	1160.951s	NA	NA	NA
dotaz č.5	1162.656s	NA	NA	NA
dotaz č.6	1240.802s	NA	NA	NA
dotaz č.7	1158.072s	NA	NA	NA
dotaz č.8	1156.187s	NA	NA	NA

Tabulka 4.6: Čas dotazu na druhém testovacím severu. NA značí, že dotaz nedoběhl.

Zde jsme narazili na několik problémů. Prvním problémem je práce s příliš mnoha malými soubory. Při rozdělení souboru do pěti minutových bloků dat je těchto bloků v testovaném rozsahu 21 * 24 * 12 = 6048. Dotaz nad tolika soubory testované utility nebyly schopny uskutečnit. Po 2 hodinách čekání jsme tento experiment ukončili.

Druhým problémem je chybná funkce utility fbmerge pro slučování jednotlivých bloků dat. Tato utilita v současné verzi nerespektuje datové typy jednotlivých sloupců a mění tak jejich hodnotu. Následkem toho nebylo možné agregovat pětiminutové bloky dat a získat z nich tříhodinové a dvacetičtyřhodinové.

Posledním problémem, na který jsme narazili při použití použití tří hodinových a větších bloků dat, byla zpráva o nedostatku paměti.

4.4 Závěrečné srovnání úložišť

Bohužel musíme konstatovat, že v tuto chvíli se nám nepodařilo najít náhradu za používaný nástroj nfdump. Samozřejmě nebylo v našich silách otestovat všechna úložiště a i jejich další vývoj může způsobit a dozajista i způsobí, že data v tomto článku přestanou být aktuální. Nicméně lze předpokládat, že alespoň hrubá platnost zachována zůstane.

Z výše testovaných úložišť vypadá FastBit asi nejlépe, ale je na něm patrná jistá nevyzrálost. Jeho dokumentace je v době psaní této práce na velmi špatné úrovni a narazili

jsme na několik nedostatků, které jsme popsali výše.

Je však důležité upozornit na to, že FastBit je v prvé řadě *úložiště*, pro práci s ním jsme v tuto chvíli používali již předpřipravené nástroje, které nám dobře sloužily pro první odhad možností tohoto způsobu ukládání dat. Nicméně pokud bychom si dali práci a napsali aplikaci, která by byla šitá na míru našim potřebám za využití API tohoto úložiště, velmi pravděpodobně bychom dosáhli lepších výsledků a naopak bychom mohli odstranit některé nedostatky výše zmíněných programů.

Dle našeho názoru má smysl zkoumat úložiště FastBit dále, protože věříme, že existuje nemalá šance na další zrychlení FastBitu jako takového. Zatímco u nfdumpu lze již jen stěží očekávat nárůst rychlosti, u databázových úložišť tato možnost je využitím pokročilejších algoritmů. Nicméně je nutno konstatovat, že v současné podobě bez rozsáhlejších prací je pro nás toto úložiště jen obtížně použitelné.

Jako zcela nevhodné se ukázaly databáze CouchDB a MongoDB. V době psaní této práce probíhalo v prostředí internetu mnoho diskuzí, jež se zabývaly novými druhy databází a odkazovaly na velkou škálovatelnost, rychlost a některé zajímavé možnosti tohoto typu databází. Nicméně se ukázalo, že pro náš typ zátěže jsou tyto databáze zcela nevyhovující. Jako jeden z možných důvodů vidíme fakt, že tyto nové druhy databází jsou optimalizovány pro běh na více serverech a horizontální škálování, což jsme v našich podmínkách nevyužívali.

Pro naši další práci se tak budeme muset spokojit pouze s nfdumpem a budeme hledat jiné cesty pro zvýšení výkonu.

Kapitola 5

Návrh aplikace

V této části naší práce si popíšeme tvorbu vlastní aplikace s přihlédnutím k již existujícím nástrojům, jež jsme si popsali v kapitole 3. Všechny aplikace, které jsme měli možnost poznat, poskytovaly data převážně agregovaná. Bylo z nich možné vyčíst informaci o celkovém množství přenesených paketů, procentuálním zastoupení protokolů a podobné údaje. Tyto informace najdou využití například u společností, jež poskytují internetové připojení, aby mohly správně účtovat náklady svým zákazníkům. Případně mohou detekovat narůstající síťový tok a upozornit administrátora na nutnost výměny síťových prvků za výkonnější a tím předejít výpadkům sítě.

Dané aplikace však nejsou příliš užitečné při zodpovídání konkrétněji položených otázek. Kupříkladu nám nepomohou v odpovědi na otázku, které IP adresy komunikovaly s naším serverem v poslední hodině, což je informace, jejíž důležitost si uvědomíme ve chvíli, kdy nám do daného serveru někdo neoprávněně vnikne. Další nedostatek těchto aplikací byl v jen omezené, případně žádné možnosti zpřístupnit data dalším uživatelům. Pokud máme na starost velkou počítačovou síť, jakou bezesporu síť VUT je, máme často potřebu rozdělit ji například na podsítě a správcům jednotlivých podsítí zpřístupnit pouze data, která jim náleží. Dalším příkladem může být situace, kdy nás státní orgán vyzve ke sledování aktivity určitého uživatele. I zde by bylo vhodné data tohoto uživatele předpřipravit do samostatného *pohledu* a danému státnímu orgánu tento *pohled* následně zpřístupnit.

Tyto a další požadavky si v této kapitole důkladněji rozebereme, na jejich základně provedeme analýzu a navrhneme strukturu naší webové aplikace.

Vzhledem k povaze údajů, se kterými bude naše aplikace pracovat, bude dbán velký důraz na zabezpečení. Proto mu věnujeme samostatnou část naší práce a důkladně se seznámíme s technikami útoku na internetové aplikace, stejně tak si řekneme jak se proti daným útokům bránit.

5.1 Analýza požadavků

Jak jsme předeslali výše, cílem práce je vytvořit nástroj, který doplní portfólio již existujících nástrojů a nalezne praktické uplatnění v síti VUT Brno. Požadavky, které jsou na tento nástroj kladeny, jsou následující.

Snadná konfigurace a udržovatelnost: Tyto pojmy jsou veskrze abstraktní. Co může být pro jednoho uživatele snadné, je pro jiného obtížné. Lze však předpokládat, že budeme-li se držet osvědčených programátorských praktik, bude systém udržovatelný a snadný pro správu.

Vysoký výkon: Z analýzy potřeb počítačové sítě VUT Brno vyplývá potřeba zvládnout minimálně 40 000 000 vložených záznamů za hodinu a zpracovávat data v řádech stovek GB. Zde využijeme kapitolu naší práce, jež se týkala ukládání dat 4. Z jejího závěru vyplývá, že nejvhodnější bude využít pro ukládání nativní formát utility nfdump a nespoléhat se na jinou databázovou vrstvu.

Vhodným se proto jeví využití současné infrastruktury, kdy jsou data ukládána pomocí utility *nfcapd*, bez agregace a vzorkování. My bychom měli s tímto počítat a nedegradovat data vzorkováním a agregací na aplikační úrovni, nebude-li si to uživatel vyloženě přát. Využití současné infrastruktury nám navíc ulehčí práci, jelikož nebudeme muset řešit samotný sběr dat a budeme se moci soustředit na vlastní analytickou část aplikace.

Použití svobodných technologií s vhodnou licencí: Souvisí částečně s požadavkem na snadnou udržovatelnost. Svobodné technologie s otevřeným zdrojovým kódem nám umožní lépe adaptovat software našim potřebám. Jako vhodné se pro naše potřeby jeví licence BSD[1], MIT[2], Apache[3], (L)GPL[4] a další.

Aplikace musí být provozovatelná na Linuxovém serveru: [5] Jedná se o funkční požadavek. Aplikace by ale neměla být vázána na konkrétní operační systém.

Aplikace musí být bezpečná: Každá aplikace, kterou tvoříme, musí být co možná nejbezpečnější, zde však z povahy údajů, se kterými budeme pracovat, je nutno dbát zvýšené pozornosti a zajistit co možná největší míru bezpečí daných údajů.

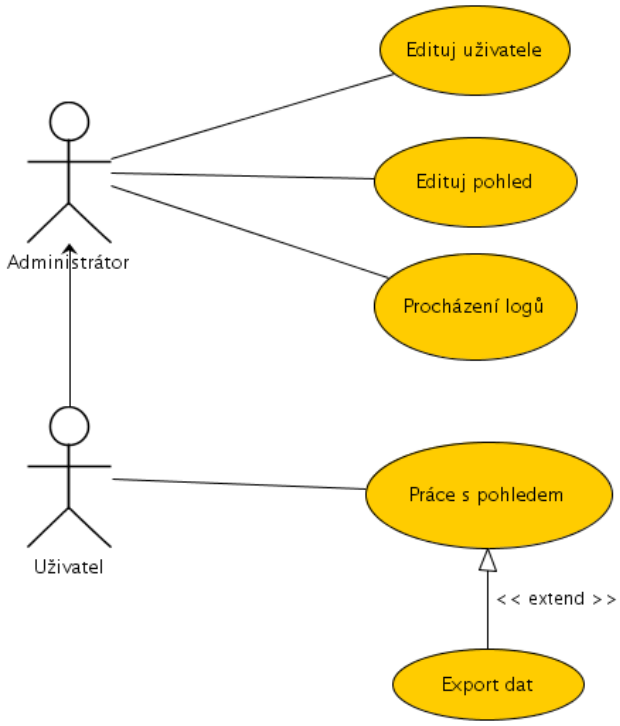
Aplikace by měla podporovat práci více uživatelů: Existuje jistý předpoklad, že danou aplikaci bude do budoucna využívat větší množství uživatelů. Je nutné, aby s tím aplikace počítala a umožnila řídit přístup těchto uživatelů k datům a jiným zdrojům, jež bude aplikace poskytovat. Uživatele rozdělíme do dvou rolí. První rolí je Administrátor, jenž má právo vytvářet další uživatelské účty a má také přístup k logům, jež zaznamenávají chování dalších uživatelů. Dále může vytvářet nové pohledy a přiřazovat práva k těmto pohledům jednotlivým uživatelům. Z toho vyplývají základní případy užití naší aplikace, jež jsou zachyceny na obrazku 5.1.

Z analýzy požadavků vyplývá, že aplikace se bude muset vypořádat s velkým množstvím údajů. Tyto údaje nebudou uloženy v databázi, není nad nimi vytvořen žádný index, a je proto nutné je procházet sekvenčně, což může být často poměrně náročné. Bylo by proto dále vhodné, aby aplikace umožňovala některé dotazy předpřipravit předem. Zároveň by bylo užitečné, kdyby z takto předpřipraveného dotazu šlo vytvářet dotazy další.

Tohoto přístupu lze zároveň s výhodou využít v případě, kdy budeme schopni uživatelům přidělovat práva k jednotlivým předpřipraveným dotazům. Tím získáme možnost omezit přístup konkrétních uživatelů pouze na data, která nezbytně potřebují.

Vzhledem k velkému množství dat o síťovém toku bychom chtěli omezit nutnost jejich přenosu uvnitř počítačové sítě. Vhodná se proto jeví serverová aplikace a tenký klient. Zamezíme tím i potenciálnímu úniku citlivých informací a usnadníme správu, jelikož veškerá

[1] http://www.opensource.org/licenses/bsd-license.php
[2] http://www.opensource.org/licenses/mit-license.php
[3] http://www.apache.org/licenses/LICENSE-2.0
[4] http://jxself.org/translations/gpl-3.cz.shtml
[5] Konkrétně pak Red Hat Enterprise Linux Server 6.2, obecně ale libovolný *nixový server, jenž splní další požadavky.



Obrázek 5.1: Základní diagram použití naší aplikace.

data mohou být uložena v jednom úložišti. Za nejvhodnější v tuto chvíli považujeme tedy aplikaci webovou, která splňuje podmínky snadné rozšiřitelnosti a přístupnosti.

Naše aplikace bude pracovat se dvěma druhy dat. První skupinou jsou data samotné aplikace, těch nebude příliš mnoho. Pro práci s nimi využijeme některý z dostupných ORM[6] nástrojů, čímž opět o něco zvýšíme přenositelnost.

Druhou skupinou dat jsou samotná data o síťovém toku. Zde jsme věnovali samostatnou kapitolu 4 hledání nejvhodnějšího možného úložiště, abychom zvýšili rychlost aplikace. Z důvodů shrnutých v části 4.4 jsme však nakonec zůstali u dat v nativním formátu utility nfdump.

I tato data však chceme uchovávat v co možná nejvýhodnější adresářové struktuře. My jsme zvolili strukturu, kdy pro každý den máme samostatný adresář, jenž obsahuje pro každou hodinu další vnořený adresář. Až ten pak obsahuje soubory vytvořené utilitou *nfcapd* pro jednotlivé pětiminutové bloky. Takto je možno velmi snadno odmazávat nejstarší data a zároveň na základě času velmi snadno určit, které soubory je nutno prohledávat. Výsledek tedy může vypadat například takto:

```
/2012-05-01
    /00
        /nfcapd.201205010000
        /nfcapd.201205010005
        /nfcapd.201205010010
```

[6] Objektově-relační mapování

5.2 Popis vlastní aplikace

V předchozích částech naší práce jsme nasbírali základní údaje a podklady pro tvorbu naší aplikace. Nyní se zaměříme na samotný vývoj. Po zhodnocení kladů a záporů jsme se rozhodli využít tyto prostředky

nfdump: pro základní zpracování netflow dat. Cílem práce je vyvinout robustní řešení, které nám umožní snadnou záměnu za jiný nástroj, případně jiné úložiště dat o síťových tocích. Nicméně jak ukázala předchozí kapitola 4, v danou chvíli nemá nad jiným způsobem ukládání záznamů příliš smysl uvažovat. Všechny alternativní způsoby měly nedostatky, které jsme vyhodnotili jako natolik zásadní, že jsme přistoupili k využití čistě nfdumpu. Na jednu stranu je škoda, že se tak vzdáváme možnosti využít pokročilých možností databázových úložišť, nicméně i využití samotných souborů využívaných utilitou nfdump nám přinese výhody v podobě snadnější administrace.

PHP: ve verzi 5.3 nebo vyšší. Tento jazyk je široce využívaný při tvorbě interaktivních webových aplikací, lze tedy očekávat, že i nám bude vyhovovat. Verze starší než 5.3 již nejsou oficiálně podporovány, proto ani my na ně nebudeme brát ohled. V průběhu psaní této práce vyšla verze 5.4, a proto si klademe za cíl zajistit dopřednou kompatibilitu i s touto verzí.

Doctrine: Pro snadnou persistenci modelů je vhodné využít některého z ORM nástrojů. My jsme se rozhodli využít nástroj Doctrine[7]. Tento nástroj na rozdíl od konkurenčního řešení Propel[8] využívá návrhový vzor Data Mapper, který lépe odpovídá našim potřebám. Více se tomuto nástroji věnujeme v sekci 5.2.1.

Zend Framework: Široce používaný MVC framework, jehož domovská stránka se nachází na adrese `http://framework.zend.com/`. MVC je zkratka pro Model - View - Controller, což je jedna z architektur, která se využívá při budování aplikací.

Snahou zde je dekompozice problému do samostatných celků, kdy vrstva Controlleru zpracovává požadavky od uživatele a za pomoci Modelů je předává do View vrstvy.

Model je část obsahující samostatnou logiku popisující náš problém a samotnou aplikaci. Běžně máme například model popisující uživatele, adresu nebo formulář.

View je část aplikace, jejímž účelem je prezentace dat uživateli. Může se jednat o X/HTML šablonu, ale například i o PDF nebo graf.

Více se tomuto nástroji věnujeme v sekci 5.2.2.

SVN: Jako systém pro správu verzí zdrojových souborů použijeme program Subversion, zkráceně SVN. Díky tomu se nemusíme bát experimentovat se změnami. Máme jistotu, že v případě potřeby jsme schopni se vrátit k jakékoli verzi projektu v čase a tím odstranit případné chyby. Zvažovali jsme také využití stále populárnějšího gitu. Ten je však určen spíše pro spolupráci více vývojářů. Jeho předností je snadné vytváření takzvaných forků, tzn. odnoží projektu. Vývojář je schopen vzít si aktuální verzi projektu, provést v ní změny a ty následně v případě potřeby snadno přesunout do původní verze projektu. Vzhledem k tomu, že v tuto chvíli nepředpokládáme větší počet vývojářů, na našem projektu využijeme Subversion z důvodu o něco lepší podpory v ostatních vývojářských nástrojích.

[7] http://www.doctrine-project.org/
[8] http://www.propelorm.org/

Více si dané nástroje popíšeme v samostatných sekcích.

5.2.1 Knihovna Doctrine 2

Doctrine 2 je ORM framework šířený pod licencí LGPL[9] pro prostředí PHP. ORM je zkratka pro objektově relační mapování, což je proces, při kterém jsou objekty daného jazyka ukládány do relační databáze a následně z ní opět načítány. Toto je úkol, se kterým se musí potýkat téměř všechny informační systémy. Z toho důvodu existuje pro většinu programovacích jazyků jedno nebo více hotových řešení.

My si v této sekci představíme námi použitý nástroj Doctrine ve verzi 2.0. Od své předchozí verze Doctrine 1 a od konkurenčního řešení Propel (v tuto chvíli ve verzi 1.6) se liší v tom, že využívá, narozdíl od vzoru ActiveRecord, návrhový vzor Data Mapper. Ten se vyznačuje tím, že veškeré operace spojené s uložením, načtením a smazáním objektu z relační databáze jsou definovány v samostatné třídě, nikoli v mapovaném objektu. Tak je zajištěn takzvaný *Single responsibility principle*[10], což je jeden z pilířů dobrého objektového návrhu[16]. V Doctrine 2 se tato třída, jenž se stará o načítání a ukládání dat do relační databáze, nazývá EntityManager.

Kód využívající návrhový vzor DataMapper pak vypadá například takto:

```
<?php

/*
* Základní inicializace proměnných a
* získání objektu typu EntityMamager.
*/
...

$flowView = new \Nefele\Entities\FlowView();
$flowView->setName('Obecný pohled');
$flowView->setDefinition('PROTO TCP');

/**
* Předáme objekt k perzistenci.
* EntintyManager od této chvíle zpravuje předaný objekt,
* zde stojí za povšimnutí, že objektu $flowView
* nebyla zaslána žádná zpráva o tom, že došlo k jeho uložení.
* Což můžeme interpretovat jako nezávislost na databázové vrstvě.
*/
$EntityManager->persist($flowView);

/**
* Veškeré změny se nyní promítnou do databáze jako příslušné
* INSERT / UPDATE / DELETE operace.
*/
$EntityManager->flush();
```

[9] GNU Lesser General Public License
[10] Princip jedné odpovědnosti

Alternativou k výše zmíněnému vzoru DataMapper je návrhový vzor ActiveRecord. Právě ten byl použit v první verzi Doctrine a nástroje Propel. Tento návrhový vzor se vyznačuje tím, že každá třída mapovaná do databáze, obsahuje v nějaké formě metody ekvivalentní operacím save, load a delete. Tento přístup má však značné nevýhody:

- Příliš silné propojení na databázovou vrstvu. Snahou návrháře softwaru by vždy měla být co nejslabší vazba mezi jednotlivými třídami a tato snaha je tímto narušena.
- Při implementaci metod load, save, delete můžeme buď v každé jedné třídě implementovat tyto metody znovu, nebo vytvořit abstraktního předka, od něhož budou tyto metody děděny. První řešení zavádí do kódu redundanci, jelikož velmi podobný kód se bude vyskytovat na mnoha místech. Druhé řešení nás zase omezí ve schopnosti podědit jinou než naši abstraktní třídu.

Kód využívající návrhový vzor ActiveRecord vypadá například takto:

```
<?php

/**
* Základní inicializace proměnných.
*/
...

/**
* Vytvoření instance nového objektu
*/
$flowView = \Nefele\Entities\FlowView::create();
$flowView->setName('Obecný pohled');
$flowView->setDefinition('PROTO TCP');


/**
* Vlastní uložení do databáze, jež se provede jako INSERT operace.
* Vidíme, že o uložení do databáze žádáme přímo náš konkrétní objekt,
* což značí silnou vazbu na databázové úložiště.
*/
$flowView->save();
```

Chceme-li využít nějakou z forem objektově relačního mapování, musíme si nejprve definovat sadu pravidel, dle kterých budeme daný objekt ukládat do relační databáze. Daná pravidla obsahují informaci jako například:

- Jméno databázové tabulky.
- Jméno sloupce tabulky, která odpovídá členské proměnné objektu.
- Datový typ sloupce tabulky, která odpovídá členské proměnné objektu.
- V neposlední řadě je nutno definovat vazby na ostatní objekty.

Tato pravidla nám Doctrine 2 umožňuje definovat vícero způsoby:

XML definice: Všechny potřebné informace popisující danou třídu objektů jsou zapsány do konfiguračního XML souboru.

YAML definice: Všechny potřebné informace popisující danou třídu objektů jsou zapsány do konfiguračního YAML souboru. YAML je zkratkou pro YAML Ain't Markup Language. Jedná se o formát pro popis strukturovaných dat, na rozdíl od XML je více přístupný člověku.

Speciální forma PHP komentáře: Daná třída a její atributy jsou popsány speciálním komentářem, který definuje způsob, jakým se má daný objekt uložit.

PHP kód: Přes příslušné metody definujeme, jakým způsobem má EntityManager mapovat naše objekty do relační databáze.

My jsme si zvolili variantu speciálních PHP komentářů. Toto řešení má dvě hlavní nevýhody:

1. Není formálně zcela správné. Do určité míry tak totiž narušujeme právě výše popsaný *Single responsibility principle*. Nicméně dle našeho názoru se jedná opravdu jen o formalizmus, jelikož komentáře jsou spíše dokumentací než výkonným kódem.

2. Využití některých nástrojů, jež si kladou za cíl urychlit běh PHP aplikací[11], se stává náročnější, jelikož tyto nástroje jako součást svých optimalizací komentáře odstraňují.

Dle našeho názoru však výhody převažují, za všechny jmenujme:

1. Snadná údržba, protože definici mapování i mapovanou třídu editujeme v jediném zdrojovém souboru.

2. Dané komentáře mají pozitivní vliv na štábní kulturu a kód naopak zpřehledňují.

Námi vytvořená třída pak může vypadat například takto:

```
<?php
namespace Nefele\Entities;

 /**
 * @\Doctrine\ORM\Mapping\Entity
 */
class FlowView{

    /**
     *  @\Doctrine\ORM\Mapping\Column(type="integer",name="id")
     *  @\Doctrine\ORM\Mapping\Id
     *  @\Doctrine\ORM\Mapping\GeneratedValue(strategy="AUTO")
     */
    protected $id;

     /**
      * Jméno daného pohledu
      *
```

[11] http://sourceforge.net/projects/eaccelerator/

```
     * @\Doctrine\ORM\Mapping\Column(type="string",nullable=false);
     */
    protected $name;

    /**
     * Uživatelé, jež mají přístup k danému pohledu
     *
     * @\Doctrine\ORM\Mapping\ManyToMany(
     *    targetEntity="User",inversedBy="flowViews")
     * @\Doctrine\ORM\Mapping\JoinColumn(
     *    name="owner",referencedColumnName="id")
     */
    protected $owners;

    /**
     * Definice daného pohledu
     *
     * @\Doctrine\ORM\Mapping\Column(type="string",nullable=false);
     */
    protected $definition;

    // další proměnné a metody této třídy
    ...
}
```

Za povšimnutí stojí rozsáhle komentovaný kód, kdy speciálním druhem komentářů sděluji objektu typu EntityManager, jak se chovat k jednotlivým atributům naší třídy. Jak jsme psali dříve, tyto komentáře mají navíc z našeho pohledu dobrý vliv na štábní kulturu. Programátor je z nich schopen získat další informace, které může využít například při automatické tvorbě formulářů a validačních pravidel, což by bylo v dynamicky typovaném jazyce jinak značně komplikované. Pro popis významu jednotlivých komentářů odkážu čtenáře na dokumentaci nástroje Doctrine[12], nicméně věřím, že kód je do jisté míry samopopisný a snadno pochopitelný.

5.2.2 Knihovna Zend Framework

Jak jsme psali v úvodu této kapitoly, Zend Framework je open-source PHP framework vyvíjený společností Zend Technologies. Tato společnost již několik let investuje do rozvoje technologií postavených na jazyku PHP. Zajímavým projektem je například Zend Server, což je mírně upravený Apache server s funkcemi pro snadný deployment PHP aplikací, jejich debugování, reportování chyb z ostrého provozu, vestavěnou optimalizací PHP kódu a dalšími užitečnými vlastnostmi.

Využitím tohoto frameworku zkrátíme čas potřebný k vývoji aplikace, zvýšíme kvalitu kódu a snížíme riziko bezpečnostní chyby. Jednou z výhod využití tohoto frameworku je, že striktně rozděluje projekt na část, která je přímo dostupná prostřednictvím webového prohlížeče. Tato část může obsahovat soubory s definicí kaskádových stylů, javascriptové

[12] http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/annotations-reference.html

soubory nebo obrázky a část, jež takto přístupná není. Sem spadá kód vlastní aplikace, konfigurační soubory a jiné části naší aplikace, jež by měly zůstat skryty.

Vstupním bodem naší aplikace pak bude jediný soubor *index.php*, na který jsou pomocí Apache modulu mod_rewirte směrovány veškeré požadavky uživatele. Díky tomu půjdou všechny uživatelské požadavky na naši aplikaci prostřednictvím jediného vstupního bodu. To nám umožní snadnou aplikaci základních bezpečnostních pravidel, jako je ověření uživatele, detekce potencionálních útoků a další. Bezpečnosti se bude více věnovat samostatná kapitola 6.

5.2.3 Popis modelů

V této části si popíšeme nejdůležitější modelové prvky naší aplikace. Ty se dají rozdělit do čtyř základních částí neboli balíčků:

Balíček Machine: Tvoří základní abstrakci nad službami poskytovanými operačním systémem.

Balíček Cli: Tvoří rozhraní naší aplikace dostupné z příkazové řádky. Přes toto rozhraní můžeme naši aplikaci snadno instalovat, odinstalovat nebo některé její části volat z cronu. Což, jak si povíme později, je velmi výhodné.

Balíček Netflow: Kolekce tříd pro práci s netflow daty.

Balíček Entities: Soubor entit pro popis stavu aplikace, jedná se například o uživatele nebo seznam předpřipravených dotazů.

Balíček Machine

Jelikož budeme z naší aplikace často volat další programy nebo jinak využívat služeb operačního systému, bude pro nás vhodné zavést si abstrakci nad jednotlivými, často se opakujícími úkony. Dalším požadavkem, který na náš systém klademe, je co možná nejvyšší míra bezpečnosti. I z toho důvodu je pro nás maximálně vhodné veškerá volání operačního systému zapouzdřit a spravovat z jednoho místa. Na tomto místě důsledně dbáme na ošetření všech potenciálně nebezpečných vstupů funkcí *escapeshellarg*, která zajistí korektní chování nezávisle od použitého operačního systému. Dále nám jednotná lokace všech funkcí spojených s operačním systémem umožní snadné logování všech prováděných operací spolu s časem a uživatelem. Díky tomu budeme moci provádět audit chování uživatelů a případně detekovat jejich nestandardní chování, jež by mohlo značit snahu napadnout náš systém.

Balíček Cli

Jak jsme psali v předchozích kapitolách, jedním z problémů ostatních nástrojů je poměrně složitá instalace a administrace aplikace. My jsme se pokusili tomuto problému čelit, a proto jsme neopomněli ani tuto část. Součástí aplikace je rozhraní, které je možné obsluhovat prostřednictvím příkazového řádku. Navíc je systém navržen tak, aby byl velice snadno rozšiřitelný o nové úlohy, kterými může být počítání statistik, automatické varování obsluhy v případě nestandardního chování v síti a další.

Spouštění jednotlivých úloh se provádí voláním `php index.php -t jménoÚlohy`, zde jsme využili možnosti mít pouze jeden vstupní bod, přes který bude naše aplikace komunikovat tak, jak jsme si popsali dříve. V případě tohoto volání *jménoÚlohy* odpovídá názvu

třídy v balíčku \Nefele\Cli\Task. Daná třída musí implementovat rozhraní *taskInterface*, které předepisuje metodu doTask. Tato metoda přebírá jako argument parametry příkazové řádky zabalené do objektu \Zend_Console_Getopt.

Dále je jednotlivým úlohám předán objekt \Zend_Application_Bootstrap_Bootstrap, který slouží jako přepravka všech zdrojů naší aplikace. Konkrétně se může jednat například o databázový adaptér, adaptér pro logování událostí nebo sada konfiguračních direktiv. Všechny tyto zdroje jsou inicializovány právě pomocí souboru *index.php*.

Balíček Netflow

Obsahuje základní třídy pro práci s NetFlow. Jedná se hlavně o třídy \Nefele\Netflow\Query pro popis vlastního dotazu nad NetFlow daty a \Nefele\Netflow\Engine\Nfdump pro abstrakci nad utilitou *nfdump*. Původním záměrem bylo vytvořit úplnou abstrakci nad použitým nástrojem a protokolem. Bylo by pak možné snadno vyměnit jednotlivé části naší aplikace, číst data z databáze, případně vyměnit NetFlow data za data v jiném formátu. V průběhu práce jsme však od tohoto cíle ustoupili. Tato podmínka by byla až příliš svazující a negativně by se podepsala na výkonu celé aplikace. Přesto jsme se však snažili problém dekomponovat do logicky souvisejících celků a případná úprava aplikace, například pro nové úložiště, by tedy měla být poměrně jednoduchá i přesto, že bude nutno editovat přímo zdrojový kód.

Balíček Entities

Téměř každá aplikace si potřebuje uchovávat informace o vnitřním stavu. V naší aplikaci zastřešuje kontrolu nad tímto vnitřním stavem skupina tříd spadající do balíčku \Nefele\Entities. Nejdůležitější třídy z tohoto balíčku jsou:

- Person
- FlowViewFormat
- FlowView
- User

Nejvíce nás budou zajímat entity FlowView a FlowViewFormat. FlowViewFormat zastupuje výstupní formát netflow dat. V první iteraci projektu se jednalo o pouhý seznam sloupců. V průběhu řešení však vyvstala otázka umožňovat nad těmito sloupci provádět různé transformace, jako například možnost dohledat vlastníka dané IP adresy. Zavedli jsme tedy možnost nad sloupcem vykonat uživatelem definovanou funkci. Pro snazší práci je možno v aplikaci vytvořit sadu předpřipravených výstupních formátů a ty následně vybírat z číselníku.

FlowView je nejdůležitější entita naší aplikace, která zastřešuje podmnožinu netflow záznamů. Je to obdoba databázového *pohledu*. Značně jsme se zde inspirovali v projektu nfsen, mnoho atributů je tedy podobných. Konkrétně se jedná o tyto:

id: Tento atribut slouží pro jednoznačnou identifikaci konkrétního pohledu.

name: Jméno konkrétního pohledu pro snazší orientaci uživatele.

creator: Tvůrce daného pohledu, tento atribut má pouze informační charakter.

owners: Vlastníci daného pohledu, tzn. uživatelé kteří mají k tomuto pohledu přístup.

definition: Definice pohledu, ve formě platného filtrovacího výrazu pro nfdump.

parentFlowView: Pohled, ze kterého je tento pohled odvozen, v aplikaci je automaticky dostupný jeden LIVE pohled, který obsahuje veškerá dostupná data, od něj pak můžeme odvozovat pohledy další.

type: Typ pohledu, naše aplikace umí pracovat se dvěma typy pohledů

- **shadow:** Tento typ funguje na principu nematerializovaného pohledu. Pracuje se tedy nad daty nadřazeného pohledu, pouze je do filtrovacích pravidel přidána nová podmínka na základě atributu *definition*. Výhodou je nenáročnost na diskový prostor, nevýhodou je delší čas potřebný pro zpracování.
- **real:** Odpovídá materializovanému pohledu. Data jsou skutečně uložena na disku, což má za následek jejich rychlejší zpracování. Nepříjemným důsledkem je však zvýšený nárok na úložnou kapacitu.

window: Daný pohled je možno vytvořit dvojím způsobem, buď jako časově ohraničenou množinu záznamů, například od 21.1.2012 12:00 do 22.1.2012 12:15, nebo pohyblivé časové okno, například za posledních 5 dní. První druh nazýváme *historical* druhý nazýváme *continuous*

startTime: Začátek generování pohledu. Smysl má pouze u pohledu typu historical.

endTime: Konec generování pohledu. Stejně jako startTime má smysl pouze u pohledu typu historical.

windowLength: Určuje za kolik posledních dní budeme generovat tento pohled. Smysl má pouze u pohledu typu *continuous*.

lastRun: Datum posledního generování pohledu. Smysl má tato hodnota pouze u pohledu typu *real continuous*.

Pro všechny výše zmíněné entity musíme zajistit persistenci. Tedy trvalé uchování jejich stavu. Jedním z řešení tohoto problému je serializace příslušných objektů do JSON[13], případně XML[14] dokumentů. Toto řešení se vyznačuje velmi dobrou přenositelností, je však poměrně pracné a pro rozsáhlejší kolekce objektů se nehodí.

My jsme proto zvolili jiné řešení, a to uložení objektů do relační databáze. Toto řešení se sice vyznačuje o něco menší přenositelností, jelikož na daném systému musí být daná relační databáze nainstalovaná, na druhou stranu je však toto řešení výkonnější a méně pracné. Problém s přenositelností do značné míry řeší užití ORM nástroje Doctrine, který tvoří abstrakci nad konkrétním databázovým serverem, a jenž jsme si popsali dříve v této kapitole 5.2.1.

[13] JavaScript Object Notation
[14] Extended Markup Language

5.3 Shrnutí

V této části jsme si v hrubých rysech načrtli strukturu naší aplikace. Popsali jsme některé nároky, jež na ni budou kladeny, a nástroje, které budeme dále využívat. Ještě před započetím vlastního programování jsme se rozhodli věnovat určitý čas zabezpečení naší aplikace. Data, se kterými budeme pracovat, by rozhodně neměla být volně dostupná a naše aplikace musí zajistit jejich bezpečnost.

Jako jméno naší aplikace jsme zvolili *Nefelé*, což byla dle řecké mytologie bohyně oblaků. Využíváme jisté podobnosti mezi slovy NetFlow, nfdump a Nefelé a zároveň využíváme odkazu na oblaka, v angličtině *cloud*, jelikož je tento termín v poslední době stále populárnější.

Kapitola 6

Zabezpečení

Ze zadání plyne, že celá práce má být tvořena formou interaktivní webové aplikace. Jelikož budeme pracovat s údaji, které by se rozhodně neměly dostat do nepovolaných rukou, je velmi důležité naši aplikaci správně zabezpečit. Praxe navíc ukazuje, že bezpečnost webových aplikací je neustálý problém, který způsobuje podnikům nemalé ekonomické ztráty. Proto se této problematice budeme hlouběji věnovat i my. V této kapitole se seznámíme s pojmy jako je autentizace, autorizace, seznámíme se i s nejběžnějšími útoky na webové prezentace a naučíme se proti nim bránit.

Jako relevantní zdroj údajů v této práci používáme údaje společnosti OWASP[1]. Tato společnost poskytuje nepravidelně údaje o nejrozšířenějších zranitelnostech webových aplikací. Z [15] jsme přebrali následující přehled nejčastějších zranitelností pro rok 2010.

Injection: Jedná se o typ útoku, kdy útočník odešle data, která jsou následně bez další validace použita v SQL dotazu, případně zadaná jako příkaz operačnímu systému, a to jiným způsobem, než bylo autorem systému zamýšleno. Více se tomuto typu útoku budeme věnovat v 6.1.

Cross Site Scripting (XSS): Jedná se o typ útoku, kdy jsou útočníkem předaná data do webového prohlížeče bez toho, aniž by tyto údaje byly správně ošetřeny. Toto umožní útočníkovi spustit libovolný skript v prohlížeči oběti, která navštíví takto postihnutou stránku, což může mít další dopad na bezpečnost oběti. Popsanému typu útoku se dále věnujeme v 6.2.

Chybná autentizace a správa sezení: Chybná implementace těchto mechanizmů může vést k podvržení identity, případně získání identity jiného uživatele systému. Problému se budeme věnovat hlavně v části 6.3.

Nezabezpečený odkaz na zdroje a soubory: K tomuto dochází, pokud programátor ponechává volně dostupné zdrojové nebo jiné soubory aplikace. Může se jednat například o konfigurační soubor obsahující přihlašovací jméno a heslo k databázi nebo jiné údaje citlivé povahy.

Tato situace nastává také v případě, kdy programátor nekontroluje dostatečně oprávnění uživatele k přístupu k daným údajům. Pak lze často pouhou modifikací URL získat přístup k údajům, které danému útočníkovi nenáležejí. Více se této problematice budeme věnovat v 6.4.

[1] Open Web Application Security Project - https://www.owasp.org/index.php/Main_Page

Cross Site Request Forgery (CSRF): Takto se označuje útok, kdy útočník odešle z prohlížeče oběti speciálně upravený HTTP požadavek, a to prostřednictvím upravené stránky, kterou oběť navštíví. Systém následně vykoná příslušný požadavek, stejně jako kdyby se jednalo o legitimní požadavek oběti. Tento útok je často prováděn spolu s XSS. Více se tomuto typu útoku budeme věnovat v 6.5.

Chybné nastavení zabezpečení na úrovni serveru a knihoven: Používané aplikace často spoléhají na prostředky třetích stran. Je tedy nutno dbát i na správné nastavení těchto prostředků. Může se jednat o webový nebo databázový server, případně použité knihovny. Navíc je potřeba dbát na aktualizaci těchto prostředků v případě, že je v nich nalezena bezpečnostní díra.

Nezabezpečené ukládání údajů: Mnoho databází stále ukládá citlivé údaje jako jsou hesla nebo čísla kreditních karet. Tyto údaje je nutno zahashovat v případě hesel, nebo alespoň zašifrovat v případě kreditních karet. Více se problému ukládání hesel v databázích věnujeme v sekci 6.4.

Nedostatečné zabezpečení odkazů: Aplikace často kontrolují oprávnění pouze při generování odkazů a spoléhají na to, že odkazy, které uživatel nevidí, nenavštíví. Jednoduchou manipulací s URL je pak možné získat přístup i k těmto neprezentovaným zdrojům. Tomuto se také věnujeme v části 6.4.

Nezabezpečená ochrana transportní vrstvy: Aplikace využívají slabý bezpečnostní certifikát nebo slabou šifru pro zabezpečení transportní vrstvy. Toto spadá spíše pod administraci samotného webového serveru a my se tomuto příliš věnovat nebudeme.

Nezabezpečené přesměrování: Mnohé webové aplikace přesměrovávají uživatele pomoci HTTP hlavičky nebo JavaScriptu na jiné spřátelené stránky. Často je možné ovlivnit cíl tohoto přesměrování například parametrem v URL. Pokud je pak oběti podstrčena tato stránka, je oběť přesměrována na stránku útočníka. To je obzvláště vhodné při útocích, kdy se snažíme z oběti vylákat citlivé údaje. Lze si představit webový obchod, který přesměrovává uživatele na stránku portálu zabezpečujícího platby. V případě zmanipulovaného přesměrování může být oběť přesměrována na stránky, které se vydávají za platební portál, ale data jsou následně poskytnuta útočníkovi. Naše aplikace však toto nevyužívá a proto se tomuto problému nebudeme více věnovat.

Za pozornost stojí i seznamy pro rok 2004 [13] a 2007 [14], kde je vidět, že se dané seznamy příliš neliší. Z toho lze usoudit na obecné a dlouhodobě přetrvávající problémy v oblasti zabezpečení webových aplikací.

6.1 SQL Injection

Při zabezpečení naší aplikace jsme se doteď věnovali pouze případům, kdy uživatel nevykonával cílenou činnost pro získání dat, jež mu nepřísluší. V této a dalších kapitolách se budeme věnovat cíleným útokům a způsobům jak jim zabránit.

Základním útokem na webové aplikace, které využívají databázi, je útok *SQL Injection*[2]. K tomuto útoku jsou náchylné všechny aplikace, kde je dotaz tvořen tímto stylem:

```
$heslo = $_POST['heslo'];
$dotaz = "SELECT name, surname
FROM uzivatele WHERE heslo = '" + $heslo + "'";
```

, přičemž heslo jsme obdrželi například z přihlašovacího formuláře. Pokud uživatel zadá jako heslo například hodnotu:

```
' OR 1=1
```

je dotaz vyhodnocen jako

```
$dotaz = "SELECT name, surname
FROM uzivatele WHERE heslo = '' OR 1=1";
```

Tímto získal útočník náhodnou identitu bez znalosti skutečného hesla. Ještě horší je situace v případě, kdy vypisujeme data z dotazu někde na stránce, což se většinou děje. V takovém případě může útočník snadno zjistit libovolný údaj v databázi, pokud vloží do formuláře hodnotu jako například:

```
" ' UNION SELECT jmeno, heslo
FROM uzivatele -- ";
```

Potom se původní kód, který měl podobu:

```
$datumGenerovaniReportu = $_GET['datum'];
$dotaz = "SELECT vyrobek, cas_dodani
FROM vyrobky WHERE cas_dodani > '" + $datumGenerovaniReportu + "' ";
```

vyhodnotí zcela jinak a proměnná *dotaz* obsahuje hodnotu:

```
$dotaz = "SELECT vyrobek,cas_dodani
FROM vyrobky WHERE cas_dodani > '' UNION SELECT jmeno, heslo
FROM uzivatele -- ' ";
```

, která způsobí, že výpis výrobků na stránce nám vypíše kromě názvů výrobků i uživatele systému [3]. Situace je o to horší, že existuje speciální databáze nazvaná information_schemas, která obsahuje popis všech ostatních databází. Díky tomu je možné získat relativně snadno strukturu databáze, jména tabulek a sloupců včetně jejich datových typů. S pomocí těchto údajů je pak možné získat z databáze libovolná data.

Za předpokladu, že výpis dotazu není zobrazován přímo uživateli, přichází na řadu technika známá jako *Blind SQL Injection*. Podstata této techniky spočívá v tom, že nestandardním způsobem ovlivníme vykonávání dotazu.

[2] Všechny příklady v této práci předpokládají vypnutou direktivu magic_quotes. Ta sice představuje jistou míru ochrany, nicméně se na ni nelze za všech okolností spolehnout a v PHP ve verzi 5.4 je navíc úplně vyřazena.

[3] Všechny příklady v této kapitole předpokládají databázi MySQL verze 5.1. V případě jiné databáze však bude situace obdobná.

Například se vrátíme k předchozímu příkladu:

```
$datumGenerovaniReportu = $_GET['datum'];
$dotaz = "SELECT vyrobek, cas_dodani FROM vyrobky
WHERE cas_dodani > '" + $datumGenerovaniReportu + "' ";
```

Útočník může do pole *datum* vložit hodnotu:

```
"' UNION SELECT IF( SUBSTR( jmeno, 0, 1 )='a',
BENCHMARK( 1000000, ENCODE( 'hello', 'goodbye') ), 1 ),jmeno
FROM uzivatele LIMIT 1 -- "
```

V případě, že první znak jména bude 'a', tak se prodlouží vykonávání dotazu a útočník získal potřebnou informaci. Takto může útočník prostou iterací získat postupně obsah celé databáze.

Obrana proti tomuto typu útoku je poměrně snadná. Spočívá v důsledném ošetřování vstupů. Jednou z cest je ošetřit veškeré vstupy příslušnou funkcí. V PHP se k tomu při využití MySQL používá funkce mysql_real_escape_string.

Lepším řešením je využití takzvaných *prepared statements*, což je technika, kterou umožňuje velká většina současných databází. V případě využití prepared statemens tvoříme dotazy v této podobě:

```
$dotaz = "SELECT vyrobek, cas_dodani FROM vyrobky WHERE cas_dodani > ? ";
```

Takto položený dotaz se následně přeloží na *prepared statement*, který přebírá další hodnoty jako parametry. Celá ukázka kódu pak vypadá následovně:

```
$dotaz = "SELECT vyrobek, cas_dodani FROM vyrobky WHERE cas_dodani > ? ";
$sth = $dbh->prepare($dotaz);
$sth->execute(array($_GET['datum']));
$vysledek = $sth->fetchAll();
```

Druhé řešení je preferováno, jelikož je snadnější a snáze se hledají chyby. Navíc je snáze přenositelné na jiné databázové platformy.

Přestože je SQL Injection jeden z nejstarších a nejznámějších útoků na webové aplikace, existuje stále mnoho webů, jež jsou proti němu nezabezpečené. Z toho důvodu jsme si jej důkladně popsali a ukázali si, jak se proti němu bránit.

6.2 XSS - Cross-site scripting

Dalším běžným typem útoku na webové aplikace je útok typu Cross-site scripting, zkráceně XSS. Tento druh útoku spočívá v pozměnění HTML kódu stránky, kdy programátor dostatečně neošetřil vstupní parametry předávané uživatelem.

Jako typickou ukázku Cross-site scriptingu si můžeme ukázat následující kód. Představme si, že někde ve stránce máme pole vyhledávacího formuláře a uživateli chceme ještě jednou ukázat co vlastně hledal:

```
Výsledky vyhledávání pro frázi:
<strong><?php echo $_GET['search'] ?></strong>
```

Pokud útočník do vyhledávacího pole vloží například hodnotu:

```
<script type="text/javascript">alert('XSS útok.');</script>
```

zobrazí se mu okno s informací o zdařeném útoku. Samozřejmě v tomto případě ohrožuje pouze sám sebe, nicméně nic mu nebrání poslat link na stránku například e-mailem ve tvaru:

```
http://example.com?serach=<script>alert('XSS útok.');</script>
```

Oběť po kliknutí na danou stránku spustí JavaScriptový kód se svými vlastními právy, což může mít neblahé následky. Jelikož tento útok nemění trvale strukturu stránky, ta je závislá na parametru v URL, hovoříme o něm jako non-persistent. Tento druh útoku se dá do určité míry odhalit obezřetným uživatelem. Pokud však útočník použije některou ze služeb, jež zkracují URL adresy na kratší formát, nemá napadený příliš mnoho šancí si útoku na první pohled všimnout.

Druhým typem zranitelnosti je persistent Cross-site scripting, kdy je neošetřená hodnota přímo uložena například v databázi. Můžeme si představit například diskuzní fórum, kdy útočník napíše příspěvek obsahující nebezpečný kód. Následně bude nebezpečný kód zobrazen všem návštěvníkům fóra.

Co je na útoku typu Cross-site scripting tak nebezpečného? Abychom toto pochopili, musíme si více vysvětlit podstatu HTTP protokolu. HTTP je protokol bezstavový, proto aby mohl server identifikovat konkrétního uživatele a pamatovat si jeho stav, se používá takzvaný identifikátor sezení, zkráceně sessionid. Tento identifikátor se většinou předává pomocí cookie a je v daném čase pro daný server jedinečný a náhodný. Díky tomuto identifikátoru je pak server schopen odlišit a identifikovat jednotlivé uživatele. Pokud útočník získá přístup k tomuto identifikátoru, nic mu následně nebrání, aby se vydával za napadenou oběť. Jak toho docílit demonstruje následující příklad.

```
<script type="text/javascript">
  document.write(
    "<img width='1px' height='1px' \
    src='https://example.com?id="+document.cookie+"' >"
  );
</script>
```

Napadenému se v prohlížeči vykreslí malý, snadno přehlédnutelný obrázek, jehož adresa bude směřovat na server útočníka. Na tomto serveru pak poběží skript, který bude logovat jednotlivé přístupy a číst z nich hodnotu cookie i s obsahem sessionid. S ním může útočník snadno získat naši identitu.

Většina moderních prohlížečů podporuje možnosti takzvaných HttpOnly cookie, což jsou cookie, ke kterým nemá JavaScript přístup. U session cookie je tedy více než vhodné tento atribut nastavit a tím k nim ztížit přístup potenciálnímu útočníkovi. Bohužel ani toto není zárukou ochrany. Například Apache server, který je nejpoužívanějším webovým serverem, poskytuje v defaultní instalaci metodu TRACE, kterou tímto doporučujeme vypnout. Podstatou této metody je, že server posílá v odpovědi na požadavek typu TRACE kopii tohoto požadavku v odpovědi. K odpovědi už má ale JavaScript plný přístup. Útočníkovi tedy nic nebrání zavolat pomocí AJAX-u TRACE metodu serveru a v odpovědi si přečíst cookie.

Zda náš sever podporuje TRACE metodu, zjistíme jednoduše například pomocí telnetu.

```
>telnet www.example.com 80
TRACE /index.html HTTP/1.1
Host: www.example.com
```

Pokud obdržíme odpověď:

```
HTTP/1.1 200 OK
Date: Sat, 17 Dec 2011 23:51:29 GMT
Connection: close
Content-Type: message/http

TRACE /index.html HTTP/1.1
Host: www.example.com
```

tak náš server podporuje metodu TRACE. Povšimněme si posledních dvou řádků, které tvoří tělo odpovědi, a obsahují kopii našeho původního požadavku. Na IIS serveru existuje alternativní metoda TRACK. Vzhledem k tomu, že dané metody se v produkčním prostředí až na vzácné výjimky téměř nepoužívají, doporučujeme je na webovém serveru zakázat úplně.

Obranou proti tomuto útoku je důkladné ošetření všech výstupů, které naše aplikace vypisuje. O něco složitější je daný problém v tom, že ošetřovat musíme vždy výstup v daném kontextu. Jinak se ošetřuje výstup v HTML atributu, jinak v HTML elementu a ještě jinak v JavaScriptovém kódu.

Další obranou je filtrování všech vstupů. Například u pole obsahujícího čas, zakážeme všechny znaky kromě čísel a dvojtečky. Poslední možností jak se bránit, je použít některý aplikační firewall. Tyto nástroje analyzují opět všechny http požadavky a pomocí heuristiky se v nich snaží vyhledat možné útoky. Za zmínku například stojí projekt PHP IDS[4], který je šířen zdarma. Problém těchto nástrojů však je, že málokdy zabezpečí 100% účinnost a naopak můžou generovat falešné poplachy.

6.3 Autentizace

Autentizace je proces, při kterém ověříme identitu uživatele. To znamená, že zjistíme, zda uživatel je ten, za koho se vydává. Pro ověření uživatele můžeme použít tyto 4 postupy:

Ověření na základě znalosti: Při tomto druhu ověření identifikujeme uživatele na základě znalosti, kterou zná pouze on. Typicky se jedná o heslo nebo pin.

Ověření na základě vlastnictví: Uživatel se prokáže vlastnictvím předmětu, například certifikátem nebo hardwarovým klíčem.

Ověření na základě samotné osoby: Do této kategorie spadá biometrické ověření jako jsou otisky prstů nebo snímek sítnice.

Ověření na základě schopnosti: Uživatel se prokáže dovedností, kterou je těžké napodobit. Například jedinečným podpisem.

[4] https://phpids.org/

Tyto metody je navíc možné libovolně kombinovat, například můžeme zkombinovat čipovou kartu a heslo nebo otisk prstu a elektronický občanský průkaz.

Při volbě autentizačních mechanizmů se snažíme volit takové, které budou uživatele co možná nejméně omezovat. Nelze například předpokládat, že by měl každý uživatel k dispozici čtečku otisků prstů. Proto je pro nás biometrické ověření jen velmi obtížně použitelné.

V prostředí webových aplikací se běžně používá ověření heslem. Tato metoda je poměrné bezpečná, je však nutné volit dostatečně dlouhá a náhodná hesla. Ideálně by mělo heslo obsahovat velké a malé písmeno, číslici a speciální znak (například pomlčku). Pro délku hesla platí, že delší heslo je obecně bezpečnější. Za obecně rozumnou délku hesla se v době psaní této práce považuje alespoň 8 znaků[5].

Poměrně rozšířenou praxí je uživatelům hesla generovat tak, aby splnila výše zmíněná pravidla. Praxe však ukazuje, že opravdu náhodná hesla si mají uživatelé problém zapamatovat. Potom nastávají situace, kdy je heslo napsané na papírku poblíž uživatelské stanice. Proto je obecně lepší nechat uživatele, ať si heslo zvolí sám, a následně ověříme, že splňuje všechny parametry bezpečného hesla. Zároveň je vhodné si ověřit, že se nejedná o nějaké běžné slovo nebo jeho variantu.

Z bezpečnostních důvodů je nutno zabezpečit, aby žádné heslo nebylo uloženo v čitelné podobě. K tomu se používají jednosměrné funkce, které spočítají otisk daného hesla, takzvanou hash. Například hash slova *heslo* spočítaný funkcí MD5[6] má hodnotu:

$$955db0b81ef1989b4a4dfeae8061a9a6.$$

Z této hashe by nemělo být možné získat původní heslo jinak než hrubou silou a zkoušením postupně všech možných kombinací.

Ověření uživatele pak probíhá způsobem, kdy se prokáže heslem, jehož hash je stejný jako je hodnota uložená v databázi. Existuje samozřejmě více slov majících stejný otisk, v takovém případě hovoříme o kolizi. Používané funkce jsou však konstruovány tak, aby pravděpodobnost těchto kolizí byla co možná nejmenší. Bezpečnost bychom navíc měli posílit takzvaným solením hesel. Což je operace, kdy zadané heslo od uživatele upravíme, například tak, že ho rozšíříme o náhodný prefix. Tento prefix pak uchováváme společně s otiskem hesla na serveru. Důvod tohoto chování je, že existují veřejně dostupné databáze s již předpočítanými hodnotami hashů pro nejčastěji používaná hesla.

Útočník pak v případě kompromitování databáze a získání otisků hesel není nucen provádět jejich dešifrování hrubou silou, ale může najít příslušné heslo již předvypočítané v dané databázi. Tyto databáze se odborně nazývají rainbow tables.

Další široce používanou možností jak autentizovat uživatele na internetu, je způsob pomocí osobního certifikátu. Tento druh zabezpečení spočívá v tom, že administrátor každému uživateli (nebo skupině uživatelů) vygeneruje certifikát, jehož vlastnictvím se následně prokazuje jeho identita. Výhodou tohoto řešení je, že drtivá většina dnes používaných prohlížečů podporuje možnost prokazovat se osobním certifikátem. Pro uživatele to tedy znamená jen nutnost importovat certifikát do prohlížeče a veškerá autentizace se následně děje automaticky. Jelikož certifikát má podobu souboru jehož velikost se v době psaní tohoto článku pohybuje v řádu kilobytů, není problém jej přenášet například na osobním flash disku a mít jej tak neustále s sebou.

[5] https://www.owasp.org/index.php/Password_length_&_complexity

[6] Nutno dodat, že MD5 v době psaní tohoto článku není považována za bezpečnou. Doporučuje se používat funkci z rodiny SHA-2

Asi nejvýhodnější je využití osobního certifikátu spolu s heslem. S tím, že ověření platnosti certifikátu bude provádět webový server a naše aplikace bude provádět ověření pouze pomocí jména a hesla. Ověření certifikátem pak bude pouze volitelnou složkou, kterou může administrátor systému zapínat a vypínat dle potřeby a vlastního uvážení.

Dále se důrazně doporučuje zapnout HTTPS, tedy šifrované HTTPS spojení, jinak útočníkovi nic nebrání v odposlechu i sebesložitějšího hesla.

6.4 Autorizace

Poté co pomocí autentizace 6.3 ověříme identitu uživatele, přichází na řadu autorizace. Autorizace je proces, kdy ověřujeme, že uživatel má právo provádět nějakou akci, případně přistupovat nebo jinak nakládat s konkrétním prostředkem.

Možností jak provádět autorizaci je více s různou granularitou. Jako příklad jednoduchého autorizačního mechanizmu si můžeme uvést práva v souborovém systému běžného operačního systému, kde systém ověřuje, kdo má právo daný soubor číst a zapisovat do něj.

Toto řešení je však poměrně omezené, například logovací nástroj, který eviduje přístupy, zaznamenává data do souboru. Proto musí mít právo pro zápis. Tím ale zároveň získává právo celý soubor přepsat. V případě, že útočník najde zneužitelnou chybu v tomto logovacím nástroji, je schopen smazat informace o své aktivitě.

Účinnějším způsobem jak řešit autorizaci jsou takzvané "Seznamy pro řízení přístupu" v originále Acces Controll List, zkráceně pak ACL.

Definice ACL se skládá z těchto prvků:

Uživatel / Skupina uživatelů / Role: Konkrétního uživatele asi netřeba vysvětlovat, rolí může být třeba vlastník, skupina uživatelů můžou být třeba studenti. Pro větší flexibilitu můžeme zavést i hierarchii rolí a systém tak zpřehlednit.

Prostředek: Například domovský adresář nebo přehled provizí v ekonomickém systému.

Oprávnění nakládat s prostředkem: Jeho editace, smazání, připsání nakonec atd.

Tímto způsobem jsme si schopni nadefinovat práva mnohem pružněji a lépe vyhovět našim potřebám. ACL je však pouze způsob jak autorizační pravidla popsat. Samostatným problémem je vynucení těchto pravidel, kdy musíme pečlivě kontrolovat přístup ke všem prostředkům, které náš systém poskytuje. Zde můžeme s výhodou využít naše architektonické řešení, kdy všechny požadavky na náš systém procházejí jedním vstupním bodem, kde jsme schopni následně provádět ověření těchto požadavků.

6.5 CSRF - Cross-site request forgery

Poslední z útoků, které si v této části představíme, je útok typu Cross-site request forgery. Na rozdíl od útoku Cross-site scripting, kdy jsme využívali chyby v nedostatečném ošetření parametrů na napadeném serveru, při tomto druhu útoku napadáme webovou aplikaci z aplikace jiné. Do této druhé aplikace nejprve vložíme nebezpečný kód například technikou XSS a následně na tuto stránku nalákáme oběť.

Nebezpečný kód může mít například formu obrázku:

```
<img width='1px' height='1px'
  src='https://example.com/delete-user/?id=1' />
```

Při renderování této stránky v prohlížeči oběti se stane následující: prohlížeč narazí na tag img s url vedoucí na https://example.com/delete-user/?id=1, vykoná tedy příslušný požadavek a k němu přiloží příslušné cookie pro daný server, mezi jinými i sessionid cookie. Cílový server vyhodnotí daný požadavek jako legitimní a provede akci delete-user, což bezpochyby není to, co si napadený uživatel přál.

První rada pro zabezpečení webových stránek tedy zní, že požadavky měnící strukturu dat na serveru by měly probíhat výhradně pomoci POST požadavků. Nicméně ani to není dostatečné, útočník může do stránky vložit skrytý formulář a ten následně odeslat pomoci JavaScriptu.

Druhým nezbytným prvkem chránící naše aplikace by tedy měl být bezpečnostní token, který aplikace generuje pro každý požadavek měnící strukturu dat, a který následně ověřuje při přijetí daného požadavku. Předpokládejme, že naše aplikace obsahuje script na adrese:

```
https://example.com/delete-user/?id=1
```

Link vygenerovaný uživateli by pak obsahoval bezpečnostní token, tedy například:

```
https://example.com/delete-user/?id=17token=edFlqSBKeSrF4F8U
```

Tento token by byl platný jenom po omezený čas a útočník by tak měl podstatně ztížené podmínky.

6.6 Zabezpečení shrnutí

Vytvořit správně zabezpečenou aplikaci není triviální problém, je to proces, který vyžaduje mnoho zkušeností a opatrnosti. Navíc i zde platí, že řetěz je tak silný, jako jeho nejslabší článek. Stačí jediné místo, kde polevíme v ostražitosti, a útočník může zneužít naší chyby. Proto jsme v této části věnovali několik stránek zabezpečení webových aplikací a popsali si nejčastější útoky na ně, popsali jsme si základy autentizace a autorizace. Spolu s útoky typu SQL injection, Cross-site scripting a Crost-site request forgery.

Obecně bychom při správném zabezpečení měli zabezpečit tyto body:

- Validovat všechny uživatelem zadané hodnoty.
- U SQL dotazů používat prepared statements, případně dbát na správné ošetření všech zadaných hodnot, jakožto obranu proti SQL injection.
- Na výstupu ošetřit všechny uživatelem zadané hodnoty - takto se budeme bránit proti XSS útokům
- Správně nastavit webový server, zabezpečit co možná nejvíce transportní vrstvu, stejně tak celé sezení.
- Do všech formulářů přidat pokud možno jednorázový token jako obranu proti CSRF.

Samostatnou kapitolu by zde mohla tvořit část věnovaná sociálnímu inženýrství, nicméně to by již přesahovalo rozsah této práce. Za zmínku zde však stojí kniha [12], která poskytuje právě popis těchto útoků a způsobů jak se proti nim bránit.

Kapitola 7

Testování aplikace, srovnání a výsledky z provozu

Po vytvoření vlastní aplikace jsme se ji rozhodli co možná nejlépe otestovat. Samotné testování jsme rozdělili do skupin. Jsou to tyto 4:

Testování funkčnosti aplikace: V průběhu tohoto testování jsme se snažili ověřit, zda aplikace dělá to, co dělat má, a nevykazuje chyby.

Testování výkonu aplikace: Zde jsme navázali na kapitolu 4 a snažili se ověřit, že naše aplikace poskytuje dostatečný výkon. Vzhledem k tomu, že jako backend využíváme již existující architekturu aplikace nfdump, nemusíme se starat o samostatné ukládání NetFlow dat. Je však nutné otestovat, zda naše aplikace nepřinesla nepřijatelnou režii a zda má tedy smysl ji používat v reálném provozu.

Testování zabezpečení aplikace: V návaznosti na kapitolu 6 jsme provedli testy zabezpečení naší webové aplikace. Více se tomuto tématu budeme věnovat v sekci 7.2.

Uživatelské testování aplikace: V této části naší práce jsme provedli test ergonomie uživatelského rozhraní na vybrané skupině uživatelů. Podrobnosti jsou detailněji popsány v části 7.3.

7.1 Testování výkonu

V této části provedeme testování výkonu. Naším cílem bude změřit dodatečnou režii, kterou naše aplikace přináší, a v případě potřeby navrhnout dodatečné optimalizace.

7.1.1 Metodika pro testování výkonu

Testovat budeme na dotazech, jež jsme využili v kapitole 4. Tedy konkrétně se bude jednat o tyto:

1. **Základní dotaz na komunikaci IP adresy.** Následující dotaz zobrazí komunikaci IP adresy 2001:67c:1220:e000::93e5:30a.

```
nfdump "host 2001:67c:1220:e000::93e5:30a"
```

2. **Filtrace dle prefixu.** Následující dotaz zobrazí komunikaci, kde zdrojová nebo cílová IP adresa náleží do podsítě 2001:67c:1220:e000::/56.

```
nfdump "net 2001:67c:1220:e000::/56"
```

3. **Filtrace prefixu + agregace podle zdrojové adresy.** Následující dotaz zobrazí komunikaci, kde zdrojová nebo cílová IP adresa náleží do podsítě 2001:67c:1220:e000::/56. Výsledek je následně agregován dle zdrojové IP adresy.

```
nfdump -a -A srcip "net 2001:67c:1220:e000::/56"
```

4. **Komunikace z vybraného prefixu do vybrané cílové sítě.** Následující dotaz zobrazí komunikaci, kde zdrojová IP adresa náleží do podsítě 2001:67c:1220:e000::/56 a cílová adresa náleží do podsítě 2001:718::/32. Výsledek je následně agregován dle zdrojové a cílové IP adresy.

```
nfdump -a -A srcip,dstip  "src net 2001:67c:1220:e000::/56
and dst net 2001:718::/32"
```

5. **Komunikace vybraných adres na cílový port 80 (http).** Následující dotaz zobrazí komunikaci, kde zdrojová IP adresa náleží do podsítě 2001:67c:1220:e000::/56 a cílový port má hodnotu 80. Výsledek je následně agregován dle zdrojové IP adresy.

```
nfdump -a -A srcip  "src net 2001:67c:1220:e000::/56
and dst port 80"
```

6. **Struktura protokolu na jednom rozhraní.** Následující dotaz zobrazí komunikaci na síťovém rozhraní s ID 1. Výsledek je následně agregován dle použitého protokolu.

```
nfdump -a -A proto  "in if 1"
```

7. **Objemy odchozích dat dle prefixu pro vybranou síť.** Následující dotaz zobrazí komunikaci, kde zdrojová IP adresa náleží do podsítě 2001:67c:1220::/48. Výsledek je následně agregován dle prefixu podsítě.

```
nfdump -a -A srcip6/64  "src net 2001:67c:1220::/48"
```

8. **Objemy odchozích dat dle prefixu pro vybranou síť a následné seřazení jednotlivých prefixů podle objemu přenesených dat.** Stejný dotaz jako v předchozím příkladu, výsledek je však seřazen dle objemu přenesených dat.

```
nfdump -a -A srcip6/64  -w -  "src net 2001:67c:1220::/48"
|  nfdump -n 0 -s srcip/bytes
```

Operační systém	Linux coyote.cis.vutbr.cz 2.6.32-131.6.1.el6.x86_64
CPU	x86_64 Intel(R) Xeon(R) CPU X5650 @ 2.67GHz * 12
RAM	64Gb
HD	6 * DELL 600GB SAS 6Gbps 15k 3.5"v Raid 5

Tabulka 7.1: Konfigurace serveru Coyote.

Pro testování opět využijeme server Coyote, jehož konfiguraci jsme popsali dříve, nicméně ji uvedeme znovu v tabulce 7.1. Jako testovací data zvolíme údaje nasbírané v síti VUT Brno od 1.5.2012 00:00 do 1.5.2012 12:00. Rychlost utility nfdump budeme měřit pomocí utility *time*. Rychlost naší aplikace budeme měřit od přijetí HTTP požadavku po vygenerování odpovědi. Vyloučíme tak samotnou latenci, kterou přináší HTTP komunikace mezi webovým prohlížečem a serverem. V případě generování HTML výstupu nám půjde o generování pouze první stránky, jež obsahuje prvních 20 záznamů výsledku.

7.1.2 Výsledky z testování výkonu

Výsledky našeho měření shrnuje tabulka 7.2. Výsledky vnímáme jako velmi pozitivní.

	nfdump	Nefelé, HTML zobrazení	Nefelé, export do CSV	počet výsledků
dotaz č.1	42.88 s	43.09 s	46.91 s	83 374
dotaz č.2	43.51 s	43.90 s	57.99 s	285 926
dotaz č.3	43.98 s	44.20 s	44.28 s	4 679
dotaz č.4	43.98 s	44.14 s	44.27 s	83
dotaz č.5	42.04 s	42.21 s	42.39 s	0
dotaz č.6	43.39 s	43.49 s	43.75 s	0
dotaz č.7	42.98 s	43.18 s	43.20 s	67
dotaz č.8	43.55 s	43.88 s	NA	67

Tabulka 7.2: Výsledky měření výkonu naší aplikace. NA v posledním dotazu značí, že daný dotaz nelze provést, což je umělé omezení naší aplikace.

V první variantě, kdy zobrazujeme podmnožinu výsledku v podobě HTML tabulky, je rychlost velice dobrá, aplikace velice rychle reaguje i na přecházení mezi stránkami. Toho jsme dosáhli uložením výsledků do dočasného souboru a následným procházením tohoto souboru pomocí utilit *head* a *tail*, které se ukázaly velmi vhodné pro tento účel a jsou schopny efektivním způsobem zpracovávat i velmi objemné soubory. Abychom však šetřili místem na disku, je tento soubor automaticky mazán. Pokud tedy s výsledkem nebude delší čas pracováno, bude muset uživatel opět čekat na jeho úplné vygenerování.

O něco hůře dopadl export do CSV formátu. Zde již má naše aplikace drobné nedostatky, nicméně věříme, že časy jsou pořád dostatečné. Za povšimnutí stojí zejména fakt, že zatímco nfdump podává veskrze stabilní výsledky, což jsme si ověřili již v kapitole 4, naše aplikace je závislá na počtu výsledků.

Dále je nutno upozornit, že naše aplikace neumožňuje setřídit exportovaná data. Toto omezení je spíše umělé, jelikož v průběhu práce jsme nenašli uživatelsky přívětivý způsob, jak v průběhu exportu definovat řazení záznamů. Nicméně vzhledem k tomu, že daný export

bude ve většině případů sloužit pro import do jiného systému, který toto řazení pravděpodobně podporovat bude, nepovažujeme tento nedostatek za zásadní. Spíše upozorníme na to, že utilita nfdump, pro kterou jsme připravili frontend, řazení podporuje pouze na některých sloupcích. My jsme toto omezení do jisté míry odstranili a jsme schopni velmi efektivně řadit téměř dle všech sloupců výsledku.

7.1.3 Shrnutí výsledků z testování výkonu

Obecně jsme s výkonem naší aplikace, přestože by mohl být dozajista vyšší, spokojeni a považujeme dodatečnou režii za jistou daň za komfort. Navíc jsme přidali některé možnosti, které *nfdump* nemá, což se na výkonu do určité míry musí projevit. Řešením jak dále zrychlit naši aplikaci, by mohlo být přepsání kritických části naší aplikace do jiného, výkonnějšího jazyka. Toto ale prozatím považujeme za předčasnou optimalizaci, jelikož bychom se pravděpodobně museli vzdát snadné udržovatelnosti naší aplikace.

7.2 Testování zabezpečení

V kapitole 6 jsme si popsali a ukázali nejčastější útoky na webové prezentace. Získané znalosti jsme se pokusili využít v průběhu naší práce. Jistotu, že naše aplikace je opravdu bezpečná, bohužel, z povahy bezpečnosti, nikdy nezískáme. Je možné se dotazovat, zda konkrétní část aplikace obsahuje konkrétní bezpečnostní chybu. Nicméně nejsme schopni obsáhnout všechny druhy chyb, které existují, například už z důvodu, že typy útoků se mění a vyvíjejí.

Přesto je naším cílem mít v naší práci co možná nejvyšší míru důvěry. Proto je potřeba software kontrolovat, zda vyhovuje bezpečnostním potřebám a předpisům (má-li naše organizace nějaké). Tyto předpisy se neustále vyvíjejí z důvodu nových útoků a hrozeb, které průběžně vznikají, a v době psaní naší aplikace se s nimi ještě nepočítalo, proto je nutné tuto kontrolu provádět opakovaně.

Existují různé způsoby testování softwaru, od kontroly zdrojových kódů, ať už ruční, nebo automatické, po penetrační testování, kdy se v roli útočníka snažíme napadnout naši aplikaci. V naší práci se zaměříme hlavně na automatickou kontrolu, jelikož její výstupy jsou dobře měřitelné a reprodukovatelné.

Nejprve se zaměříme na statickou analýzu kódu. K tomu slouží nástroje obecně nazývané *lint*. Tyto nástroje procházejí zdrojový kód a pomocí heuristiky hledají potenciálně nebezpečné konstrukce. Výsledky těchto nástrojů nejsou zdaleka stoprocentní, hlavně v prostředí dynamicky typovaných jazyků, kdy je velmi obtížné, často až nemožné, odhadnout datový typ proměnné. Přesto jsou tyto nástroje užitečné, jelikož je lze snadno integrovat do vývojového procesu, například jako automatická kontrola, která neumožní uložit soubor, který obsahuje chyby. My jsme využili kontrolu vestavěnou v IDE[1] NetBeans.

Pod pojmem penetrační testování se rozumí snaha napadnout systém způsobem, který by pravděpodobně použil útočník. Jedná se o proces, kdy zkoumáme, zda aplikace správně ošetřuje vstupy a jak se zachová, pokud jí předáme na vstup nepředpokládaný argument.

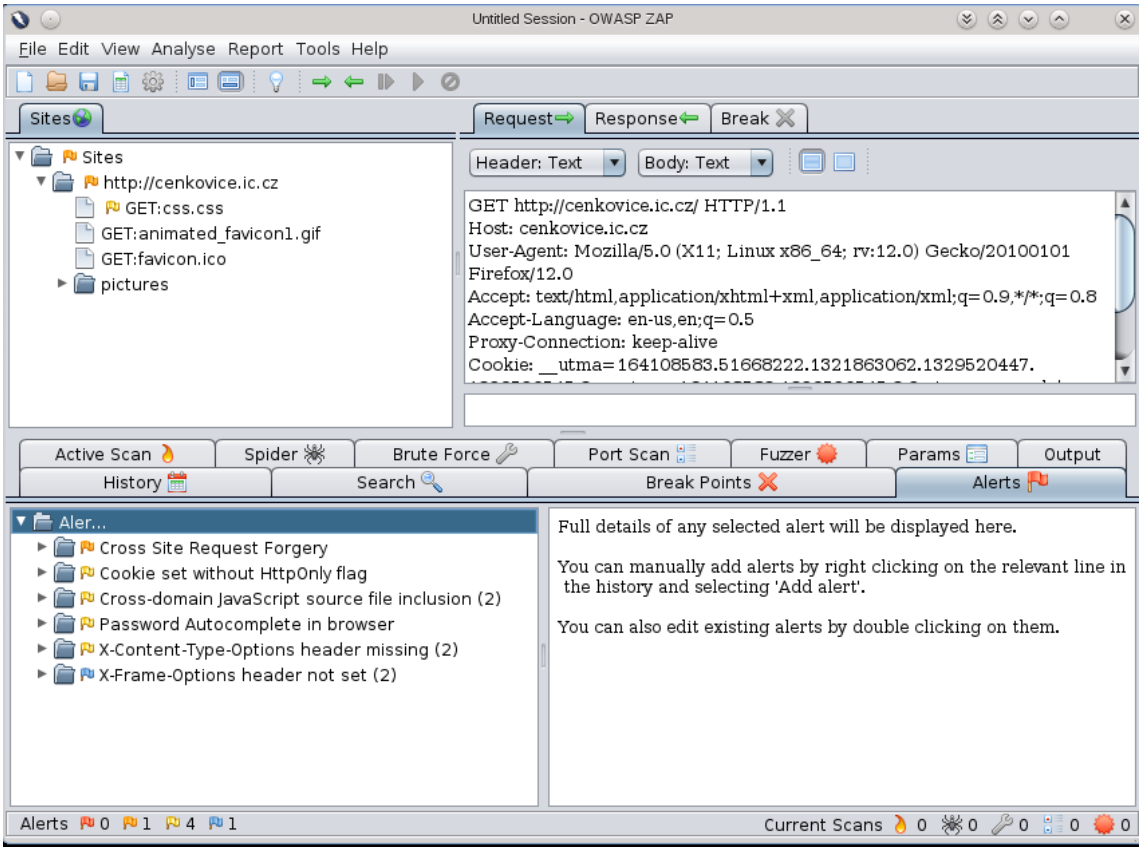
Penetrační testování je proces velmi náročný a v závislosti na velikosti aplikace může trvat i velmi dlouhou dobu. Pro jeho zjednodušení existují pomocné automatické nástroje,

[1] Integrated Development Environment

například komerční Nessus[2], Acunetix[3], nebo zadarmo šířený OWASP Zed Attack Proxy[4].

7.2.1 Způsob testování

Pro testování naší aplikace jsme zvolili OWASP Zed Attack Proxy ve verzi 1.4.0.1. Práce s tímto nástrojem probíhá následujícím způsobem. Po instalaci a spuštění se OWASP Zed Attack Proxy začne chovat jako proxy v základní instalaci fungující na portu 8080. Nastavíme námi používaný internetový prohlížeč tak, aby k práci využíval právě tuto proxy, a navštívíme stránku, kterou chceme testovat. Výsledek vidíme na obrázku 7.1.



Obrázek 7.1: OWASP Zed Attack Proxy: Zachycení procházející komunikace.

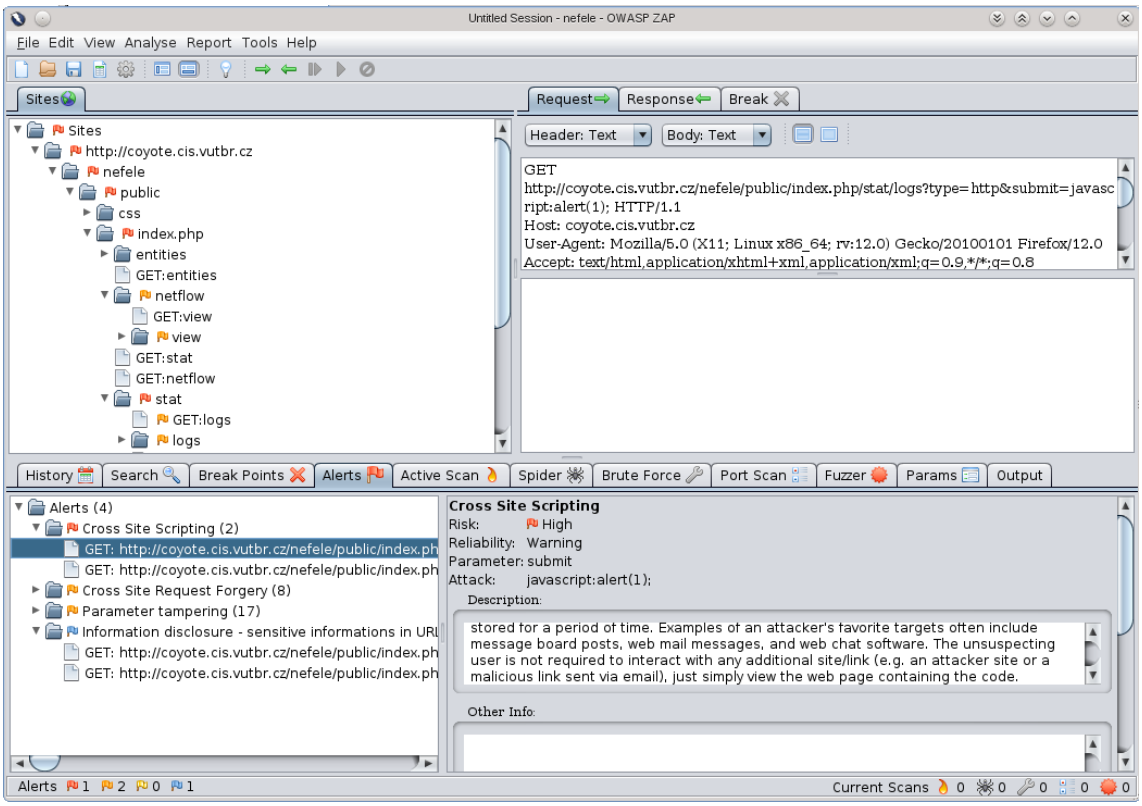
Aplikace nás sama upozorní, pokud na dané stránce objeví bezpečnostní chybu. Takto bychom mohli postupně otestovat celou aplikaci, nicméně by to bylo velmi zdlouhavé a zdaleka bychom neobsáhli všechny možné druhy útoků. Většina penetračních nástrojů z tohoto důvodu umožňuje provádět i automatické testy, dle předpřipravených pravidel. Tato pravidla jsou uchovávána v samostatných souborech a je možné jejich definice upravovat k našim potřebám. Výhodou aplikace OWASP Zed Attack Proxy je, že obsahuje již předpřipravená pravidla pro různé druhy útoků, ze kterých je možné vybírat. Tato pravidla neobsahují pouze nejčastější známé typy útoků, ale i ty méně často se vyskytující, což nám ušetří nemálo práce. I my jsme se rozhodli využít jednu z těchto sad předdefinovaných

[2] http://www.tenable.com/products/nessus
[3] http://www.acunetix.com/
[4] https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

útoků, konkrétně tu, jenž je obsažena v souboru *directory-list-1.0.txt*, který je distribuován spolu s programem.

7.2.2 Výsledky testování

Jak vypadá výsledek testování zachycuje obrázek 7.2.



Obrázek 7.2: OWASP Zed Attack Proxy: Výsledek testování. V naší aplikaci můžeme najít 4 potenciální zranitelnosti. Jedná se o cross site scripting, cross site request forgery, vyzrazení citlivé informace prostřednictvím URL a možnost manipulovat s GET parametry.

Potěšujícím výsledkem je, že v naší aplikaci nebyly nalezeny žádné zásadní bezpečnostní nedostatky, až na jeden, který je popsaný dále. Mezi výsledky se však nacházela některá varování, která jsme však vyhodnotili jako neopodstatněná.

Shrnutí nalezených chyb a varování:

Chybějící token chránící proti CSRF u některých formulářů. Konkrétně se jednalo o formuláře fungující jako filtry nad daty. Vzhledem k tomu, že v těchto případech nedochází ke změně dat na serveru, se nejedná o příliš velkou nebezpečnost. To nejhorší, čeho by útočník mohl dosáhnout, by byl DoS[5] útok, kdyby po systému chtěl příliš velké množství reportů. Což je potenciálně náročná operace. Z tohoto důvodu jsme pro jistotu i do těchto formulářů přidali bezpečnostní token.

[5]Denial-of-service - odepření služby na základě přetížení

Potenciální vyzrazení citlivých údajů z URL adresy. Toto varování jsme vyhodnotili jako neopodstatněné. Při podrobné analýze vyšlo najevo, že podezření vzbudila příliš dlouhá a komplexní www adresa, kterou náš systém generuje.

Potenciální cros-site scripting útok. I toto nebezpečí jsme vyhodnotili jako neopodstatněné. Podezření vzbudila možnost do systému vložit položky s názvy jenž obsahují javascriptový kód. Přestože souhlasíme s tím, že položka se jménem $< script > alert('XSS') < /script >$ není úplně běžnou, v případě, že systém správným způsobem ošetřuje výstupy, není ničím závadná. Naopak by pro uživatele mohlo být omezující, kdybychom mu podobné jméno zakázali.

Možnost manipulovat s GET parametry. Důvod proč penetrační nástroj reportoval tuto zranitelnost byl ten, že testovaná aplikace byla nastavena v módu vypisování ladících výpisů. Při úpravě parametrů předávaných v URL tak aplikace vypsala chybové hlášení i s příslušným ladícím výpisem. Toto bylo detekováno jako potenciální vyzrazení tajné informace. Tuto zranitelnost tedy vnímáme jako opodstatněnou. Při přepnutí do produkčního módu a opakování testů však již k této chybě nedocházelo.

7.3 Uživatelské testování

Zadáním naší práce bylo vytvořit interaktivní webové rozhraní pro práci s NetFLow daty. Toho jsme dosáhli, nicméně, aby bylo dané rozhraní co možná nejpřínosnější, rozhodli jsme se je podrobit také uživatelskému testování. Uživatelské testování je možno provádět různými způsoby. Jmenujme například A/B testování, kdy různým skupinám uživatelů zobrazíme různou variantu stránek a měříme, jak dlouho které skupině trvá zadaný úkol. Takto jsme schopni zvolit variantu uživatelského rozhraní díky které dosáhli účastníci nejlepších výsledků.

7.3.1 Metodika uživatelského testování

My jsme při našem testování zvolili metodiku, kdy jsme uživatele posadili před náš systém a nechali je vykonávat zadané úkoly. Přitom jsme pozorovali a zaznamenávali jejich činnost. Naším cílem bylo vytvořit uživatelské prostředí co nejintuitivnější, a to až do takové míry, že by uživatel neměl být nucen studovat manuál. Uživatelé proto nebyli dopředu seznámeni se vzhledem ani přesnými parametry systému. K dispozici měli pouze toto zadání:

```
V prohlížeči před vámi je přihlašovací obrazovka systému pro práci s daty
popisujícími síťový tok ve formátu NetFlow.

Vašim úkolem je:
1) Přihlásit se pomocí: uživatelského jména: uzivatel1 a hesla: TajneHeslo1

2) V "pohledu" nazvaném HTTP spojení vyhledat komunikaci se zdrojovou
IP adresou 209.237.137.10 v časech 1.5.2012 9:00 až 10:00
a zobrazit daná data jako HTML tabulku.

3) V "pohledu" nazvaném HTTP spojení vyhledat komunikaci se zdrojovou
IP adresou 209.237.137.10 v časech 1.5.2012 9:00 až 10:00
a data exportovat ve formátu CSV.
```

```
4) V "pohledu" nazvaném UDP spojení vyhledat agregovaná data dle zdrojové
podsítě v prostředí IPv4 a délkou masky 24 bitů v časech 1.5.2012 9:00 až 10:00
a zobrazit daná data jako HTML tabulku.

5) Na volné místo pod touto otázkou vepište podrobné informace
o daném pohledu (jaká je jeho definice, rozsah dat, pro které je platný,
případně další relevantní informace)
```

Jedná se tedy o typický případ užití běžného uživatele, tak, jak jsme si jej představili v 5.1. Po splnění všech úkolů jsme uživatelům položili následující otázky:

1. Bylo zadání nejasné?
2. Která část zadání vám činila největší problém?
3. Co byste v dané aplikaci udělali jinak?

Testování probíhalo na sestavě popsané v tabulce 7.3 a zúčastnilo se jej celkem 8 osob. Protože koncoví uživatelé naší aplikace budou technicky vzdělaní uživatelé, i my jsme jako testované subjekty zvolili technicky vzdělané osoby z řad studentů Vysokého učení technického v Brně, převážně pak studenty fakulty informatiky.

Operační systém	OpenSuse 12.1 64bit
Internetový prohlížeč	Chromium 20.0.1094.0 (svn131123)
Rozlišení obrazovky	1680 * 1080

Tabulka 7.3: Testovací sestava, na níž byly provedeny testy uživatelského rozhraní. Přestože testy byly provedeny v prohlížeči Chromium, aplikace je funkční i v dalších běžně používaných prohlížečích. Testovány byly Chrome 18, Chromium 20, Firefox 12 a Internet Explorer 9

7.3.2 Výsledky uživatelského testování

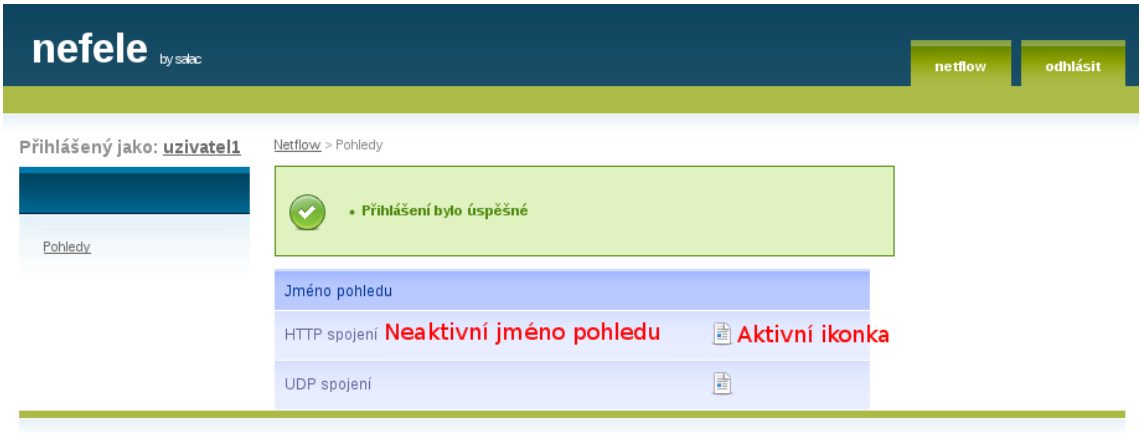
Pozorování testovaných uživatelů jsme shrnuli prostřednictvím přehledu:

Úkol č.1 : S přihlášením do systému neměli uživatelé žádný problém. Drobný problém však nastal při přechodu do požadovaného pohledu. Většina uživatelů se nejprve pokoušela kliknout na jméno daného pohledu, které je však neaktivní. Pro přesun na detail sloužila malá ikonka tak, jak je znázorněno na 7.3.

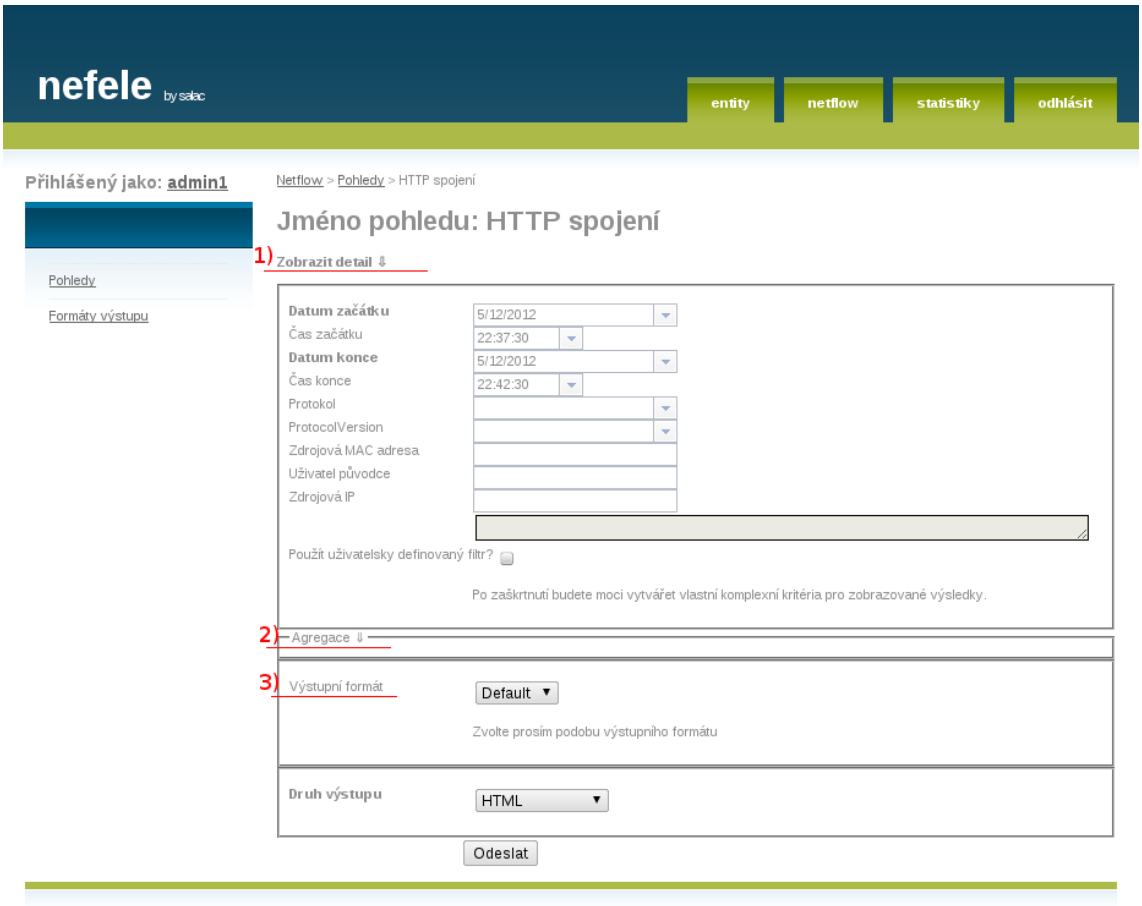
Úkol č.2 : Se zobrazením daných dat formou HTML tabulky neměli uživatelé žádný problém.

Úkol č.3 : S exportem daných dat do CSV formátu neměli uživatelé žádný problém.

Úkol č.4 : Zde nastal první vážný problém. Někteří uživatelé nebyli schopní nalézt volbu pro agregaci daných údajů. Tato volba tedy nebyla dostatečně výrazná (viz 7.4). Po ukázání příslušného prvku v uživatelském rozhraní však neměli problém.



Obrázek 7.3: Původní uživatelské rozhraní naší aplikace. Zobrazení definovaných pohledů.



Obrázek 7.4: Původní uživatelské rozhraní naší aplikace. Za povšimnutí stojí příliš malé a nejasné ovládací prvky 1 a 2. A nejasný význam prvku 3.

Úkol č.5 : Stejná situace jako v úkolu č.5. Příslušný prvek uživatelského rozhraní je příliš malý. Po jeho ukázání však uživatelé neměli problém.

Odpovědi na otázky, které jsme položili uživatelům po vykonání testu, byly následující:

Bylo zadání nejasné? Zde se všichni účastnící shodli na tom, že zadání bylo dostatečně jasné.

Která část zadání vám činila největší problém? V rovných 50 % případů byla odpovědí otázka č.4., v 62.5 % případů pak byla problematická i otázka č.5

Co byste v dané aplikaci udělali jinak? Zde byly odpovědi pestřejší, nejčastější připomínky směřovaly k nevýrazným prvkům, jež skrývaly některé části uživatelského rozhraní. Také se vyskytly připomínky k nejasnému významu některých formulářových polí. Konkrétně se jednalo o prvky *Výstupní formát* a *Obecný dotaz*. Dále panovala obecná nespokojenost s nemožností přesunout se do detailu pomocí jména pohledu.

7.3.3 Závěry z uživatelského testování

Provedené uživatelské testování nám pomohlo identifikovat některé nedostatky našeho původního rozhraní. Po pozorování reakcí uživatelů v průběhu testu a vyhodnocení dotazníku, které jsme jim položili, jsme usoudili na dva základní nedostatky naší aplikace:

- Některé prvky uživatelského rozhraní nejsou dostatečně výrazné.
- V seznamech nelze využít k přechodu do detailu jméno položky, ale pouze ikonku, jež je navíc neintuitivně umístěna.

Ani jedna ze závad nebrání uživateli pracovat s našim systémem, nicméně je vnímáme jako oprávněné připomínky, a proto jsme z nich vyvodili následující změny:

- Zvětšení příslušných ovládacích prvků pro zobrazení a skrytí částí uživatelského rozhraní.
- Přidali jsme popis s nápovědou k některým ovládacím prvkům.
- Umožnili jsme využít jméno položky v seznamech, k přesunu na detail dané položky.

Všechny tyto prvky jsme následně implementovali a výsledek je patrný na obrázku 7.5.

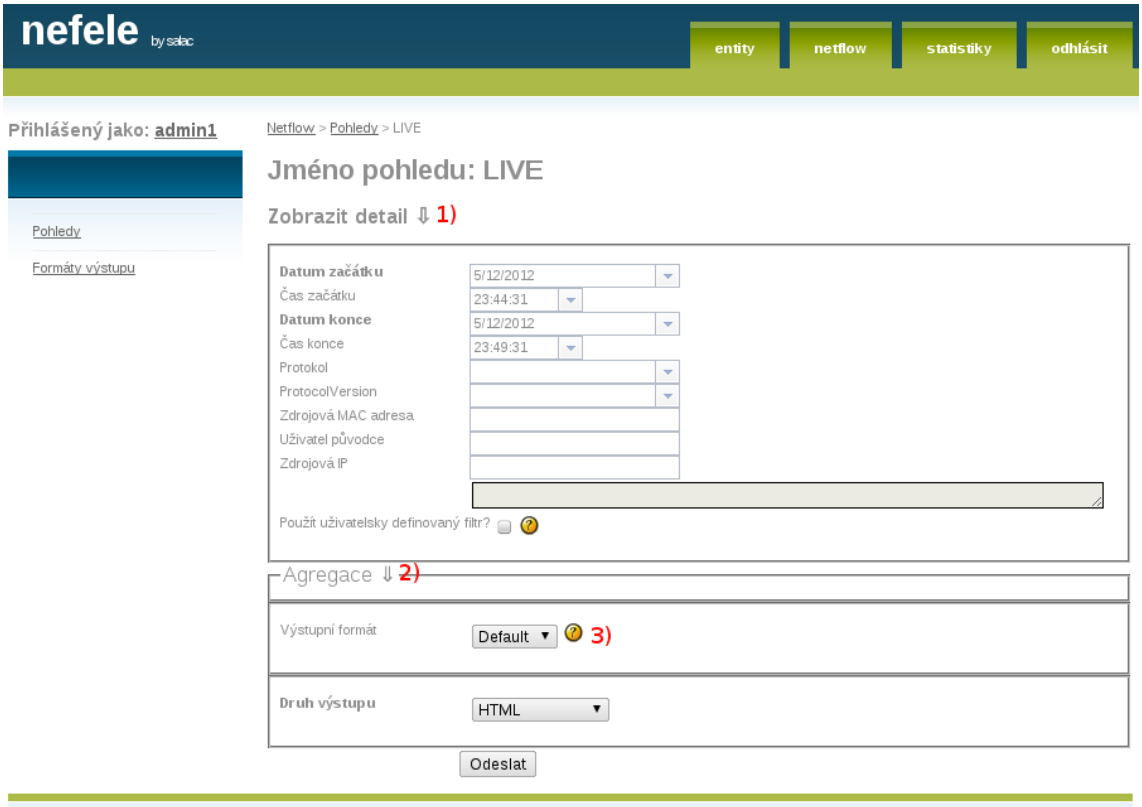
7.4 Shrnutí

V rámci této kapitoly jsme se snažili naši aplikaci podrobit některým testům. Slibujeme si od toho dosazení lepší kvality výsledné aplikace, a to jak po stránce uživatelské přívětivosti, tak po stránce výkonu a zabezpečení.

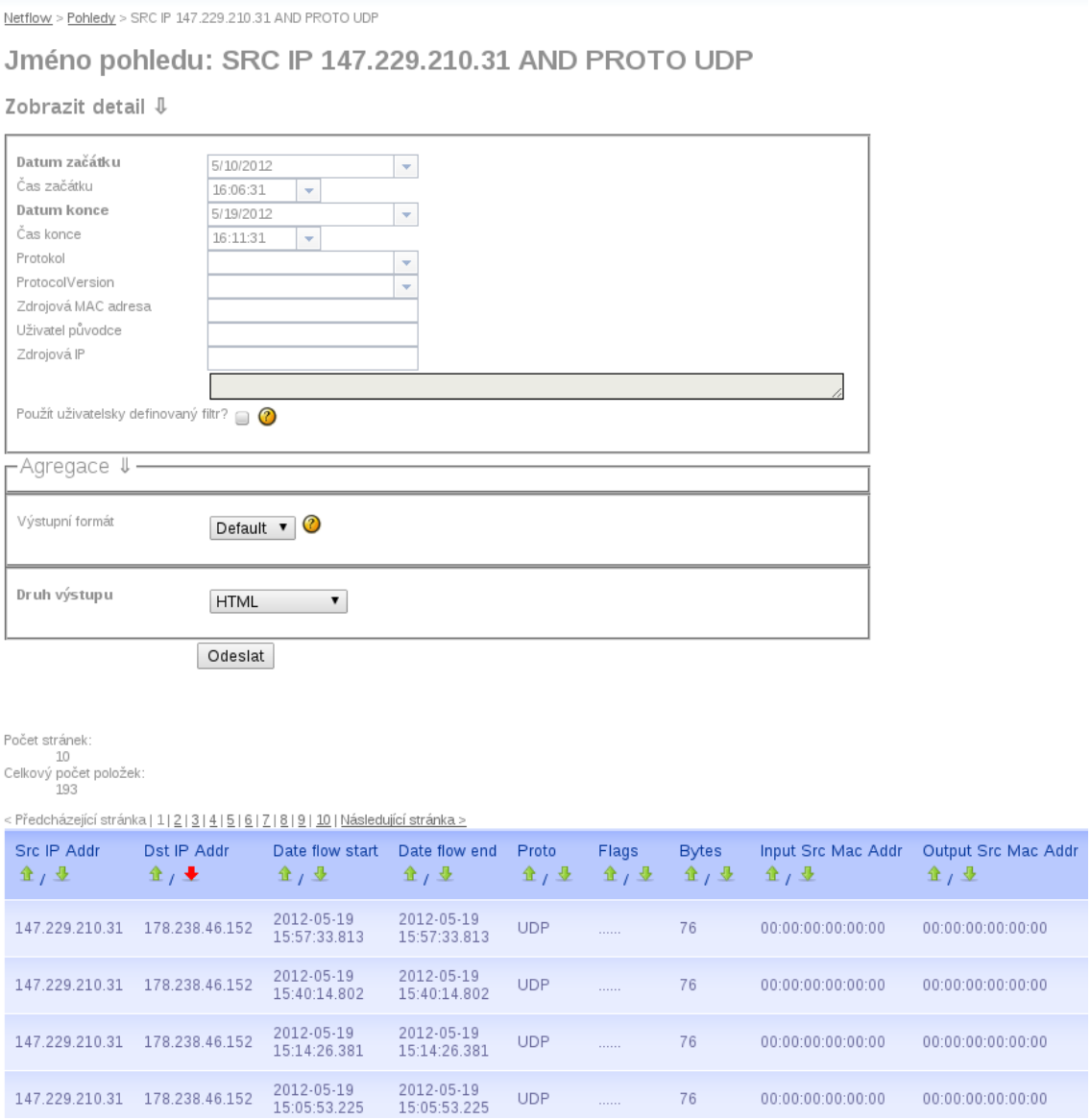
Výsledky testů dle našeho názoru dopadly kladně. Při testování uživatelského rozhraní jsme nalezli několik nedostatků, nicméně ty jsme následně opravili jak je popsáno v sekci 7.3.

V části věnované testování výkonu jsme ověřili, že režie, kterou má naše aplikace při zpracování dat o síťovém toku, nedosahuje přehnaně vysokých hodnot. Tím jsme se snažili zajistit komfort pro koncového uživatele, který nebude muset přehnaně dlouho čekat na zpracování svých požadavků.

I penetrační testování, kterému jsme naší aplikaci podrobili, dopadlo dle našeho názoru velmi pozitivně. Věříme proto, že naše aplikace je rozumně zabezpečená proti všem běžným útokům.



Obrázek 7.5: Upravené uživatelské rozhraní naší aplikace. Zvětšené prvky 1 a 2. Nápověda k prvkům je nově jako vyskakovací dialog nad otazníkem 3, což nám umožnilo zvětšit rozsah textu v nápovědě.



Obrázek 7.6: Ukázka naší aplikace zobrazující data seřazená dle cílové IP adresy.

Kapitola 8

Závěr

V této práci jsme se hlouběji seznámili s problematikou monitoringu síťového provozu. V kapitole 2 jsme popsali základní protokoly pro popis síťového toku, založené na IP flow. Se dvěma z nich, konkrétně NetFlow a IPFIX, jsme se seznámili důkladněji.

Vysvětlili jsme si také základní schéma práce s daty pro popis síťového toku. To se skládá z měřící sondy, kolektoru, který sbírá z této sondy data, aplikace řídící samotnou sondu a vlastní analytické aplikace, která z daných dat získává relevantní údaje. Jedním z problémů, se kterým se musí kolektor vypořádat, je způsob uložení údajů ze sondy. Na jedné straně je potřeba uložení všech přijatých dat v co možná nejkratším čase a beze ztrát, na druhé straně je nutno tato data uložit takovým způsobem, aby k nim analytická aplikace měla snadný a rychlý přístup.

V souvislosti s tímto jsme provedli řadu experimentů, jež jsou popsány v kapitole 4. Cílem těchto experimentů bylo prozkoumat možnost ukládat tato data v databázích. Ukázalo se však, že žádná z námi testovaných databází nesplňuje potřebná kritéria. Těmi byly v prvé řadě rychlost importu a následně větší rychlost prováděných dotazů ve srovnání s utilitou nfdump. K tomu nám měly pomoci hlavně indexy a další pokročilé možnosti databázových úložišť. Bohužel žádná z testovaných databází se neukázala jako dostatečně výkonná a stabilní, abychom ji mohli bez výhrad doporučit. Mezi největší problémy patřila doba vytváření indexu nad danými daty a stabilita pod zátěží.

Nakonec, po zvážení všech pro a proti, jsme se tedy rozhodli využít samotný nfdump a vytvořit nad ním vrstvu pro práci více uživatelů. Předtím jsme ale prozkoumali již existující nástroje pro analýzu síťového toku. Převážně pak ty, jež se zabývají daty ve formátu NetFlow.

U těchto nástrojů jsme se snažili najít jejich silné a slabé stránky a získané znalosti následně uplatnit při vývoji aplikace vlastní. Tomu jsme se věnovali v kapitole 3.

Zde musíme konstatovat, že počet existujících nástrojů nás mírně zklamal. Většina těchto nástrojů již nebyla dále vyvíjena, byla komerční povahy, nebo byla svázána s konkrétní platformou a nesplnila tak jednu z podmínek naší práce, kterou byla funkčnost na *nixovém serveru. Zaměřili jsme se tedy pouze na ty nástroje, které byly dostupné zdarma a byly aktivně vyvíjeny, zbytek jsme se snažili co nejlépe nastudovat z dostupné dokumentace.

Po nastudování těchto nástrojů jsme si vytkli cíl vytvořit aplikaci, která by tyto nástroje doplňovala. Existuje totiž několik málo oblastí, do nichž současné aplikace nezasahují vůbec, nebo jen ve velmi omezené míře.

Jednou z těchto oblastí je podpora pro jednorázové dotazy. Existující nástroje často generují souhrnné statistiky a určují například distribuci jednotlivých protokolů, už ale neposkytnou informaci, jakými protokoly komunikovala jedna konkrétní IP za poslední dva dny. Na tuto a další otázky by měla odpovědět právě naše aplikace. Při jejím návrhu jsme se nechali inspirovat v některých rysech již existujícími aplikacemi. Jedná se hlavně o myšlenku předpřipravených dotazů, jež v naší aplikaci nazýváme *pohledy*.

S touto myšlenkou jsme se setkali v aplikaci NfSen, kde se tyto dotazy nazývají profily. Avšak data z těchto profilů získaná jsou opět agregovaná a mají tudíž poněkud jiný účel. Nedostatkem NfSenu je navíc to, že profil není možné odvodit z profilu jiného. Není tedy možné vytvořit například profil *TCP* a z něj odvodit profil *TCP AND PORT 80*, tato vlastnost by přitom umožnila zrychlit některé operace. Systém by pak nemusel neustále procházet úplnou množinu dat, což by se pozitivně projevilo na výkonu.

V naší aplikaci jsme se tohoto omezení zbavili a odvození z jiného *pohledu* je tedy možné. Navíc lze takovýto *pohled* zpřístupnit dalším uživatelům systému, což shledáváme obzvláště užitečné ve chvíli, kdy se systémem pracuje větší množství uživatelů. Například tak můžeme rozdělit síť na podsítě a každému správci zpřístupnit pouze data, která mu náleží. Tato možnost, řídit přístup uživatelů k přesně ohraničeným datům, je dle našeho názoru velmi užitečná a ne všechny aplikace, které jsme zkoumali, toto umožňují. Při implementaci výše popsaného jsme navíc dbali v maximální možné míře na zabezpečení, tak jak je popsáno v kapitole 6 a naši implementaci jsme ověřili penetračním testováním v sekci 7.2.

Vzhledem k tomu, že některé operace jsou pomocí utility nfdump poměrně problematické, rozhodli jsme se je vyřešit v rámci naší aplikace, a uživateli tak přinést dodatečný komfort. Jedná se například o rozšířenou možnost seřadit výsledek, jelikož utilita nfdump je v tomto velmi omezená.

Také jsme přidali možnost tato data dále exportovat do několika běžných formátů, včetně možnosti komprese, a to velmi snadnou a uživatelsky přívětivou formou. Do výčtu formátů jsme zahrnuli i formát určený pro ministerstvo vnitra dle zákona č. 485/2005 sbírky zákonů[1]. Tento zákon byl sice v nedávné době nálezem ústavního soudu zrušen[1], nicméně v tuto chvíli se však připravuje novela zákona i prováděcí vyhlášky a lze očekávat, že formát exportu požadovaný tímto zákonem se nebude od původního příliš lišit.

Součástí naší aplikace je také napojení na externí databázi Nav. Zde jsme využili práci pana Bc. Miroslava Šoltese, který pro nás upravil kolektor *nfcapd*. Úprava kolektoru spočívala v doplnění do exportovaného souboru ID uživatele, od kterého nebo ke kterému daný síťový tok směřuje, a to za předpokladu, že jsme schopni tohoto uživatele identifikovat. My jsme pak na základě tohoto ID schopni filtrovat data a zobrazit pouze toky náležící danému uživateli. Více je o této prací napsáno v [9]. Když už jsme zmínili práci pana Bc. Miroslava Šoltese, upozorníme i na jeho úpravy nfdumpu, díky kterým naše aplikace umožňuje zobrazení informací o průběhu zpracování daného dotazu. To vnímáme opět jako přínos pro uživatele.

Jak jsme psali v kapitole 7 věnované testování, naše aplikace v tuto chvíli uspokojuje naše potřeby. Velice nás těší výkon, kterého se nám podařilo dosáhnout při zobrazení v podobě HTML tabulky, a rychlost exportu považujeme také za dostatečnou. Nicméně pokud máme být kritičtí, prostor ke zlepšení tu bezesporu je. To by mohl být i jeden ze směrů, kterými by se mohl ubírat další vývoj. Zde, v případě, že bychom mohli přehodnotit některá naše rozhodnutí, jednalo by se pravděpodobně o využití PHP jako programovacího jazyka. Nově by volba padla pravděpodobně na Python a to díky podpoře vláken a širším možnostem

[1] http://nalus.usoud.cz/Search/GetText.aspx?sz=pl-24-10

při volání služeb operačního systému, což jsme velmi intenzivně využívali.

Druhým směrem, kterým by se mohl ubírat další vývoj naší aplikace, je rozšíření možnosti našeho nástroje o nové funkce. Může se jednat například o možnost automatického hlášení všech pohybů definovaných pohledem. Také bychom mohli přidat podporu pro výpočet statistiky nad síťovým tokem. Tím bychom však začali zasahovat do funkcí, které podporují již existující nástroje, čemuž jsme se v počátku naší práce rozhodli vyhnout. Je zde i otázka, zda v případě potřeby statistiky nad síťovým tokem nevyužít některý již existující nástroj.

Tím máme na mysli například NfSen, který můžeme v případě potřeby také dále upravit díky otevřenému zdrojovému kódu.

Literatura

[1] Vyhláška č. 485/2005 Sb. , o rozsahu provozních a lokalizačních údajů, době jejich uchovávání a formě a způsobu jejich předávání orgánům oprávněným k jejich využívání, ve znění účinném do 12. 4. 2011.

[2] BROWNLEE, N.; of Auckland, T. U.; MILLS, C.; aj.: Traffic Flow Measurement: Architecture, RFC 2722, 1999.

[3] CISCO: NetFlow Version 9 Flow-Record Format [poslední úpravy: květen 2011, citováno 9. 5. 2012].
URL <http://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html>

[4] CLAISE, B.; Cisco Systems, I.: Cisco Systems NetFlow Services Export Version 9, RFC 3954, 2004.

[5] CLAISE, B.; Cisco Systems, I.: Datagram Transport Layer Security, RFC 4347, 2006.

[6] CLAISE, B.; Cisco Systems, I.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information, RFC 5101, 2008.

[7] DIERKS, T.; RESCORLA, E.: The Transport Layer Security (TLS) Protocol Version 1.2, RFC 5246, 2008.

[8] ERICKSON, J.: *Hacking - umění exploitace*. Zoner Press, 2005, ISBN 80-86815-21-8.

[9] GRÉGR, M.; PODERMAŃSKI, T.; ŠOLTES, M.; aj.: Design of Data Retention System in IPv6 network. Technická zpráva, 2011.
URL <http://www.fit.vutbr.cz/research/view_pub.php?id=9840>

[10] HOWARD, M.; LEBLANC, D.: *Bezpečný kód: techniky a strategie tvorby bezpečných webových aplikací*. Computer Press, 2008, ISBN 978-80-251-2050-7.

[11] KREJČÍ, R.: *Network Traffic Collection with IPFIX Protocol [online]*. Diplomová práce, Masarykova univerzita, Fakulta informatiky, 2009.
URL <http://theses.cz/id/l9mzlh/>

[12] MITNICK, K.: *Umění klamu*. Helion, 2003, ISBN 83-7361-210-6.

[13] OWASP: Top 10 2004 - Lists of the most serious web application vulnerabilities [poslední úpravy: 20. 4. 2010, citováno 9. 5. 2012].
URL <https://www.owasp.org/index.php/Top_10_2004>

[14] OWASP: Top 10 2007 - Lists of the most serious web application vulnerabilities [poslední úpravy: 20. 4. 2010, citováno 9. 5. 2012].
URL <https://www.owasp.org/index.php/Top_10_2007>

[15] OWASP: Top 10 2010 - Lists of the most serious web application vulnerabilities [poslední úpravy: 27. 4. 2010, citováno 9. 5. 2012].
URL <https://www.owasp.org/index.php/Top_10_2010-Main>

[16] PECINOVSKÝ, R.: *Návrhové vzory*. Computer press, 2007, ISBN 978-80-251-1582-4.

[17] PHAAL, P.; PANCHEN, S.; McKEE, N.; aj.: InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks, RFC 3176, 2001.

[18] RAJAHALME, J.; Nokia; CONTA, A.; aj.: IPv6 Flow Label Specification, RFC 3697, 2004.

Příloha A

Obsah DVD

- Zdrojové soubory aplikace Nefelé
- Textová zpráva
- Výsledný report penetračního testování aplikace Nefelé
- Uživatelský manuál

Příloha B

Manuál