VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATIZOVANÉ TESTOVÁNÍ PRODUKTŮ FLOWMON

AUTOMATED TESTING OF FLOWMON PRODUKTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR HROMÁDKA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. PETR MATOUŠEK, Ph.D., M.A.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Hromádka Petr**
Obor: Informační technologie
Téma: **Automatizované testování produktů Flowmon**
Automated Testing of Flowmon Products
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se s nástroji FlowMon kolektor a FlowMon ADS.
2. Navrhněte postup automatizovaného testování těchto nástrojů podle doporučení konzultanta. Vstupem pro testy bude zachycený síťový provoz ve formátu PCAP. Systém bude ověřovat chování nástrojů pro zadané události či podle definovaných pravidel chování.
3. Implementujte testovací procesy pro FlowMon kolektor i ADS. Proces bude navržen jako otevřený a konfigurovatelný pomocí zadaného konfiguračního souboru.
4. Otestujte automatizovaného testování na několika příkladech, popište výsledky.
5. Diskutujte možnosti využití a budoucího rozšíření.

Literatura:

- Dokumentace k nástrojů FlowMon, viz https://www.flowmon.com/en/resources, září 2016.
- Manuálové stránky programu nfdump.
- B. Claise: Cisco Systems NetFlow Services Export Version 9, IETF RFC 3954, 2004
- B. Claise, B. Trammel, P. Aitkins: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information, IETF RFC 7011, 2013

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

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

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

Vedoucí: **Matoušek Petr, Ing., Ph.D., M.A.**, UIFS FIT VUT
Konzultant: Špringl Petr, Ing., FlowMon
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce se zabývá způsoby testování produktů: Flowmon kolektor a Flowmon ADS. Cílem práce je vytvořit frameworky pro automatizované testování pro nástroje firmy Flowmon. Úvod je věnován teorii testování. Další části se zabývají popisem testovaných nástrojů a návrhem samotných testovacích frameworků.

Abstract

This thesis is about testing of Flowmon products: Flowmon collector and Flowmon ADS. The goal is to implement frameworks for automatic testing of Flowmon tools. Beginning is about theory of testing. In next parts are dedicated to description of Flowmon tools and draft of testing frameworks.

Klíčová slova

Flowmon Networks, Flowmon kolektor, Flowmon ADS, automatizované testování, testovací framework, Python

Keywords

Flowmon Networks, Flowmon collector, Flowmon ADS, automatic testing, testing framework, Python

Citace

HROMÁDKA, Petr. *Automatizované testování produktů Flowmon.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Matoušek Petr.

Automatizované testování produktů Flowmon

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Matouška Ph.d. M.A. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Hromádka
14. května 2017

Poděkování

Tímto bych chtěl poděkovat firmě Flowmon Networks za zadání této práce, kolegům za věcné poznámky a zpětnou vazbu, doktoru Matouškovi za akademické vedení práce a rodině za neutuchající podporu.

Obsah

Kapitola 1

Úvod

Spolehlivá a dobře zabezpečená počítačová síť je základem k úspěšnému fungování téměř každé organizace. I krátkodobý výpadek sítě nebo úspěšný útok na data může způsobit obrovské škody, poškození jména společnosti nebo dokonce ztrátu zákazníků. Síťoví správci tak mají složitý úkol, při kterém musí zvládat neustále se vyvíjející se IT prostředí a zajistit ochranu proti stále pokročilejším a četnějším útokům, či jiným hrozbám. To vyžaduje využívání moderních přístupů a nástrojů, které umožňují efektivně monitorovat a spravovat počítačovou síť.

Vzhledem ke kritičnosti práce síťového správce je potřeba mít nástroj, který opravdu zobrazuje reálný provoz, proto zde možná i více než u jiného softwaru platí, že je potřeba při vývoji nástroj pořádně testovat. Testování může být z mnoha hledisek ať z uživatelského hlediska, jestli správce najde a vidí v rozumné době vše, co potřebuje, z funkcionálního hlediska, zda nástroj dělá opravdu to co má, nebo z výkonnostního hlediska, tedy, které části jsou pomalé a bylo by je potřeba zlepšit.

Flowmon je kompletní řešení pro detailní viditelnost do síťového provozu, optimalizaci výkonu sítě a ochranu před moderními kybernetickými hrozbami. Flowmon se skládá z Flowmon sond pro export Netflow/IPFIX statistik, Flowmon kolektorů k uložení, vizualizaci a analýze síťového provozu a Flowmon rozšiřujících modulů.

Cílem této bakalářské práce je vytvořit testovací frameworky v jazyce Python verze 2.6 pro nástroje Flowmon kolektor a Flowmon ADS. O testování, jaké jsou typy, možnosti, klasifikace se zabývám v druhé kapitole. Stručný popis testovaných nástrojů je ve třetí kapitole. Samotný popis, návrh i případové studie testovacích frameworků jsou ve čtvrté a páté kapitole.

Postup práce by měl vypadat následovně:

1. Nastudovat techniky testování
2. Nastudovat a vyzkoušet si Flowmon nástroje
3. Zhodnotit jaký bude nejvhodnější přístup
4. Vytvořit návrh a první prototyp
5. Sbírat zpětnou reakci a upravovat/rozšiřovat testovací frameworky

Kapitola 2

Způsoby testování

V této kapitole bude teoretický základ k testování, jaké jsou typy testování a jakým způsobem se provádí.

Proč se testuje? Testování má dva odlišné cíle: najít bugy nebo demonstrovat korektní chování [7]. Nejprve je potřeba zadefinovat co je to vlastně bug. Podle knihy Software testing od Rona Pattona [8] se bug v softwaru vyskytuje, pokud je splněno jedno z následujících pravidel:

1. Software nedělá něco co by podle specifikace[1] měl dělat.
2. Software dělá něco co by podle specifikace neměl dělat.
3. Software dělá něco o čem se specifikace nezmiňuje, ale pravděpodobně by to neměl dělat.
4. Software nedělá něco co ve specifikaci není zmíněno, ale měl by to dělat.
5. Software je nepřehledný, špatně se používá, je pomalý...

Existuje několik různých přístupů k testování, pro úplné otestování produktu je obvykle potřeba kombinace několika způsobů. Při zpracování následujících přístupů jsem vycházel zejména z již výše citované knihy Software testing od R. Pattona [8] a Software Testing Techniques od B. Beizera [1].

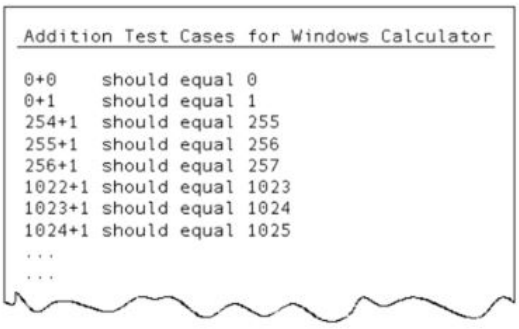
Statické Black-Box testování

Testování produktové specifikace, ne produktu jako takového. Účelem tohoto testování je objevit bugy ještě dříve než vůbec vzniknou, odhalit špatně pochopitelné nebo nedostatečně popsané funkcionality, případně nadbytečné nebo chybějící funkce. Při tomto přístupu tester vidí pouze to co uvidí i zákazník a měl by přemýšlet jako on.

Dynamické Black-Box testování

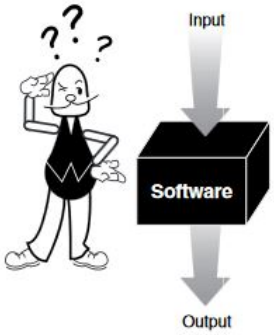
Také nazývané *testování s klapkami na očích*, pracuje už s produktem. Základem je sada vstupů a k nim sada očekávaných výstupů.

[1] Produktová specifikace je detailní popis produktu, který popisuje jednotlivé části produktu a dává informace o používání a funkcionalitách `https://techwhirl.com/writing-software-requirements-specifications`



Obrázek 2.1: Příklad vstupů a k nim očekávaných výstupů pro kalkulačku. [8]

Tester nemá přístup ke zdrojovému kódu produktu, aby používané sady mohl přizpůsobit, ale chová se k produktu stejně jako zákazník. Při dynamickém black-box testování mohou být sady navržené tak, aby prošly a demonstrovaly korektní chování nebo neprošly a zkontrolovaly jak se produkt chová v případě chyby, jak informuje o chybě, či jestli chyba nezpůsobí neočekávaný pád celého produktu.



Obrázek 2.2: Znázornění black-box testování. [8]

Statické White-Box testování

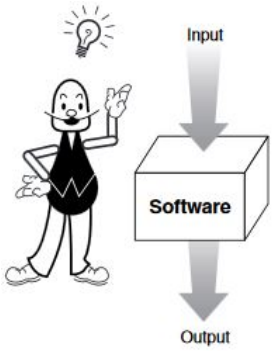
Inspekce návrhu a kódu. Jiný název pro tento přístup je *strukturální analýza*. Při tomto přístupu má tester přístup do kódu a snaží se nalézt chyby v kódu, nevhodnou architekturu bez toho, aby tester produkt spouštěl. Tento přístup obvykle využívají sami vývojáři, protože většinou mají větší zkušenosti s programováním než testeři. Velkou pomocí v tomto případě bývají překladače[2], které dokáží upozornit na nevhodné přístupy do paměti, špatné přetypování, či jiné techniky, které se při samotném běhu nemusí projevit vždy, nebo specializovaný software, který kontroluje určité standardy při psaní kódu[3].

[2]Příkladem mohou být argumenty -Wall nebo -Wextra u překladače gcc `https://gcc.gnu.org/`.
[3]Například knihovna pep8 pro python `https://pypi.python.org/pypi/pep8`.

Dynamické White-Box testování

Jiný název pro tento přístup je *strukturální testování*, protože je vidět struktura produktu a při testování je produkt spuštěn. Tester využívá přístupu ke zdrojovým kódům a může určit co testovat a co ne. Tento přístup není limitován pouze na pozorování. Dynamické white-box testování zahrnuje 4 oblasti:

- Přímé testování nízkoúrovňových funkcí, procedur, nebo knihoven.
- Testování produktu na nejvyšší úrovni jako celku, podobně jako dynamické black-box testování s tím, že lze testy upravit na základě vnitřní struktury produktu.
- Přístup ke čtení proměnných a stavových informací umožňuje ověřit, že testy opravdu dělají to co mají testovat, zároveň je tester schopný vypisovat více informací, které se mohou k testování hodit, což by v případě dynamického black-box testování nebylo možné.
- Měření jaké množství kódu se provádí při spuštění testů za účelem odstranění nějakých redundantních testů a přidáním chybějících.



Obrázek 2.3: White-box testování. [8]

Integrační testování

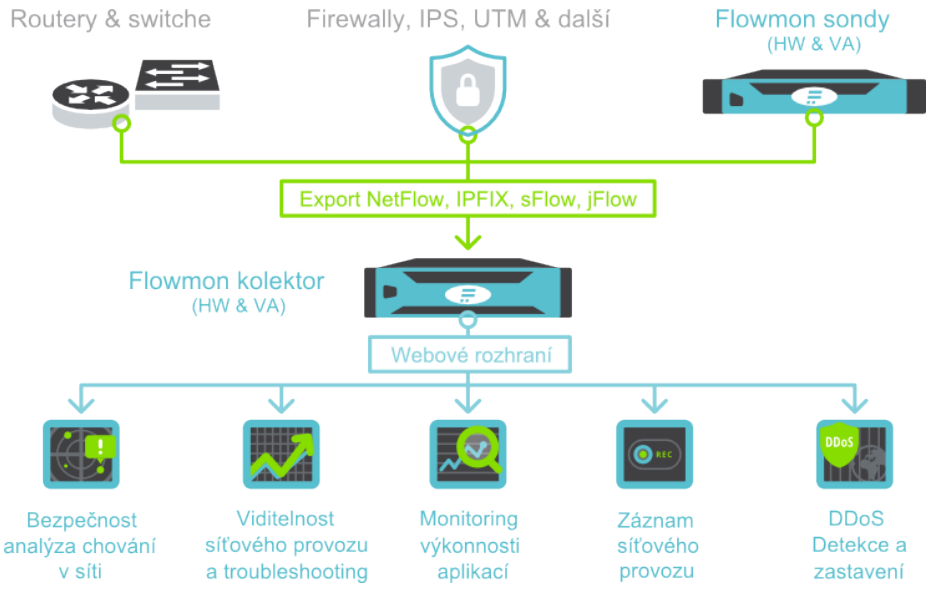
Slouží k testování spojených prvků produktu. Při psaní velkých projektů je obvykle scénář takový, že se samostatně nejprve otestují jednotlivé komponenty a následně integrační testy testují skupiny modelů případně produkt jako celek. Pokud se postupuje tímto způsobem a objeví se nějaké bugy jedná se s největší pravděpodobností o nějaké nekompatibility spojených modulů.

Kapitola 3

Flowmon nástroje

Tato kapitola popisuje Flowmon nástroje, nejprve se zaměřím na Flowmon kolektor, jeho funkcionality a důležité součásti. Dále pak podobným způsobem bude rozbrán modul Flowmon ADS.

Flowmon tvoří ucelené řešení pro monitorování sítí na bázi datových toků[1] ve formě Netflow nebo IPFIX. Zahrnuje Flowmon sondy, které generují statistiky o síťovém provozu, Flowmon kolektory ukládající a analyzující tyto statistiky s možností zobrazení, Flowmon ADS[2] pro automatickou analýzu provozu a identifikaci bezpečnostních a provozních problémů, Flowmon DDoS Defender pro ochranu proti volumetrickým DDoS útokům, Flowmon APM pro minotorování výkonnosti HTTP/HTTPS a databázových aplikací a Flowmon Traffic Recorder pro záznam síťového provozu [5].



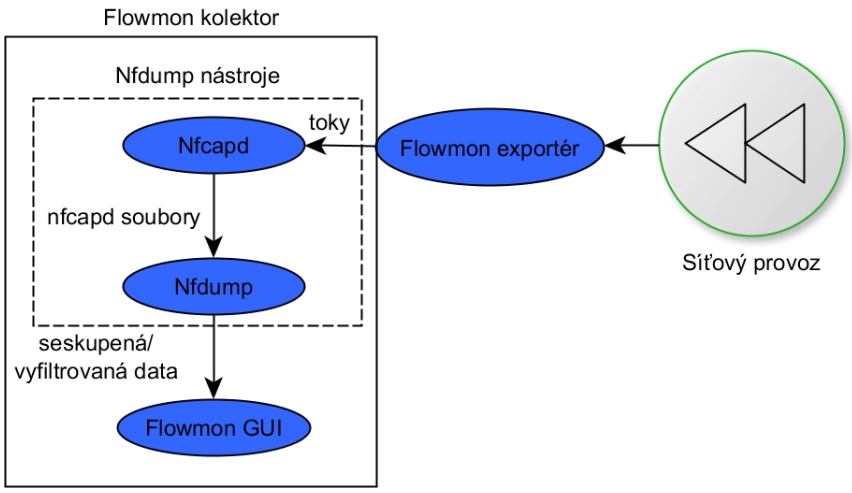
Obrázek 3.1: Schéma Flowmon řešení [6].

[1] Tok je v terminologii NetFlow definován jako sekvence paketů se shodnou pětící údajů: cílová/zdrojová IP adresa, cílový/zdrojový port a číslo protokolu. Pro každý tok je zaznamenávána doba jeho vzniku, délka jeho trvání, počet přenesených paketů a bajtů a další údaje. `http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html`

[2] Anomaly Detection System

3.1 Flowmon kolektor

Flowmon kolektor je fyzické či virtuální zařízení určené pro dlouhodobé ukládání, zobrazování a analýzu síťového provozu ve formátech Netflow/IPFIX, které generují switche, routery, síťové sondy[3] či jiné zdroje [4]. Kolektor je založený na open sourcovém nástroji Nfdump[4], který je rozšiřován o další funkcionality a optimalizován.



Obrázek 3.2: Schéma Flowmon kolektoru.

Flowmon kolektor je složený z několika částí:

1. Nfdump nástroje - slouží k ukládání síťového provozu a z uložených dat umožňuje filtrovat určitá data na základě filtrů, či agregace.

2. Flowmon GUI - je určen k zobrazování statistik, vykreslování grafů a jednoduchému vytváření filtrů a agregace pro Nfdump nástroje.

Flowmon exportér není součástí kolektoru, ale je součástí sondy. V některých případech, ale může běžet přímo na kolektoru, aby bylo možné zachytávat data i z jiných síťových zařízení než jsou sondy či jiné zdroje exportující data přímo pro kolektor.

3.1.1 Nfdump nástroje

Nfdump nástroje je sada nástrojů pro sbírání a práci s Netflow daty. Flowmon řešení podporuje Netflow v1/v5/v7/v9/IPFIX a sFlow, IPv4 i IPv6.

Nfcapd je zachytávací daemon, který čte Netflow/IPFIX data ze síťového rozhraní (nebo přímo ze zachyceného pcap souboru) kam data přeposílají exportovací zařízení. Flowmon řešení podporuje velké množství v9/IPFIX rozšiřujích položek položek třetích stran[5]. Pro optimalizaci paměťového prostoru a výkonu jsou v9 položky seskupovány do rozšíření, která

[3]Sonda je pasivní zařízení v síti, které monitoruje provoz a vytváří statistiky v podobě IP toků a zasílá je k dalšímu zpracování na kolektor.

[4]Dostupné z `http://nfdump.sourceforge.net/`.

[5]Společnosti si mohou zaregistrovat *Private enterprise number* u operátora IANA, které pak slouží jako identifikátor pro Netflow/IPFIX položky.

mohou nebo nemusí být uloženy ve výstupním souboru. Nfcapd musí používat stejné v9 položky jako exportující zařízení, aby je mohl uložit, pokud některým exportovaným položkám *nerozumí* jednoduše je zahazuje.

Nfdump slouží k zobrazování a analýze nfcapd souborů. Čte Netflow/IPFIX data ze souborů vytvořených nfcapd a zpracuje je podle zadaných argumentů. Flowmon řešení umí filtrovat, agregovat, zobrazovat různé statistiky pro různé Netflow/IPFIX položky.

3.1.2 Flowmon exportér

Flowmon exportér sleduje provoz na síti, počítá detailní statistiky o síťové komunikaci, přináší tak přesné informace o komunikujících stranách, protokoly, množství dat a další důležité informace z vrstev L2-L4 a převádí ho na záznamy toků. Exportér může být spuštěn s různými pluginy, které určují jeho chování a mohou poskytovat i viditelnost do aplikační L7 vrstvy.. Existují vstupní, procesní, filtrační a výstupní pluginy.

Vstupní pluginy předávájí exportéru vstupní data ve formě packetů. Ty jsou následně zpracovány v procesních pluginech, kde je z nich vytvořen záznam toku a uložen do cache paměti exportéru. Pakety jsou v cache paměti adresovány pomocí hashe vytvořeného z CRC[6] vypočítaného z IP adresy, portu a L4 protokolu. Při vytváření záznamu toku se některé informace vyplňují automaticky např. časová známka, počet packetů atd. Počet elementů v záznamu toku může být rozšířen použitím různých procesních pluginů.

Cache paměť exportéru je periodicky kontrolována na aktivní[7] a neaktivní[8] timeouty. Pokud záznam toku přesáhne některý z timeoutů, nebo je detekován konec toku (např. FIN příznak u TCP) je předán filtračním pluginům, které určují jaká data exportovat a následně je výstupními pluginy vyexportován tok ve formě Netflow/IPFIX nebo zobrazen v jiném formátu např. json, csv.

3.2 Flowmon ADS

Flowmon ADS (Anomaly Detection System) je rozšiřující plugin Flowmon kolektoru nebo Flowmon sondy pro odhalování provozních problémů a ochranu podnikových sítí před moderními kybernetickými hrozbami, které využívá technologii detekce anomálií a analýzy chování v síti (NBAD). Díky tomu je možné identifikovat hrozby, které překonaly zabezpečení na perimetru, byly zavlečeny do datové sítě jiným způsobem nebo pro ně dosud neexistuje signatura. Automatická detekce bezpečnostních incidentů, anomálií provozu datové sítě a konfiguračních problémů výrazně zjednodušuje správu datové sítě, zvyšuje její bezpečnost a umožňuje proaktivně identifikovat příčiny problémů [3].

Základem je databáze, nad kterou je prováděna behaviorální analýza pomocí algoritmů s prvky umělé inteligence, která umožňuje sledovat provoz na síti jako celek a tak odhalovat a reagovat na doposud neznámé nebo specifické hrozby, včetně těch, pro které neexistuje signatura.

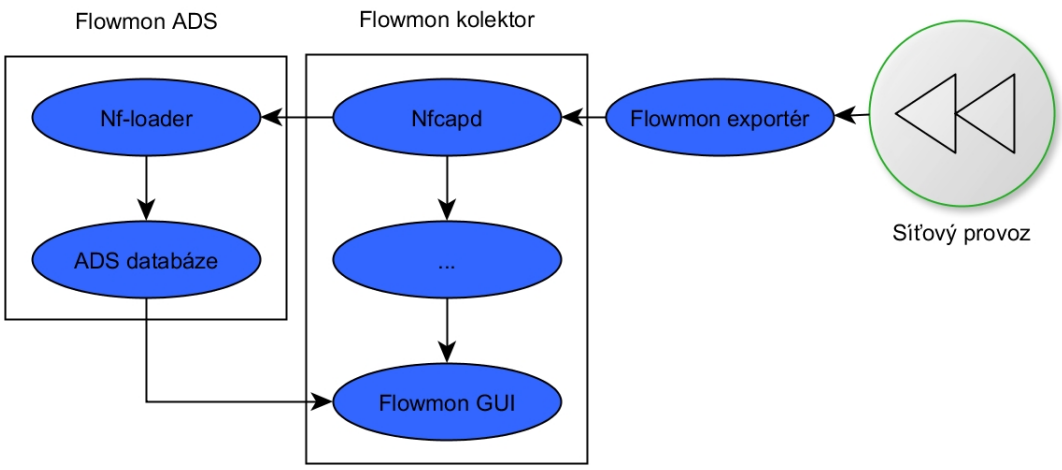
K nahrávání dat do databáze slouží nástroj ads-nf-loader, který na vstup bere nfcapd soubory a postupně je nahrává do databáze v 5 minutových intervalech.

Výstup z ADS potom tvoří grafy a seznam nalezených událostí. Události jsou řazeny podle priority, která jim je přiřazena na základě perspektivy. V rámci perspektivy lze udá-

[6] Obvykle se využívá jako kontrolní součet pro detekci chyb během přenosu či ukládání dat, ale zde vystupuje také jako hashovací funkce.

[7] Příliš dlouhý tok je rozdělen.

[8] Neaktivní tok.



Obrázek 3.3: Schéma Flowmon ADS.

lostem přiřadit kritickou, vysokou, střední, nízkou nebo informativní váhu. Kromě grafů a seznamu lze zapnout hlášení nalezených událostí, které může být prováděno pomocí emailu, syslogu, SNMP pastí a uživatelského skripu. Zpravy pro syslog a SNMP jsou odesílány okamžitě, pro email a uživatelský skript jsou zprávy generovány podle priority události nastavené perspektivou:

1. kritická: okamžitě
2. vysoká: po 1 hodině
3. střední: po 6 hodinách
4. nízká: jednou denně
5. informační: jednou týdně

Existují tři typy událostí:

- Jednoduché - Událost, která se vygeneruje v případě nějakého konkrétního provozu.
- Naučené - Událost, která sleduje provoz na síti a vygeneruje se až v případě dlouhodobého podezřelého provozu, nebo v případě, že na jednoduchou událost navazuje jiná.
- Složené - Je pouze informativní událost a umožňuje pojmenovat posloupnost podezřelých událostí. Tato událost se nehlásí do žádného modulu, pouze se může zobrazit v grafech.

Kapitola 4

Testování Flowmon kolektoru

Jedním z cílů této práce je vytvořit testovací framework pro Flowmon kolektor, umožňující automatizované testování. Podrobnější požadavky na samotné testování budou rozebrány v následující sekci. Dále se budu zabývat, tím jaké přístupy testování popsané v kapitole 2 jsem použil a z jakého důvodu. V dalších sekcích této kapitoly se zabývám návrhem frameworku, co jsou jeho vstupy, výstupy, následuje gramatika konfiguračního souboru, který ovlivňuje chod celého frameworku. Konec této kapitoly je věnován případové studii a úplně na závěr obsahuje integraci do systému Jenkins, tedy to, jakým způsobem se framework reálně využívá.

4.1 Požadavky na framework

Požadavkem bylo vytvořit framework pro automatizované testování backendu Flowmon kolektoru. Původní požadavek byl pak rozšířen i o možnost testovat přímo Flowmon exportér, protože spolu úzce souvisí, zejména tím, že pokud se vytvoří nový plugin do Flowmon exportéru, je potřeba rozšířit rovněž Nfdump nástroje. Další důvod pro zahrnutí Flowmon exportéru přímo do frameworku byl fakt, že je to nejjednodušší způsob, jak simulovat síťový provoz podobným způsobem, jaký probíhá u zákazníků. Výsledné požadavky tedy jsou:

1. Možnost automatizovaného testování backendu Flowmon kolektoru a Flowmon exportéru konfigurovatelné vstupním souborem.
2. Spustitelnost na Flowmon kolektoru (založený na CentOS 5).
3. Testování v prostředí VMware ESXi[1].
4. V jazyce Python verze 2.6.

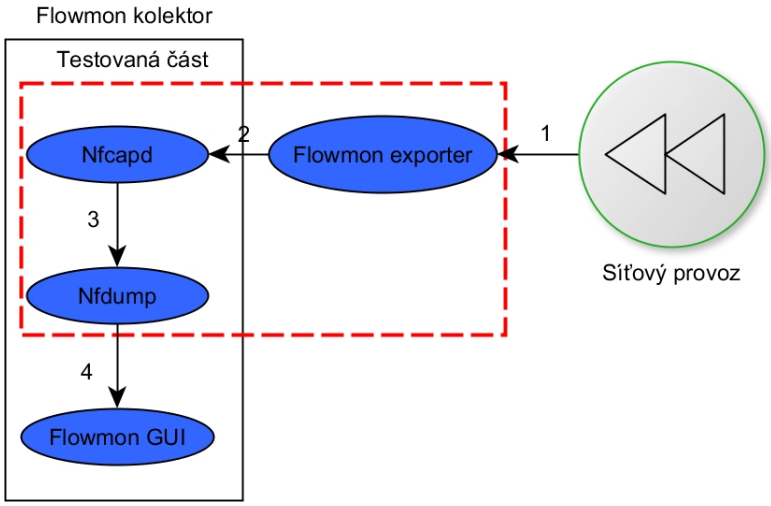
4.2 Návrh frameworku

Při návrhu frameworku byl kladen důraz na komplexnost, rozšiřitelnost a konfigurovatelnost řešení. Testování bylo navrženo jako white-box testování s prvky integračního testování. To znamená, že na vstupu je sada vstupů a výsledek je porovnáván se sadou očekávaných výstupů s tím, že očekávané výstupy jsou upraveny díky možnosti náhledu do vnitřní struktury, např. při testování pluginu není potřeba kontrolovat formátování času, protože to není

[1] Prostředí pro vytváření a obsluhu virtuálních počítačů `https://en.wikipedia.org/wiki/VMware_ESXi`

v režii pluginu. Zároveň v případě chyby lze jednoduše dohledat, ve kterém z testovaných prvků chyba nastala.

Následující obrázek zobrazuje, jaká oblast Flowmon kolektoru je testována. Zároveň znázorňuje vstupy a výstupy jednotlivých součástí Flowmon kolektoru, které jsou stručně popsány v podkapitole 3.1. Samotné struktuře frameworku bude věnována další podkapitola.



Obrázek 4.1: Návaznost testovacího frameworku na Flowmon kolektor.

Testování pomocí frameworku zahrnuje nfcapd a nfdump jakožto backend Flowmon kolektoru a Flowmon exportér, který s kolektorem úzce souvisí. V obrázku je vyznačeno několik částí, což jsou veškeré vstupy/výstupy frameworku.

1. Na vstupu je sada souborů typu pcap se síťovým provozem, která se přehraje do Flowmon exportéru. Soubory typu pcap jsou vytvořeny tak, aby obsahovaly různé síťové protokoly a druhy provozu podle účelu testu. Framework umožňuje přeskočit tento krok a tím pádem testovat pouze nfdump nástroje, nicméně je k tomu zapotřebí pcap soubor s provozem, který je již zpracován Flowmon exportérem nebo nějakým jiným nástrojem do formy toků. Pro přehrávání je využíván nástroj Tcpreplay[2]

2. Flowmon exportér získává ze síťového provozu toky. Při testování pouze nfdump nástrojů je pcap soubor se zachycenými toky přehráván po loopbacku.

3. Nfcapd vytvoří soubory, které dále zpracovává nfdump. Framework je ukládá do samostatné složky pro případné manuální testování.

4. Sada textových souborů je vytvořena nástrojem nfdump, data z nfcapd jsou vyfiltrována, agregována, či seřazena. Tato sada je porovnávána s očekávanými výsledky. Porovnávání může probíhat mnoha způsoby detaily budou rozbrány v sekci Gramatika konfiguračního souboru 4.2.2.

[2]Dostupný na `https://sourceforge.net/projects/tcpreplay/`

4.2.1 Struktura frameworku

Framework dodržuje určitou strukturu a je vytvořen v jazyce Python, nicméně používá pomocné skripty v jazyce Bash. Kořenový adresář musí být ve složce `/home/flowmon`. Struktura frameworku pak vypadá následovně:

Výpis 4.1: Adresářová struktura frameworku

```
Adresarova struktura pro testovaci framework:
   test/                        (kořenový adresář)
    -test                       (hlavní pythonovský skript)
    -clean.sh                   (skript pro čištění výstupů testů)
    -stop_exporters.sh          (skript pro zastavení defaultních exportérů)
    -prepare.sh                 (skript pro stažení potřebných knihoven)
    -run_tests                  (pomocný skript pro automaticke spuštění)
    -modules/                   (složka s moduly pro skript "test")
      -nfdump.py
      -nfcapd.py
      -replay.py
      -shared_functions.py
      -check_config.py
    -tests/                     (složka zastřešující konfigurační soubory)
      -example_test_local/
        -config.ini
        -pcaps/
        -test_outputs/
          -nfcapd_out/
          -nfdump_out/
          -exporter_out.json
          -comparing.log
      -example_test_with_exporter/
        -config.ini
        -exporter.json
        -exporter_template.txt
        -dalsi soubory pro pluginy
        -pcaps/
```

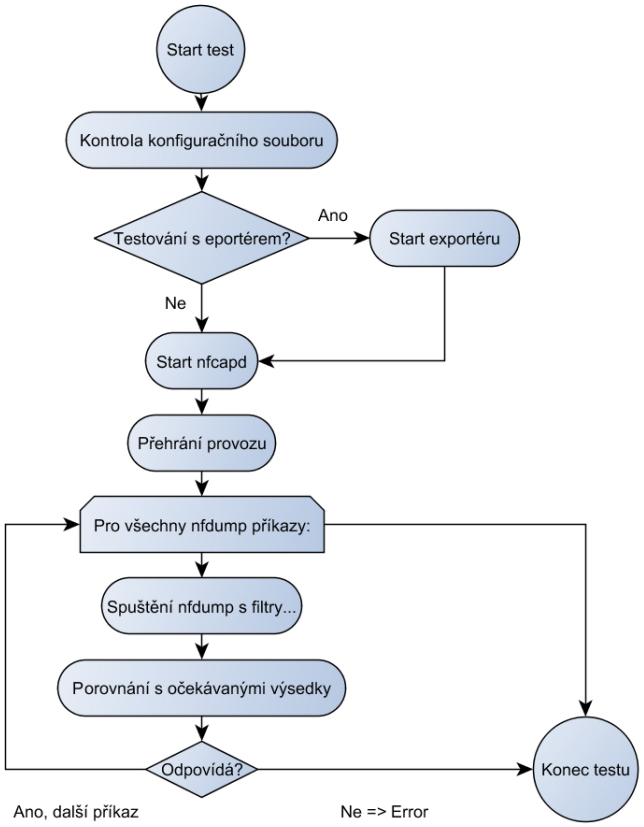
- Pythonovský skript *test* je hlavní skript sloužící k testování. Tento skript využívá funkce ze složky *modules*. Zastřešuje veškeré funkcionality testu. Při běhu vytváří výstupní soubory ve složce, kde se nachází vstupní konfigurační soubor. Detail je v následujícím obrázku 4.2

- Bashovský skript *clean.sh* je pomocný skript, který je určen k čištění všech výstupních souborů vytvořených při běhu testu a pro opakování testu.

- *stop_exporters.sh* ukončí výchozí exportéry a přidá výjimku do watchdogu [3], takže je možné v rámci testování spouštět vlastní exportéry (jinak to na kolektoru není možné, resp. watchdog je periodicky ukončuje).

- *prepare.sh* stáhne nástroj tcpreplay, nainstaluje potřebné pythonovské knihovny a vytvoří dummy interface, pomocí kterého je možné přehrávat provoz po loopbacku pro lokální testování pouze nfdump nástrojů. Dummy interface je potřeba vytvořit

[3] Watchdog je softwarové nebo hardwarové zařízení, které periodicky kontroluje systém před chybami. Běžící aplikace posílají signály. Pokud watchdog nepříjme správný signál tak provede nadefinovanou akci, například restart aplikace, či celého systému.



Obrázek 4.2: Diagram datových toků.

kvůli tomu, že tcpreplay potřebuje přehrávat provoz na cílovou MAC adresu, která běžně nelze přiřadit na loopback. Dummy interface se pak chová stejně jako loopback, ale obsahuje navíc MAC adresu.

- *run_tests* je pomocný python skript, který jako parametr bere skupiny testů ze složky *tests* a ty následně spustí (např. ./run_tests DHCP). Spouští hlavní skript *test* automatizovaně pro více konfiguračních souborů za sebou. Zároveň vytváří logovací soubor.

- Složka *tests*, slouží k uchovávání všech testů. Testy jsou logicky členěny podle jejich zaměření. Každá podsložka pak obsahuje veškeré potřebné soubory k testování, tedy konfigurační soubor a souboy typu pcap. Pro exportér je to pak navíc konfigurační soubor pro exportér, dále textový soubor s šablonami pro výstupní plugin, který exportuje data ve formě IPFIX toků. Dále to mohou být další textové soubory s informacemi pro určité pluginy, příkladem může být překlád VLAN na rozhraní, kde jsou v souboru obsaženy pravidla pro překlad.

  Rozčleněním na jednotlivé podsložky lze testy automatizovaně spouštět po určitých skupinách nebo všechny. Navíc tato zavedená adresářová struktura umožňuje jednoduchý přístup k výstupním souborům, které jsou vždy tvořeny v umístění konfiguračního souboru, tedy v jednotlivých podsložkách.

4.2.2 Gramatika konfiguračního souboru

Obsah konfiguračního souboru určuje spuštění samotného testovacího frameworku a významně ovlivňuje samotný chod. Podle určitých pravidel se může spustit Flowmon exportér, lze ovlivnit kontrolu logovacích zpráv, zda nenastávají chyby, které přímo neovlivňují výsledek, apod. Gramatika konfiguračního souboru je zapsaná pomocí ABNF [2]:

Výpis 4.2: Gramatika konfiguračního souboru

```
configFile = common_settings
             nfcapd_settings
             1*pcap
             1*nfdump_command
             *comment

common_settings    = "[common_settings]" CRLF name exporter check_syslog
                     [change_mtu][debug]
name               = "name=" NAME CRLF
exporter           = exporter_set/
                     (exporter_set exporter_arguments [exporter_options]
                     [exporter_output])
exporter_set       = "exporter=" "yes"/"no" CRLF
exporter_arguments = "exp_args=" PATH_TO_JSON CRLF
exporter_options   = "remote=" IP_ADDRESS CRLF
                     "remote_user=" USER CRLF
                     "remote_PW=" PASSWORD CRLF
                     "remote_root_pw=" ROOT_PASSWORD CRLF
exporter_output    = "exp_out=yes" CRLF
check_syslog       = "check_syslog=" "yes"/"no" CRLF
change_mtu         = "change_mtu=" NUMBER CRLF
debug              = "debug=yes" CRLF

nfcapd_settings = "[nfcapd_settings]" CRLF extensions [port] [path]
extensions      = "ext=" NFCAPD_EXTENSIONS CRLF
port            = "port=" NUMBER CRLF
path            = "path=" PATH CRLF

pcap   = "[pcap" NUMBER "]" CRLF path speed replay
speed  = "nspeed=" "0"/"1" CRLF
replay = "replay=" "l"/"r" CRLF
nfdump_command    = "[nfdump_" NUMBER/STRING "]" CRLF arguments [path]
                    comparing_command
arguments         = "args=" NFDUMP_ARGUMENTS CRLF

comparing_command = lineX/NlinesX/findline/findNlines/findnline/
                    findnNline/foundline/linesbetween/file_compare
lineX             = "line" NUMBER "=" ["-"] REGULAR_EXPRESION CRLF
NlinesX           = NUMBER "lines" NUMBER "=" [-] REGULAR_EXPRESION CRLF
findline          = "findline=" [-] REGULAR_EXPRESION CRLF
findNlines        = "find" NUMBER "lines=" [-] REGULAR_EXPRESION CRLF
findnline         = "findnline=" [[-]REGULAR_EXPRESION] CRLF
findnNlines       = "findn" NUMBER "lines=" [-] REGULAR_EXPRESION CRLF
foundlineX        = "foundline" NUMBER "=" [-] REGULAR_EXPRESION CRLF
linesbetween      = l_start l_end l_between
l_start           = "linesbetween_start=" REGULAR_EXPRESION CRLF
l_end             = "linesbetween_end=" REGULAR_EXPRESION CRLF
l_between         = "linesbetween_between=" [-] REGULAR_EXPRESION CRLF
```

```
file_compare      = file_cmp [file_shift] [file_ignore_lines] [
    file_regex_X]
file_cmp          = "file_cmp=" FILE CRLF
file_shift        = "file_shift=" [-] NUMBER CRLF
file_ignore_lines = "file_ignore=" NUMBER *(, NUMBER) CRLF
file_regex_X      = "file_regex_" NUMBER "=" REGULAR_EXPRESION CRLF

comment  = CRLF "#" STRING CRLF
```

Konfigurační soubor je složen z několika sekcí s tím, že kromě komentářů je každá sekce povinná.

Sekce *common_settings* obsahuje možnosti spuštění celého frameworku. Je zde definice názvu testu, možnosti testování s exportérem, zapnutí výstupu exportéru, kontrola syslogu o chyby, které přímo nemusí ovlivnit test (např. zahazování packetů). Pokud se zahodí packety, které nebudou mít přímo nějaký vliv na kontrolované toky, tak se chyba nemusí projevit. Dále je zde vzdálené přehrávání provozu, změna velikosti MTU či zapnutí debugovacích výpisů.

Sekce *nfcapd_settings* určuje spuštění nfcapd nástroje. Jediný povinný parametr v této sekci je seznam rozšíření, se kterými se má nfcapd spustit, volitelně pak číslo portu a cesta k binárnímu souboru, které umožňují testovat jinou verzi nfcapd než právě nainstalovanou.

Sekce *pcap* se může opakovat a obsahuje cestu k přehrávanému pcap souboru, dále určuje rychlost přehrávání (obvykle 1, tedy rychlost se kterou byl pcap soubor zachycen) a lokální, či vzdálené přehrávání.

Poslední sekce *nfdump_command* zajišťuje parametry pro spuštění nfdump nástroje, případně cestu k jinému binárnímu souboru podobně jako v sekci *nfcapd_settings*. Hodnoty pro položku *NFCAPD_EXTENSIONS* je seznam rozšíření, se kterými lze nfdump spustit[4]. Navíc obsahuje pravidla pro porovnávání výsledku. K dispozici je mnoho způsobů jak výsledek porovnat, výsledek se vždy porovnává po řádcích:

1. *lineX* - aplikuje regex na daný řádek.
2. *NlinesX* - porovná sekvenci N řádků začínající na X řádku s regulárním výrazem.
3. *findline* - najde první řádek odpovídající regulárnímu výrazu, číslo řádku si zapamatuje. Pokud nic nenajde nastává chyba.
4. *findNlines* - najde N řádků odpovídajících regulárnímu výrazu, zapamatuje si číslo posledního řádku. Pokud najde méně než N řádků nastává chyba.
5. *findnline* - najde následující řádek odpovídající regulárnímu výrazu z předchozího findline/findNlines/findnline příkazu.
6. *findnNlines* - podobně jako findNlines s tím, že hledá další výskyty.
7. *foundlineX* - porovná X řádek po posledním řádku nalezeném pomocí findline/findNlines/findnline/findnNlines s regulárním výrazem.
8. *linesbetween* - je sekvence tří příkazů, při které se pomocí regulárních výrazů najde začátek, konec a poté se všechny řádky mezi tím porovnají regulárním výrazem.

[4]Kompletní seznam lze získat na Flowmon kolektoru příkazem: man nfcapd

9. *file_compare* - příkaz porovná výstup s textovým souborem zadaný tímto příkazem. Lze rozšířit o posouvání výstupu/textového souboru, ignorování některých řádků, případně porovnání určitých řádků s regulárním výrazem (např. řádků, které obsahují statistiky jako počet bitů za sekundu, které se téměř vždy budou lišit).

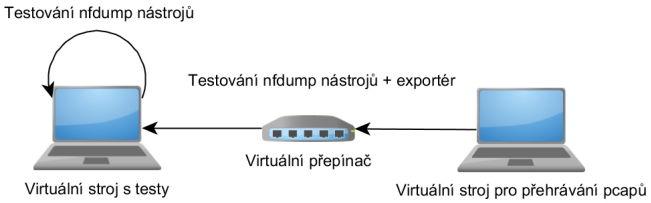
4.3 Spuštění frameworku

Framework se spouští pomocí skriptu *test*. Jeho jediným parametrem je cesta ke konfiguračnímu souboru, který obsahuje všechny další informace o spuštění.

4.3.1 Příprava prostředí k testování

Pro samotné testování je potřeba připravit prostředí, proto před prvním spuštěním je potřeba spustit skript *prepare.sh*, který nainstaluje nástroj Tcpreplay, potřebné pythonovské knihovny a vytvoří dummy rozhraní pro přehrávání přes loopback při testování Nfdump nástrojů.

Pro testování s Flowmon exportérem je navíc potřeba, aby testovaný virtuální stroj (Flowmon kolektor + Flowmon exportér) měl některé rozhraní nastavené na síť, která je spojená pomocí virtuálního přepínače v promiskuitním režimu[5] s virtuálním počítačem, na kterém se bude provoz přehrávat.



Obrázek 4.3: Zapojení při testování.

Při testování s exportérem je také potřeba vypnout výchozí exportéry na virtuálním stroji a přidat výjimku do watchdogu, který jinak Flowmon exportéry na všech aktivních rozhraních kromě výchozích ukončuje. K tomuto účelu slouží bashovský skript *stop_exporters.sh*.

4.3.2 Možnosti spuštění

Spouštění musí probíhat s rootovskými právy z kořenové složky testovacího frameworku, který musí být umístěn v adresáři `/home/flowmon`. Konfigurační soubory by měly být umístěny ve složce *tests*. Zde už to není nutná podmínka, ale umožňuje automatizaci pro systém Jenkins, navíc dodává logické členění. Příklad spuštění může vypadat následovně:

```
[root@localhost flowmon]# cd /home/flowmon/test/
[root@localhost test]# ./test tests/Variable_lenghts/variable_lenghts.ini
```

Obrázek 4.4: Příklad spuštění testu.

[5]Režim při, kterém se přepínač chová jako rozbočovač, přeposílá komunikaci na veškerá svá aktivní rozhraní

Kromě této možnosti byl vytvořen ještě pomocný pythonovský skript *run_tests*. Tento skript je parametrizovatelný a umožňuje automatizovaně spustit skript *test* pro všechny konfigurační soubory ve složce *tests* nebo jen pro vybrané. Tento skript kromě jiného vytváří logovací soubor, který umožňuje strojové zpracování pro systém Jenkins.

```
[root@localhost flowmon]# cd /home/flowmon/test/
[root@localhost test]# ./run_tests ALL
```

Obrázek 4.5: Příklad spuštění pomocného skriptu pro všechny testy.

```
[root@localhost flowmon]# cd /home/flowmon/test/
[root@localhost test]# ls tests/
DHCP  Extended_TCP  HTTP_methods  MAC_to_AS  SMB              VLAN_to_INF
DNS   Gigamon       HTTP_OS_APP   NBAR2      Variable_lenghts  VOIP
[root@localhost test]# ./run_tests "DNS,DHCP"
```

Obrázek 4.6: Příklad spuštění pomocného skriptu pro vybrané testy.

4.3.3 Odhad běhu testu

Odhadnout přesnou dobu běhu testu automatizovaně je poměrně složité, protože je závislá na mnoha proměnných. Například aktivní a neaktivní timeouty při spuštění exportéru se obvykle liší v závislosti na množství provozu. Také přehrávání provozu se liší pcap soubor od souboru.

Nicméně přibližná doba lze odhadnout podle následujícího vzorce:

```
t = čas_na_spuštění + čas_posledního_packetu +
[čas_exportování] + čas_na_zpracování
```

- *čas_na_spuštění*: zahrnuje režii spojenou se spouštěním nfcapd a exportéru, která se může lišit v závislosti na počtu zapínaných rozšíření a pluginů. Experimentálně bylo zjištěno, že tento čas by neměl být větší než 6 vteřin, proto je tato hodnota nastavena s rezervou na 10 vteřin. Navíc se tato doba může zvětšit v případě obsazenosti portu, potom se případě hledá další volný port.

- *čas_posledního_packetu*: souvisí s přehráváním provozu. Nejčastěji se používá možnost přehrávání tak jak byl provoz zachycen, tedy ve stejném čase. Existuje možnost maximální rychlosti, ale ta často vede k zahazování packetů a následným sekvenčním chybám. Proto tuto možnost lze využít pouze ve specifických případech. Tato hodnota nemá žádnou konstantní velikost a nezávisí na velikosti pcap souboru, ale na čase posledního packetu v pcap souboru. V aktuálně vytvořené testovací sadě se nejčastěji pohybuje mezi 5-20 vteřinami.

- *čas_exportování*: v případě testování s exportérem je potřeba navíc přičíst režii zpracování exportéru. Ta zahrnuje zejména čekání na neaktivní timeouty a zpracování samotných toků. Tato hodnota je nastavena na 60 vteřin.

- *čas_na_zpracování*: zpracování probíhá nfcapd nástrojem, který výsledky ukládá v určitých časových intervalech, nejkratší časový interval je 60 vteřin, ke kterému se musí přičíst určitá doba na zpracování. Tato hodnota je nastavena na 70 vteřin.

Přibližná doba testu s ohledem na vzorec je přibližně 150 vteřin při testování s exportérem a 90 vteřin při testování pouze nfdump nástrojů, ale v závislosti zejména na *čas_posledního_packetu* se může lišit.

4.4 Případová studie

Následující příklad zobrazuje testování s položkami variabilní délky[6] pro Netflow v9 a IPFIX, tedy zda se správně zpracovávají šablony a následně datové toky. V tomto případě jde pouze o testování nfdump nástrojů, na vstupu je k dispozici pcap soubor se zachycenými toky z exportéru. V následujících obrázcích je vidět detail tohoto souboru: 4.7 a detail provozu v souboru: 4.8.

y a display filter ... <Ctrl-/>

	Time	Source	Destination	Protocc	Length	Info
1	0.000000	10.69.101.89	10.69.101.90	CFLOW	578	IPFIX flow (536 bytes) Obs-Domain-ID= 256 [Data:262]
2	10.907423	10.69.101.89	10.69.101.90	CFLOW	106	total: 1 (v9) record Obs-Domain-ID= 200 [Data:256]
3	16.988393	10.69.101.89	10.69.101.90	CFLOW	106	total: 1 (v9) record Obs-Domain-ID= 200 [Data:256]
4	19.594418	10.69.101.89	10.69.101.90	CFLOW	206	total: 4 (v9) records Obs-Domain-ID= 200 [Data-Template:259
5	24.996397	10.69.101.89	10.69.101.90	CFLOW	406	IPFIX flow (364 bytes) Obs-Domain-ID= 256 [Data:262]
6	27.997120	10.69.101.89	10.69.101.90	CFLOW	406	IPFIX flow (364 bytes) Obs-Domain-ID= 256 [Data:262]
7	28.997280	10.69.101.89	10.69.101.90	CFLOW	406	IPFIX flow (364 bytes) Obs-Domain-ID= 256 [Data:262]
8	29.997609	10.69.101.89	10.69.101.90	CFLOW	406	IPFIX flow (364 bytes) Obs-Domain-ID= 256 [Data:262]
9	31.974474	10.69.101.89	10.69.101.90	CFLOW	106	total: 1 (v9) record Obs-Domain-ID= 200 [Data:256]
10	31.998651	10.69.101.89	10.69.101.90	CFLOW	1438	IPFIX flow (1396 bytes) Obs-Domain-ID= 256 [Data:262]
11	31.998768	10.69.101.89	10.69.101.90	CFLOW	1438	IPFIX flow (1396 bytes) Obs-Domain-ID= 256 [Data:262]
12	31.998777	10.69.101.89	10.69.101.90	CFLOW	750	IPFIX flow (708 bytes) Obs-Domain-ID= 256 [Data:262]
13	32.999013	10.69.101.89	10.69.101.90	CFLOW	406	IPFIX flow (364 bytes) Obs-Domain-ID= 256 [Data:262]
14	36.000735	10.69.101.89	10.69.101.90	CFLOW	1438	IPFIX flow (1396 bytes) Obs-Domain-ID= 256 [Data:262]
15	36.000795	10.69.101.89	10.69.101.90	CFLOW	1438	IPFIX flow (1396 bytes) Obs-Domain-ID= 256 [Data:262]
16	36.000804	10.69.101.89	10.69.101.90	CFLOW	1438	IPFIX flow (1396 bytes) Obs-Domain-ID= 256 [Data:262]
17	36.000811	10.69.101.89	10.69.101.90	CFLOW	406	IPFIX flow (364 bytes) Obs-Domain-ID= 256 [Data:262]

Obrázek 4.7: Pcap soubor s přehrávaným provozem.

```
▼ Set 1 [id=2] (Data Template): 262
    FlowSet Id: Data Template (V10 [IPFIX]) (2)
    FlowSet Length: 100
  ▼ Template (Id = 262, Count = 18)
      Template Id: 262
      Field Count: 18
    ▶ Field (1/18): IP_SRC_ADDR
    ▶ Field (2/18): IP_DST_ADDR
    ▶ Field (3/18): PROTOCOL
    ▶ Field (4/18): L4_SRC_PORT
    ▶ Field (5/18): L4_DST_PORT
    ▶ Field (6/18): ingressVRFID
    ▶ Field (7/18): INPUT_SNMP
    ▶ Field (8/18): connectionTransactionId
    ▶ Field (9/18): APPLICATION_ID
    ▼ Field (10/18): 9357 [pen: ciscoSystems]
        1... .... .... .... = Pen provided: Yes
        .010 0100 1000 1101 = Type: 9357 [pen: ciscoSystems]
        Length: 65535 [i.e.: "Variable Length"]
        PEN: ciscoSystems (9)
    ▼ Field (11/18): 12235 [pen: ciscoSystems]
        1... .... .... .... = Pen provided: Yes
        .010 1111 1100 1011 = Type: 12235 [pen: ciscoSystems]
        Length: 65535 [i.e.: "Variable Length"]
        PEN: ciscoSystems (9)
    ▼ Field (12/18): 12235 [pen: ciscoSystems]
        1... .... .... .... = Pen provided: Yes
        .010 1111 1100 1011 = Type: 12235 [pen: ciscoSystems]
        Length: 65535 [i.e.: "Variable Length"]
        PEN: ciscoSystems (9)
```

Obrázek 4.8: Detail pcap souboru s přehrávaným provozem.

[6]Elementy s variabilní délkou mají proměnlivou délku na základě dat. Skutečná délka daného elementu je získána z šablony `https://tools.ietf.org/html/rfc5101`

V obrázku 4.8 je vidět detail IPFIX toku, který obsahuje položky označené jako *Variable lenght*, tedy položky s variabilní délkou.

Pro nasimulování chyby jsem spustil test varibilních délek s verzí nfdump nástrojů, ve které ještě nebyly implementovány. Test tedy skončil s chybou.

```
[root@localhost test]# ./test tests/Variable_lenghts/variable_lenghts.ini
ConfigFile checked.
Starting nfcapd on port: 16665
Replaying pcaps.
tests/Variable_lenghts/cisco_variable_lenghts.pcap
Waiting 70 secs for nfcapd processing:
Terminating nfcapd.
Checking syslog for errors.
Comparing nfdump commands.
Problem with command linesbetween which ends on line: 43
Comparing error: there is nothing between linesbetween_start and linesbetween_end.
In file: tests/Variable_lenghts/test_variable_lengths_outputs/nfdump_outputs/nfdump_hhost
For more details see: tests/Variable_lenghts/test_variable_lengths_outputs/comparing

FAILED;
```

Obrázek 4.9: Vystup testu.

Samotný výstup neobsahuje podrobnější infromace, pouze číslo řádku, porovnávací příkaz a s jakým nfdump příkazem skončil test s chybou a cestu k adresáři s podrobnějšími informacemi.

Test byl spuštěn s konfiguračním souborem v příloze: A. Konfigurační soubor byl vytvořen na základě provozu. V detailu vstupního pcap souboru je vidět provoz, který je přehrán, zejména pak položky označené pen[7] *ciscoSystems* s popisem *Variable lengths* z nichž jedna odpovídá *Host URL*. Test tedy bude kontrolovat, zda obsahuje korektní URL, ale vzhledem k variabilním délkám položek by mohlo dojít k tomu, že se některým položkám přiřadí nesprávná délka a tím pádem vzniknou nesprávná data, proto test obsahuje kontrolu i pro časové známky, nfcapd položky zarovnává automaticky takže to není potřeba řešit, ale nesmí nastat situace kdy by byla časová značka např. 31. února. Podobným způsobem je kontrolována korektní IP adresa.

Při běhu testu se vytvoří složka *variable_lengths_test_outputs*, určená položkou *name* v konfiguračním souboru, v rámci níž se poté vytvoří sloužka *nfcapd_files* se všemi výstupními soubory. Tato složka pak může sloužit k případné manuální kontrole.

Dále se vytvoří podsložka *nfdump_outs*, která obsahuje výstupy pro jednotlivé sekce *nfdump_X*, každý výstup tedy odpovídá samostatnému spuštěním nfdump nástroje. Konkrétní výstup pro *[nfdump_hurl]* vypadá následovně.

```
Date first seen          Duration      Src IP Addr      Dst IP Addr  HTTP hostname   HTTP URL
Dst Pt  Packets    Bytes
Summary: total flows: 0, total bytes: 0, total packets: 0, avg bps: 0, avg pps: 0, avg bpp: 0
Time window: 2016-02-29 14:08:40 - 2016-02-29 14:13:39
Total flows processed: 2621, Blocks skipped: 0, Bytes read: 843078
Sys: 0.007s flows/second: 327706.9   Wall: 0.001s flows/second: 2115415.7
```

Obrázek 4.10: Vystup z nfdumpu.

Ve výstupu je vidět, že je zde 0 zobrazených toků místo 105 podle očekávání, ale zpracováno jich bylo 2621 přesně podle očekávání. Důvodem je, že nfdump nenajde žádné položky

[7] Private enterprise number

odpovídající filtru *'hurl'*, proto se ani žádné toky nezobrazí, což jednoznačně dokazuje, že nfdump nástroje této variabilní položce nerozumí.

Poslední výstupní soubor, který se vytváři je *compare.log*. Ten obsahuje detaily porovnávání.

```
Executing: linesbetween_start      =   ^Date.*
OK
Executing: linesbetween_end        =   ^Summary.*
OK
Executing: linesbetween_between    =   ^(20|1[1-9])[0-9]{2}-((0(1|[3-9])|1[0-2])-([0-2][0-9]|3[0-1])|
02-[0-2][0-9])\s([0-1][0-9]|2[0-3])(:[0-5][0-9]){2}.[0-9]{3},.*
Comparing error: there is nothing between linesbetween_start and linesbetween_end.
```

Obrázek 4.11: compare.log s detaily porovnávání.

4.5 Integrace do systému Jenkins

Integrace do systému Jenkins[8] je nastavena tak, aby vše probíhalo zcela automaticky. Skládá se z několika kroků:

- Vytvoření Flowmon stroje do prostředí ESXi[9].
- Přeložení a instalace nejnovějších nfdump nástrojů a Flowmon exportéru
- Spuštění testů
- Vyhodnocení testů

Vytvoření Flowmon stroje probíhá pomocí vSphere Management SDK[10], při kterém se musí zajistit korektní vytvoření stroje, nahrání licence, aby se aktivovala síťová rozhraní a správné nastavení síťových rozhraní, tak aby odpovídalo zapojení v kapitole Příprava prostředí 4.3.1.

Překládá se vždy vývojová větev nfdump nástrojů a exportéru. Následně se nainstalují na vytvořeném stroji. Také se stáhnou testy. Toto zajišťují další pomocné pythonovské skripty.

Spuštění testů je potřeba provést s rootovskými právy, z tohoto důvodu bylo nejjednoduší zase využít vSphere Management SDK, dále jsou spuštěny pomocné skripty *prepare.sh* a *stop_exporters.sh* a poté už *run_tests*, který spustí veškeré testy.

Posledním krokem je vyhodnocení testů. Skript *run_tests* vytváří logovací soubor 4.12, který je v tomto kroku potřeba rozdělit na jednotlivé testy a zkontrolovat. Znovu probíhá pomocí pythonovských skriptů.

Obrázek 4.13 zobrazuje spuštění testů po sérii úprav. Testy zaznamenaly úpravy. Ve většině případů se jednalo pouze o úpravu výstupů některých pluginů, ale v několika případech došlo úpravami i k chybám.

Detail výpisu na obrázku 4.14 zobrazuje stejné informace jako jednotlivé testy. Účelem systému Jenkins je automatizované náplánování spuštění každý den, vytvoření nového virtuálního stroje, příprava prostředí a spuštění testů.

[8]Systém Jenkins je nástroj sloužící k automatizaci projektů `https://jenkins.io/`

[9]Prostředí pro vytváření a obsluhu virtuálních počítačů `https://en.wikipedia.org/wiki/VMware_ESXi`

[10]https://code.vmware.com/web/sdk/60/vsphere-management

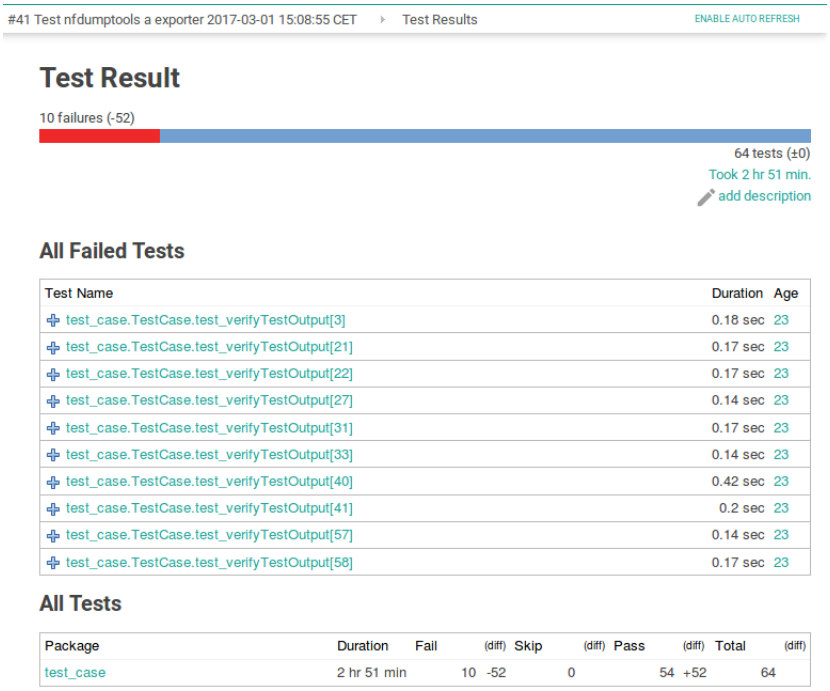
```
INFO:root:08.03.2017 04:42:56
INFO:root:Running: ./home/flowmon/test/test /home/flowmon/test/tests/VOIP/test_real_dump.ini
INFO:root:/home/flowmon/test/tests/VOIP/test_real_dump.ini
ERROR:root:Problem with line: 32
findline = \s*192\.168\.26\.144,\s*23000,\s*4151926929@192_168_1_150,\s*sip:402@tester\.optimcall\.cz,
\s*sip:402@tester\.optimcall\.cz,\s*SIP\/2\.0\/UDP 192\.168\.1\.150:5060,\s*0,\s*0,\s*6,\s*0\.0\.0\.0,\s*0,\s*0,\s*1,\s*0,\s*0,\s*0,
\s*0,\s*0,\s*0
Comparing error: Findline found nothing.
Take a look to output: /home/flowmon/test/tests/VOIP/voip_real_dump_test_outputs/nfdump_outputs/nfdump_voip_call
For more details see: /home/flowmon/test/tests/VOIP/voip_real_dump_test_outputs/comparing
FAILED;


INFO:root:08.03.2017 04:45:24
INFO:root:Running: ./home/flowmon/test/test /home/flowmon/test/tests/DHCP/PRIV_bootp-both_overload.ini
INFO:root:/home/flowmon/test/tests/DHCP/PRIV_bootp-both_overload.ini
INFO:root:OK;

INFO:root:08.03.2017 04:47:52
INFO:root:Running: ./home/flowmon/test/test /home/flowmon/test/tests/HTTP_methods/test_http_patch_ssl.ini
INFO:root:/home/flowmon/test/tests/HTTP_methods/test_http_patch_ssl.ini
INFO:root:OK;

INFO:root:08.03.2017 04:50:22
INFO:root:Running: ./home/flowmon/test/test /home/flowmon/test/tests/HTTP_methods/test_http_post.ini
INFO:root:/home/flowmon/test/tests/HTTP_methods/test_http_post.ini
INFO:root:OK;

INFO:root:08.03.2017 04:52:52
INFO:root:Running: ./home/flowmon/test/test /home/flowmon/test/tests/HTTP_methods/test_http_options.ini
INFO:root:/home/flowmon/test/tests/HTTP_methods/test_http_options.ini
INFO:root:OK;

INFO:root:08.03.2017 04:55:23
INFO:root:Running: ./home/flowmon/test/test /home/flowmon/test/tests/HTTP_methods/test_http_copy.ini
INFO:root:/home/flowmon/test/tests/HTTP_methods/test_http_copy.ini
INFO:root:OK;

INFO:root:08.03.2017 04:57:54
INFO:root:Running: ./home/flowmon/test/test /home/flowmon/test/tests/HTTP_methods/test_http_head.ini
INFO:root:/home/flowmon/test/tests/HTTP_methods/test_http_head.ini
INFO:root:OK;

INFO:root:08.03.2017 05:00:24
INFO:root:Running: ./home/flowmon/test/test /home/flowmon/test/tests/HTTP_methods/test_http_delete.ini
INFO:root:/home/flowmon/test/tests/HTTP_methods/test_http_delete.ini
INFO:root:OK;

INFO:root:08.03.2017 05:02:56
INFO:root:Running: ./home/flowmon/test/test /home/flowmon/test/tests/HTTP_methods/test_http_put.ini
INFO:root:/home/flowmon/test/tests/HTTP_methods/test_http_put.ini
INFO:root:OK;

INFO:root:Finished
INFO:root:Summary: 61 tests from 62 passed!
```

Obrázek 4.12: Logovaci soubor vytvoreny *run_tests*

4.6 Shrnutí

Testovací framework pro Flowmon kolektor a Flowmon exportér se používá na denní bázi, vždy o půlnoci testuje všechny procesní pluginy exportéru a tedy i všechny skupiny rozšíření nfdump nástrojů. Tím, že se testuje každý den lze poměrně jednoduše detekovat co chybu/změnu chování (každá změna oproti testům nemusí být nutně chyba, ale pouze třeba změna některých položek) způsobilo a sjednat okamžitou nápravu.

Toto testování je důležité zejména proto, že do vývojové větve přispívá mnoho programátorů a i když jednotlivé části mohou fungovat bezchybně, typicky rozšíření nfdump nástrojů vývojáři testují pouze přímo načtením pcap souboru, nicméně exportér může po nějaké změně posílat lehce odlišné položky, dohromady tedy nemusí být ve vývoji vždy konzistentní, a proto je potřeba testovat celý backend jako celek.



Obrázek 4.13: Příklad výstupu ze systému Jenkins.

Stacktrace

```
cls = <test_case.TestCase object at 0x7faafc6d4400>, NUMBER = 3

    def test_verifyTestOutput(cls, NUMBER):
        #test, ktery postupne bude parsovat output
        global tests_failed
        if(tests_failed == 0):
            TestClass = ParseTestClass(cls.flowmon_ip)
            ret_val = TestClass.run_parse_test(NUMBER, cls.count)
>           assert (ret_val[0] == "OK"), "Test " + str(NUMBER) + "Failed. In
stack trace should be all useful information." + ret_val[1]
E           AssertionError: Test 3Failed. In stack trace should be all useful
information.
E
E             INFO:root:24.04.2017 02:25:50
E             INFO:root:Running: ./home/flowmon/test/test /home/flowmon
/test/tests/VOIP/test_dump.ini
E             INFO:root:/home/flowmon/test/tests/VOIP/test_dump.ini
E             ERROR:root:Problem with line: 25
E             findline = \s*85\.93\.163\.139,\s*192\.168\.26\.144,\s*23000,
\s*2146406845-5074-1@BJC\.BGI\.B\.EC,\s*sip:0111@chladnicek\.optimcall
\.cz:23000,\s*sip:0111@chladnicek\.optimcall\.cz:23000,\s*SIP\/2\.0\/UDP
192\.168\.1\.42:5074,\s*0,\s*0,\s*2,\s*0\.0\.0\.0,\s*0,\s*0,\s*1,\s*0,\s*0,
\s*0,\s*0,\s*0,\s*0
E             Comparing error: Findline found nothing.
E             Take a look to output: /home/flowmon/test/tests
/VOIP/voip_dump_test_outputs/nfdump_outputs/nfdump_voip_call
E             For more details see: /home/flowmon/test/tests
/VOIP/voip_dump_test_outputs/comparing
E             FAILED
E           assert 'FAILED' == 'OK'
E             - FAILED
E             + OK

test_case.py:160: AssertionError
```

Obrázek 4.14: Detail výstupu chyby.

Kapitola 5

Testování Flowmon ADS

Zde bude podrobně rozepsáno testování Flowmon ADS od počátečních požadavků přes podrobný návrh testovacího frameworku až po výslednou strukturu. Součástí této kapitoly je pak také popis a možnosti spuštění frameworku a případová studie.

5.1 Požadavky na framework

Požadavkem bylo vytvořit framework pro automatizované testování Flowmon ADS. Do frameworku je zahrnuto i importování nové konfigurace a mazání dat z ADS databáze, čímž nepřímo ověřuje fungování těchto funkcí. Požadavky tedy jsou:

1. Možnost automatizovaného testování Flowmon ADS konfigurovatelné vstupním souborem.
2. Spustitelnost na Flowmon kolektoru (založený na CentOS 5).
3. Testování v prostředí VMware ESXi[1].
4. V jazyce Python verze 2.6.

5.2 Návrh frameworku

Framework byl navržen s podobnými požadavky jako framework pro testování kolektoru, tedy s důrazem na univerzálnost a rozšiřitelnost. Testování bylo navrženo jako white-box testování, očekávané výstupy jsou upraveny díky možnosti náhledu do vnitřní struktury (např. při testování nové metody není potřeba testovat formát časové značky, protože to není v režii Flowmon ADS).

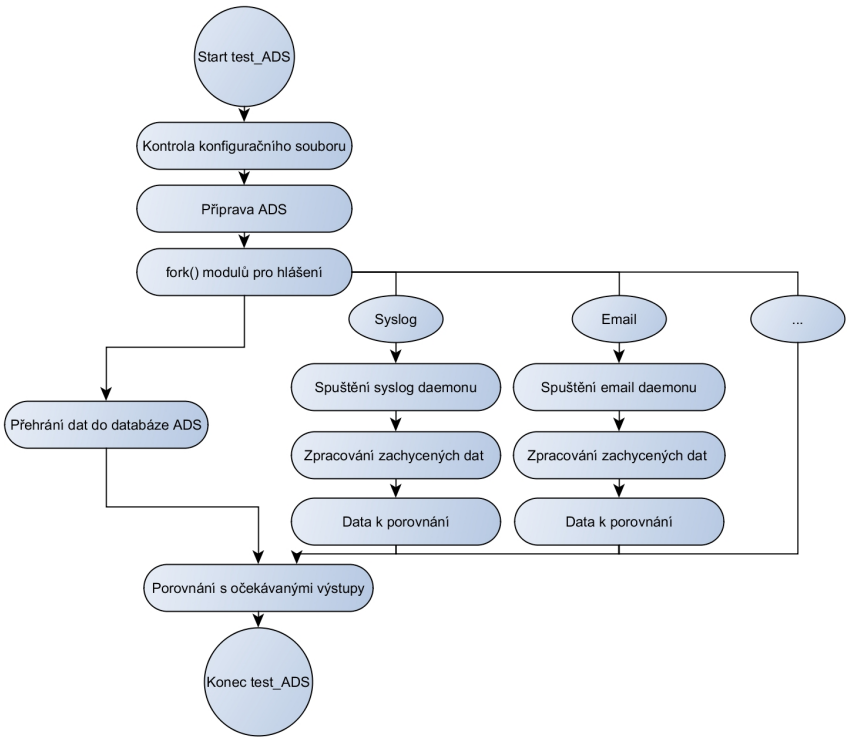
Výpis 5.1: Adresářová struktura frameworku

```
ADS_tests/                  (kořenový adresář)
 -test_ADS                  (hlavní pythonovský skript)
 -prepare.sh                (pomocný bashovský skript)
 -syslog_daemon.py
 -snmp_daemon.py
```

[1]Prostředí pro vytváření a obsluhu virtuálních počítačů `https://en.wikipedia.org/wiki/VMware_ESXi`

```
-modules/                  (složka s moduly pro skript "test_ADS")
  -checking_modules/
    -email.py
    -syslog.py
    -snmp.py
    -custom_script.py
    -shared_functions.py
  -parser.py
  -process.py
-tests/                    (složka zastřešující konfigurační soubory)
  -example_test/
    -config.ini
    -nfcapd_files/         (složka s nfcapd soubory)
     -year/
      -month/
       -day/
        -file.nfcapd
    -config.xml
-test_ADS_outputs/         (složka pro výstupy z testu)
  -example_test/
    -error.log
    -email.log
    -syslog.log
    ...
```

- Celý framework je modulárně rozdělen. Framework se spouští pomocí hlavního pythonovského skriptu *test_ADS*, který jako první zavolá modul *parser.py*.



Obrázek 5.1: Diagram datových toků.

- Modul *parser.py* zkontroluje konfigurační soubor a uloží jednotlivé konfigurační položky do vnitřní struktury. Po kontrole se připravuje prostředí ADS a následně se spouští jednotlivé moduly pro kontrolu hlášení událostí jako *fork* hlavního skriptu. Poté se spustí ads-nf-loader, který na vstup bere nfcapd soubory a postupně nahrává provoz do databáze. Moduly zpracují zprávy, které zachytí a vrátí události ve struktuře, která obsahuje:

  1. Název události
  2. Zdrojovou IP adresu
  3. Časovou značku události
  4. Detailní popis události
  5. Cílové IP adresy

  Následně se zpracují přijaté struktury a a porovnají se s očekávanými výsledky. Očekávané výsledky jsou v sekci *event* v konfiguračním souboru, která obsahuje počet očekávaných událostí, které se musí zpracovat a další podrobnosti. Navíc je prováděno porovnávání, zda všechny moduly pro kontrolu hlášení událostí zpracují stejné události. Podrobnosti budou rozebrány v následující sekci.

- Adresářová struktura je rozčleněna na složku *modules*, která obsahuje moduly pro hlavní pythonovský skript *test_ADS*, složku *tests*, která zastřešuje konfigurační soubory spolu s nfcapd soubory pro přehrávání provozu, případně konfigurační soubor pro ADS ve formě xml.

  Složka *nfcapd_files* obsahuje všechny nfcapd soubory. Nástroj ads-nf-loader tuto složku prohledává a podle časové značky přehrává jednotlivé soubory. Je potřeba zachovat strukturu `/nfcapd_files/year/month/day`.

- Pomocný bashovský skript *prepare.sh* zajistí správné nastavení externího syslog serveru a emailového serveru pro odesílání hlášení pomocí ADS.

- Při běhu frameworku se vytváří složka *test_ADS_outputs*, ve které se pro jednotlivé testy vytvářejí výstupní soubory, které zahrnují logovací soubory pro spuštěné moduly hlášení událostí a *error.log* s podrobnostmi při případných chybách.

5.2.1 Gramatika konfiguračního souboru

Konfigurační soubor určuje běh celého frameworku. Obsah konfiguračního souboru je před spuštěním kontrolován, aby odpovídal určité gramatice. Zde zapsáno v ABNF normě.

Výpis 5.2: Gramatika konfiguračního souboru

```
configFile = common_settings
             1*event
             *comment

common_settings  = "[common_settings]" CRLF reset snmp syslog email
                   custom_script statistics [config] [timewait]
reset            = "reset=" ""/"data"/"all" CRLF
snmp             = "snmp=" "0"/"1" CRLF
syslog           = "syslog=" "0"/"1" CRLF
email            = "email=" "0"/"1" CRLF
```

```
custom_script    = "custom_script=" "0"/"1" CRLF
statistics       = "statistics=" "0"/"1" CRLF
config           = "config=" PATH_TO_XML CRLF
timewait         = "timewait=" NUMBER CRLF

event = "[event_" STRING "]" CRLF count eventtype
        [timestamp] [source] [detail] [targets]

count      = "count" OPERAND NUMBER CRLF
eventtype  = "eventtype=" STRING CRLF
timestamp  = "timestamp=" TIME CRLF
source     = "source=" IP_ADDRESS CRLF
detail     = "detail=" REGULAR_EXPRESION CRLF
targets    = "targets=" IP_ADDRESS ["," IP_ADDRESS] CRLF
comment    = "#" STRING CRLF
```

Konfigurační soubor je složen z několika částí. Všechny kromě komentářů jsou povinné a některé se mohou opakovat.

Sekce *common_settings* obsahuje možnost resetování, která určuje, zda resetovat celé nastavení ADS, data v databázi nebo neresetovat nic. Dále obsahuje, které moduly pro hlášení událostí spustit. Jako nepovinný je zde parametr pro nahrání vlastní konfigurace ve formě xml. Framework je možné spustit bez jakéhokoli modulu pro hlášení událostí, čímž lze nepřímo ověřit funkčnost resetování a případně nahrávání konfigurace. Poslední nepovinný parametr *timewait*, který je vhodný použít u emailového modulu nebo modulu uživatelský skript, které události hlásí události podle priority i po několika hodinách.

Sekce *event* obsahuje podrobnosti o kontrole událostí. Parametr *count* udává kolikrát se musí událost zahlásit. Lze využít operandy pro porovnávání (=, <, <=, >, >=). Prametr *eventtype* je název události, která se má kontrolovat. Doposud zmíněné položky jsou povinné, další pouze rozšiřují kontrolu událostí o další detaily. *timestamp*, kontrola času kdy událost vznikla, *source* ip adresa zdroje události, *detail* obsahuje obvykle textový popis události a *targets* seznam cílů události. Tímto způsobem pak lze porovnávat celkový počet událostí, ale i detailně vyhledávat nějaké určité události.

5.2.2 Testovací scénáře

Při testování ADS je potřeba ověřit, že se správně generují události. Generované události jsou dvojího typu: jednoduché a naučené. Jednoduché události vznikají, když se objeví určitý provoz např. MULTICAST, tedy událost, která informuje o tom, že v síti se objevil nějaký multicastový provoz. Událostí, která potřebuje čas k učení, může být HTTPDICT, která označuje událost, kdy dochází v síti ke slovníkovému útoku na webovou službu, nebo DOS, která označuje DoS útok. ADS k těmto událostem potřebuje nějaký čas typicky alespoň dvě dávky. Dávka je pětiminutový časový interval s provozem a odpovídá jednou souboru typu nfcapd.

Z toho vyplývá, že existují dva testovací scénáře. Pro jednoduché události:

1. Přehrát provoz pomocí nástroje ads-nf-loader.
2. Ověřit, že se vygenerují očekávané jednoduché události.

Pro události s učením:

1. Vymazat data z databáze.
2. Přehrát první dávku s provozem.

3. Přehrát další dávky s provozem.

4. Ověřit očekávané naučené události.

5.3 Spuštění frameworku

Před spuštěním frameworku je potřeba spustit pomocný bashovský skript *prepare.sh*, který do konfigurace ADS nastaví syslog, email a další servery pro testování událostí.

Framework se spouští pomocí skriptu *test_ADS*. Jeho jediným parametrem je cesta ke konfiguračnímu souboru, který obsahuje všechny další informace o spuštění.

Doba běhu závisí na mnoha proměnných. Je závislá na velikosti i typu provozu, zatížení stroje a stavem databáze ADS před spuštěním, proto odhadnout nelze.

5.4 Případová studie

Následující příklad zobrazuje detekci DOS události. Událost DOS je jednou z učených událostí, proto je potřeba více dávek k naučení. Generování události DOS je založena na několika různých metodách detekce. Například metoda založená na kontrole poměru přijatých a odeslaných paketů jednotlivými zařízeními v síti. S využitím předchozích dat je pro každou dávku předpovězena hranice, při jejímž překročení dojde k vygenerování události. Další metodou může být detekce kombinace jednoduchých událostí, např. velké množství provozu od jednoho klienta a zároveň skenování portů serveru apod.

Na následujícím obrázku je vidět jak test dopadne v případě, že se přehraje pouze jedna dávka zachyceného provozu.

```
[root@localhost ADS_tests]# ./test_ADS tests/DOS/dos.ini
Checking configuration file:
OK
Preparing settings:
ResetData
OK
Upload new configuration
OK
Forcing new configuration
OK
Stopping live profiles
OK
Processing started:
Tue Apr 18 14:41:27 CEST 2017
Starting modules
Starting syslog
Starting email
Waiting for processing by all active modules.
ads-nf-loader: started
OK
Email: waiting: 0 hours (+1 minute) for all emails
Email: 3 events reported
Syslog: 3 events reported
Comparing
Event: Type:DOS, count:1, source:None, timestamp:None, targets:None, detail:None
syslog found: 0
Event: Type:SCANS, count:1, source:None, timestamp:None, targets:None, detail:None
syslog found: 0
Event: Type:HIGHTRANSF, count:6, source:None, timestamp:None, targets:None, detail:None
syslog found: 2
Event: Type:SRVNA, count:3, source:None, timestamp:None, targets:None, detail:None
syslog found: 1
Event: Type:DOS, count:1, source:None, timestamp:None, targets:None, detail:None
email found: 0
Event: Type:SCANS, count:1, source:None, timestamp:None, targets:None, detail:None
email found: 0
Event: Type:HIGHTRANSF, count:6, source:None, timestamp:None, targets:None, detail:None
email found: 2
Event: Type:SRVNA, count:3, source:None, timestamp:None, targets:None, detail:None
email found: 1
comparing modules between each other
More infromation in: test_ADS_output/DOS/error.log
FAILED
[root@localhost ADS_tests]#
```

Obrázek 5.2: DOS po přehrání jedné dávky.

Vygenerují se pouze dva typy událostí a to SRVNA - Služba je nedostupná a HIGHTRANFS - Velké množství provozu, které patří mezi jednoduché události, proto se vygenerují hned jakmile se objeví jejich signatura v provozu. Pro SRVNA to může být například *Destination Host Unreachable* odpověď na `ping`. Událost HIGHTRANFS se vygeneruje pokud některý klient v síti odesílá velké množství dat, ve výchozím nastavení je to 5 GBs.

Provoz byl simulován pomocí nástroje `hping`[2], pomocí kterého byl generován *TCP SYN flood*[3], tedy bylo posíláno velké množství TCP packetů s příznakem SYN[4]. Server se na tyto požadavky snaží odpovědět příznakem ACK[5], ale již bez další odpovědi.

Na dalším obrázku je vidět seznam zachycených událostí. Obsahuje 2 události HIGHTRANFS, které se vygenerovaly pro klienta zahlcujícího server SYN požadavky a i pro opačný směr, tedy server odpovídající ACK příznaky. Další událost SRVNA značí, že se serveru nepodařilo ustavit spojení.

```
<182>Apr 18 14:47:33 localhost ADS: CEF:0|Flowmon Networks|Flowmon ADS Business|9.00.00|
HIGHTRANSF|High volume of transferred data|10|src=10.0.0.1 start=Apr 05 2017 13:35:00
msg=Transferred: 6.83 GiB, top peer transfer: 6.83 GiB. targetList: 10.0.0.2
<182>Apr 18 14:47:45 localhost ADS: CEF:0|Flowmon Networks|Flowmon ADS Business|9.00.00|
HIGHTRANSF|High volume of transferred data|10|src=10.0.0.2 start=Apr 05 2017 13:35:00
msg=Transferred: 6.83 GiB, top peer transfer: 6.83 GiB. targetList: 10.0.0.1
<182>Apr 18 14:49:07 localhost ADS: CEF:0|Flowmon Networks|Flowmon ADS Business|9.00.00|SRVNA|
Service not available|10|src=10.0.0.1 start=Apr 05 2017 13:35:00 msg=Unavailable services
(TCP/80, http). Unsuccessful traffic - clients: 1, rejected connections: 0, connections without
response: 131040. Successful traffic - clients: 0, connections: 0. targetList: 10.0.0.2
```

Obrázek 5.3: Seznam zachycených událostí pomocí syslog serveru.

```
[root@localhost ADS_tests]# ./test_ADS tests/DOS/dos.ini
Checking configuration file:
OK
Preparing settings:
ResetData
OK
Upload new configuration
OK
Forcing new configuration
OK
Stopping live profiles
OK
Processing started:
Tue Apr 18 14:23:03 CEST 2017
Starting modules
Starting syslog
Starting email
Waiting for processing by all active modules.
ads-nf-loader: started
OK
Email: waiting: 0 hours (+1 minute) for all emails
Email: 11 events reported
Syslog: 11 events reported
Comparing
Event: Type:DOS, count:1, source:None, timestamp:None, targets:None, detail:None
syslog found: 1
Event: Type:SCANS, count:1, source:None, timestamp:None, targets:None, detail:None
syslog found: 1
Event: Type:HIGHTRANSF, count:6, source:None, timestamp:None, targets:None, detail:None
syslog found: 6
Event: Type:SRVNA, count:3, source:None, timestamp:None, targets:None, detail:None
syslog found: 3
Event: Type:DOS, count:1, source:None, timestamp:None, targets:None, detail:None
email found: 1
Event: Type:SCANS, count:1, source:None, timestamp:None, targets:None, detail:None
email found: 1
Event: Type:HIGHTRANSF, count:6, source:None, timestamp:None, targets:None, detail:None
email found: 6
Event: Type:SRVNA, count:3, source:None, timestamp:None, targets:None, detail:None
email found: 3
comparing modules between each other
OK
[root@localhost ADS_tests#
```

Obrázek 5.4: DOS po přehrání více dávek.

[2]Dostupný na `http://www.hping.org/`

[3]Podrobnosti útoku: `https://cs.wikipedia.org/wiki/SYN_flood`

[4]synchronizovat

[5]potvrdit

Pro vygenerování DOS události ADS potřebuje čas k učení alespoň dvě dávky. Obrázek 5.4 ukazuje jak test dopadne při přehrání více dávek, konkrétně byly přehrány celkem 3 dávky.

Z obrázku je vidět, že se vygenerovala událost DOS. Navíc přibyly další údálosti HIGHTRANFS a SRVNA, které se generují pro každou dávku.

5.5 Shrnutí

Testovací framework pro Flowmon ADS slouží k testování kontrolních hlášení generovaných událostí. Ověřuje, že Flowmon ADS vygeneruje ze stejného provozu, stejné události, tedy chová se deterministicky. Zatím se však pravidelně nepoužívá. Je potřeba dovytvořit testovací sadu pro všechny události, aby bylo ověřeno, že při přidání nového druhu události nebo změny stávájích, nedojde ke změně ostatních.

V současné době lze testovat hlášení událostí pouze z modulů email a syslog, bylo by tedy potřeba dodělat moduly SNMP a uživatelský skript. Nicméně pokud je potřeba otestovat nově přidávané metody detekce chyb jsou tyto moduly dostačující, zejména proto, že moduly SNMP i uživatelský skript jsou i mezi zákazníky málo využívané.

Kapitola 6

Závěr

Podařilo se vytvořit framework pro automatizované testování Flowmon kolektoru. Framework je zaintegrován do systému Jenkins a slouží ke každodenním testům, které od svého nasazení pomohly odhalit mnoho menších i větších chyb prakticky okamžitě.

V průběhu implementace se objevilo mnoho věcí, se kterými se v původním návrhu úplně nepočítalo, ale ve výsledku bylo vhodné o ně framework rozšířit. Jednou z těchto věcí může být možnost porovnávat výstupy pomocí testových souborů s očekávanými výsledky, v původním návrhu se počítalo pouze s porovnáváním pomocí regulárních výrazů což v případě většího množství testovaného provozu mohlo být velice pracné. Další věcí byla možnost vypsat si výstup přímo z exportéru, protože pomocí této možnosti lze poměrně jednoduše lokalizovat zda případná chyba nastala v exportéru či nfdump nástrojích.

Do budoucna je možné, že se framework bude stále rozšiřovat, což by neměl být problém vzhledem k tomu, že jsem se vše snažil psát univerzálně a rozšiřitelně. Jedna z možností, která by šla určitě vylepšit je nějaký sofistikovanější odhad doby běhu, nicméně vzhledem k tomu, že framework slouží hlavně k automatizovanému testování na systému Jenkins, který výsledky strojově zpracovává není to priorita.

Framework pro automatizované testování Flowmon ADS je hotový pouze z větší části. Aktuálně lze testovat hlášení událostí pouze z modulů email a syslog. Moduly pro SNMP a uživatelský skript nejsou implementované, což ale nebrání testování nově implementovaných metod.

Do budoucna se pravděpodobně bude dodělávat statistika jak dlouho trvá zpracování dávky, což by zároveň mohlo sloužit jako výkonostní testy. Také bude pro úplnost potřeba dodělat chybějící moduly, přestože jsou i mezi zákazníky málo využívané.

Literatura

[1] BEIZER, B.: *Software Testing Techniques*. Dreamtech, 2003.

[2] CROCKER, D.; OVERELL, P.: Augmented BNF for Syntax Specifications: ABNF. STD 68, RFC 5234, January 2008.

[3] Flowmon Networks: Flowmon ADS. 2016, navštíveno: 25.3.2017.
URL `Dostupné:https://www.flowmon.com/getattachment/3498c8ec-8c61-4f2f-96a7-82a611cecf13/Flowmon-ADS.aspx`

[4] Flowmon Networks: Flowmon kolektor. 2016, navštíveno: 25.2.2017.
URL `Dostupné:https://www.flowmon.com/getattachment/8d285feb-a58a-4d27-a643-43b24c868859/Flowmon-netflow-ipfix-collector-spec.aspx`

[5] Flowmon Networks: Řešení Flowmon. 2016, navštíveno: 25.2.2017.
URL `Dostupné:https://www.flowmon.com/getattachment/dce90edf-20a2-4366-8c99-e2145f938ac9/Flowmon-Solution-Product-Brief.aspx`

[6] Flowmon Networks: Řešení Flowmon - schéma. 2016, navštíveno: 25.2.2017.
URL `https://www.flowmon.com/cs/products/flowmon`

[7] JORGENSEN, P. C.: *Software testing - A craftsman s Approach*. CRC Press, 2014.

[8] PATTON, R.: *Software testing*. Indianapolis : Sams Publishing, 2006.

Přílohy

Příloha A

Případová studie - Konfigurační soubor pro Flowmon kolektor framework

Výpis A.1: Konfigurační soubor

```
*Pozn. části označené hvězdičkami jsou pouze komentáře do konfiguračního souboru jinak
    nepatří*
*Sekce common settings obsahuje obecná nastavení frameworku. V tomto případě je vidět,
    že se spustí bez exportéru a je zapnutá kontrola syslogu*
[common_settings]
name                    =   test_variable_lengths
exporter                =   no
check_syslog            =   yes

*Sekce nfcapd settings definuje jak se bude spouštět nfcapd. V tomto případě se spustí
    se všemi rozšířeními na portu 16665.*
[nfcapd_settings]
ext                     =   -T all
port                    =   16665

*Sekce pcap se může opakovat a definuje jaké pcap soubory a jak budou přehrávany. Zde
    se lokálně přehraje cisco_variable_lenghts.pcap rychlostí tak jak byl zachycen*
[pcap1]
path                    =   cisco_variable_lenghts.pcap
nspeed                  =   1
replay                  =   l

*Jednotlivé sekce nfdump command určují spouštění nfdump nastroje. nfdump_hurl zobrazí
    prvních 1000 položek v určitém formátu s filtrem ńhurl:*
[nfdump_hurl]
args                    =   -c '1000' -o 'fmt:%ts,%td,%sa,%da,%hhost,%
    hurl,%dp,%pkt,%byt' 'hurl'
linesbetween_start      =   ^Date.*
linesbetween_end        =   ^Summary.*
*Na všechny řádky mezi Date a Summary se aplikuje regex, který zkontroluje korektní
    datum*
linesbetween_between    =   ^(20|1[1-9])[0-9]{2}-((0(1|[3-9])|1[0-2])
    -([0-2][0-9]|3[0-1])|02-[0-2][0-9])\s([0-1][0-9]|2[0-3])(:[0-5][0-9])
    {2}.[0-9]{3},.*
```

```
linesbetween_start      =   ^Date.*
linesbetween_end        =   ^Summary.*
*Na všechny řádky mezi Date a Summary se aplikuje regex, který zkontroluje korektní IP
    adresu*
linesbetween_between    =   .*(\s+(((1[0-9]{2})|(2(([0-4][0-9])|(5[0-5]))
    )|([1-9][0-9]?))\.)(((1[0-9]{2})|(2(([0-4][0-9])|(5[0-5])))
    |(0|[1-9][0-9]*))\.){2}(((1[0-9]{2})|(2(([0-4][0-9])|(5[0-5])))
    |(0|[1-9][0-9]*)),){2}.*

linesbetween_start      =   ^Date.*
linesbetween_end        =   ^Summary.*
*Na všechny řádky mezi Date a Summary se aplikuje regex, který zkontroluje korektní
    URL*
linesbetween_between    =   .*,(/\w+[-_]*\w+/)*(\w+[-_.=&?]*\w+)*\s*,.*

findline                =   ^Summary.*
foundline0              =   .*total flows: 105,.*
foundline2              =   .*flows processed: 2611,.*

*nfdump_hhost zobrazí prvních 1000 položek v určitém formátu s filtrem hhost.*
[nfdump_hhost]
args                    =   -c '10000' -o 'fmt:%ts,%td,%sa,%da,%hhost,%
    hurl,%dp,%pkt,%byt' 'hhost'
linesbetween_start      =   ^Date.*
linesbetween_end        =   ^Summary.*
linesbetween_between    =   ^(20|1[1-9])[0-9]{2}-((0(1|[3-9])|1[0-2])
    -([0-2][0-9]|3[0-1])|02-[0-2][0-9])\s([0-1][0-9]|2[0-3])(:[0-5][0-9])
    {2}.[0-9]{3},.*

linesbetween_start      =   ^Date.*
linesbetween_end        =   ^Summary.*
linesbetween_between    =   .*(\s+(((1[0-9]{2})|(2(([0-4][0-9])|(5[0-5]))
    )|([1-9][0-9]?))\.)(((1[0-9]{2})|(2(([0-4][0-9])|(5[0-5])))
    |(0|[1-9][0-9]*))\.){2}(((1[0-9]{2})|(2(([0-4][0-9])|(5[0-5])))
    |(0|[1-9][0-9]*)),){2}.*

linesbetween_start      =   ^Date.*
linesbetween_end        =   ^Summary.*
linesbetween_between    =   .*,(/\w+[-_]*\w+/)*(\w+[-_.=&?]*\w+)*\s*,.*

findline                =   ^Summary.*
foundline0              =   .*total flows: 105,.*
foundline2              =   .*flows processed: 2611,.*

*nfdump_nofilter zobrazí prvních 1000 položek v určitém formátu bez filteru.*
[nfdump_nofilter]
args                    =   -c '10000' -o 'fmt:%ts,%td,%sa,%da,%hhost,%
    hurl,%dp,%pkt,%byt'
linesbetween_start      =   ^Date.*
linesbetween_end        =   ^Summary.*
linesbetween_between    =   ^(20|1[1-9])[0-9]{2}-((0(1|[3-9])|1[0-2])
    -([0-2][0-9]|3[0-1])|02-[0-2][0-9])\s([0-1][0-9]|2[0-3])(:[0-5][0-9])
    {2}.[0-9]{3},.*

linesbetween_start      =   ^Date.*
linesbetween_end        =   ^Summary.*
linesbetween_between    =   .*(\s+(((1[0-9]{2})|(2(([0-4][0-9])|(5[0-5]))
    )|([1-9][0-9]?))\.)(((1[0-9]{2})|(2(([0-4][0-9])|(5[0-5])))
```

```
    |(0|[1-9][0-9]*))\.){2}(((1[0-9]{2})|(2(([0-4][0-9])|(5[0-5])))
    |(0|[1-9][0-9]*)),){2}.*

findline                =   ^Summary.*
foundline0              =   .*total flows: 2611,.*
foundline2              =   .*flows processed: 2611,.*

*nfdump_A_dstip_srcip zobrazí prvních 10000 položek v určitém formátu seskupené podle
    zdrojové a cílové IP adresy.*
[nfdump_A_dstip_srcip]
args                    =   -c '10000' -A 'srcip,dstip' -o 'fmt:%ts,%td,%
    sa,%da,%hhost,%hurl,%dp,%pkt,%byt'
linesbetween_start      =   ^Date.*
linesbetween_end        =   ^Summary.*
linesbetween_between    =   ^(20|1[1-9])[0-9]{2}-((0(1|[3-9])|1[0-2])
    -([0-2][0-9]|3[0-1])|02-[0-2][0-9])\s([0-1][0-9]|2[0-3])(:[0-5][0-9])
    {2}.[0-9]{3},.*

linesbetween_start      =   ^Date.*
linesbetween_end        =   ^Summary.*
linesbetween_between    =   .*(\s+(((1[0-9]{2})|(2(([0-4][0-9])|(5[0-5]))
    )|([1-9][0-9]?))\.)(((1[0-9]{2})|(2(([0-4][0-9])|(5[0-5])))
    |(0|[1-9][0-9]*))\.){2}(((1[0-9]{2})|(2(([0-4][0-9])|(5[0-5])))
    |(0|[1-9][0-9]*)),){2}.*

findline                =   ^Summary.*
foundline0              =   .*total flows: 2611,.*
foundline2              =   .*flows processed: 2611,.*
```

Příloha B

Manual for nfdump tools and exporter testing framework (Pro potřeby zadavatele v AJ)

Description

- Framework for nfdump tools and Flowmon exporter testing
- Nfdump tools testing is made localy
- Nfdump tools + Flowmon exporter is made remotely
  - Because of this your machine has to be connected to virtual switch TestGenerator by eth2 (all tests run exporter on eth2, if you want to run just your own tests you can connect different interface)
  - And there is VM (PH_nfdumptools_test: 192.168.51.57) where traffic is replayed
- Framework is using tcpreplay for replaying pcaps.
- Framework is configured by configuration (.ini) file
- As output folders with nfcapd files and nfdump outputs are created together with detailed log file

Installation/prerequisites

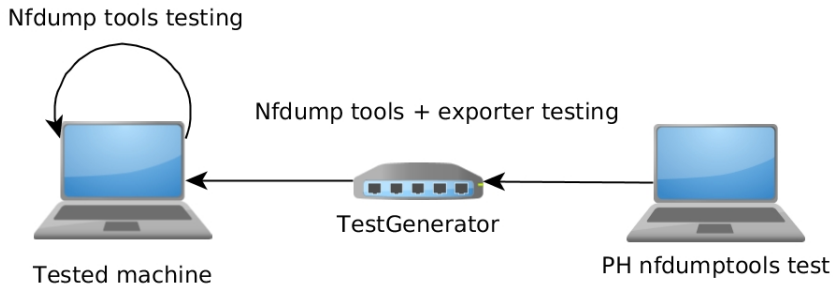
- Framework and all other script must be run as root.
- Before running framework there is a script *prepare.sh*, which create dummy interface for local replaying, install tcpreplay and download all required python libraries
- For exporter testing moreover there is a script *stop_exporters.sh*, which stops default exporters and allows in watchdog to run explicit exporters.
- For exporter testing your machine has to be on the host 192.168.3.110.
- Root folder has to be in `/home/flowmon`.
- All config files have to in folder *tests*.

File structure

```
* Files structure should be like this:
  test/                          (test root folder, has to be in /home/
      flowmon/)
   -test                         (testing script)
   -clean.sh                     (script for cleaning test files)
   -stop_exporters.sh            (script for stoping exporters)
   -run_tests                    (script runs test for all config files in
       folder tests)
   -prepare.sh                   (script for creating dummy inf, download
      tcpreplay, python libraries)
   -modules/                     (modules for testing script)
   -tests/                       (folder with all tests)
     -example_test_local/
       -config.ini               (configuration file)
       -pcaps/                   (folder with pcaps)
       -test_outputs/            (will be created by test)
         -nfcapd_out/            (folder with nfcapd outputs)
         -nfdump_out/            (folder with nfdump outputs)
         -comparing.log          (log file with details about comparing
             regexes)
     -example_test_with_exporter/
       -config.ini
       -exporter.json            (json file with configuration for
          exporter)
       -exporter_template.txt    (template for exporter)
       -other files for exporter plugins
       -pcaps/
```

Testing procedure

- Framework and all other script must be run as root.
- For nfdump tools test only:
  - Run `./prepare.sh download`
  - Put config file to folder tests
  - Run `./test tests/yourFolder/yourConfigFile.ini`
- For nfdump + exporter tests:
  - Interconnect your machine as shown in scheme below
  - Run `./prepare.sh download`
  - Run `./stop_exporters`
  - Put config file to folder tests
  - Run `./test tests/yourFolder/yourConfigFile.ini`
- For running all tests in in folder tests there is a script *run_tests*. The script has one argument, which specifies tests to run:
  `./run_tests ALL` or `./run_tests "Folder_name,Another_folder"`



Obrázek B.1: Testing scheme.

Configuration file

```
#required section
[common_settings]
#required option
name=test_name          (need to be unique in root folder)
exporter=yes/no
check_syslog=yes/no     (check syslog for errors/drops)

#required if exporter is yes
exp_args=json file
exp_out=yes             (if yes, exporter will export also json file)
debug=yes               (if option is yes, test start to write more
    information)

#optional option
change_mtu=mtu          (option for change mtu if bigger packets are going
     to be replayed)
remote=IP adress        (when not used => 192.168.51.57)
remote_user=user        (when not used => flowmon is user)
remote_PW=PW            (when not used => default flowmon PW is used)
remote_root_PW=root_PW (when not used => default flowmon root PW is used)

#required section
[nfcapd_settings]
#required option
ext=extension          (all extensions to start with)

#optional option
port=port_number
path=path to binary    (path to different binary than the installed one)

#required section
[pcapX]                (there can be more sections, with different number
    X)
#required options
path=path              (to pcap file)
nspeed=0/1             (0 means max speed, 1 means normal speed)
replay=l/r             (local/remote)

#required section
[nfdump_X]             (there can be more sections, with different X, X
    can be string, number, whatever)
```

```
#required option
args=args              (all args to start with except -r/-R/-M)

#optional option
path=path to binary    (path to different binary than the installed one)

#functions for comparing

#Compare line X with regex
lineX=regex            (Compare line X with regex => True is Success)
lineX!=regex           (Compare line X with regex => True is Error)

#Compare N lines after X with regex
NlinesX=regex          (Compare N lines after X with regex => all True is
    Success)
NlinesX!=regex         (Compare N lines after X with regex => any True is
    Error)

#Try to find line with matching pattern
findline=regex         (Try to find line with pattern: if find => success
    and point to first occurence)
findline!=regex        (Try to find first line matching pattern: if find =>
     error, else => success, doesnt change found_line)
findNlines=regex       (Try to find N consecutive lines with pattern =>
    sequnce of N True is Success and point to N occurence)

#Try to find next line matching patter, parameter can be empty to use
    last used pattern (by previous findline or findnline)
findnline=regex        (Try to find next line matching pattern => True is
    Success and change found_line)
findnline!=regex       (Try to not find matching pattern => True is Error,
    doesnt change found_line)
findnNlines=regex      (Try to find N next consecutive lines matching
    pattern => True is Success, found_line point to N found line)

#Compare line X + Last_found_line with regex
foundlineX=regex       (Compare line X + found line => True is Success)
foundlineX!=regex      (Compare line X + found line => True is Error)

#Function find start with regex, end with regex and compare everything
    between it
#Its required to use all 3 parts
linesbetween_start=regex      (Find start)
linesbetween_end=regex        (Find end)
linesbetween_between=regex    (Compare all lines between start and end
    with matching pattern => all True is Success)
linesbetween_between!=regex   (Compare all lines between start and end
    with not matching pattern => all False is Success)

#Function for file comparing

file_cmp=file                 (File to compare with, can be used only once
     per nfdump_command)
file_shift=num                (with this option output can be shifted)
                              (for expample: file_shift = 1 means:)
                              (input file will be shifted by 1...so
                                  nfdump_line 0 will be compared with file
                                   line 1)
```

```
                                (negative value affects nfdump output so -1
                                   means:)
                                (nfdump_line 1 will be compared with
                                   file_line 0)
file_ignore=num,num...          (ignore lines with exact number/s)
                                (index starts at 0 so probably -1 according
                                   to text editors)
file_regex_X=regex              (compare line just with regex)
```

Example

```
[common_settings]
name=test1
exporter=no
check_syslog = yes

[nfcapd_settings]
ext=-T all
port=16665
path=/home/flowmon/8.02/bin/nfdump

[pcap1]
path=pcaps/1.pcap
nspeed=0
replay=1

[pcap2]
path=/home/flowmon/pcaps/2.pcap
nspeed=1
replay=1

[nfdump_nofilter]
args = -n '10'
path = /home/flowmon/8.01/bin/nfcapd

#(compare 1  line with regex)
line1 = Date.*

#(find line with Summary.*)
findline = Summary.*

#(Compare line 2 lines after Summary.*)
foundline2 = Total flows processed: 1080.*

#(find line with Time.* after Summary.*)
findnline = Time.*

#(find line with .*?IGNORE.*)
findline = .*?IGNORE.*

#(find line with .*?IGNORE.* after .*?IGNORE.*)
findnline =

#(find sequence of 10 lines matching pattern)
find10lines = .*

#(Find starting line with expresion ^Date.*)
linesbetween_start    = ^Date.*
```

```
#(Find ending line with expresion ^Summary.*)
linesbetween_end    = ^Summary.*

#(Compare everything between it)
linesbetween_between = ^(20|1[1-9])[0-9]*.*

[nfdump_hurl]
args = -c '1000' -o 'fmt:%ts,%td,%sa,%da,%hhost,%hurl,%dp,%pkt,%byt' -6 '
    hurl'
line1 = .*

[nfdump_file_cmp]
args=-n '10'

#file to compare with
file_cmp = /home/flowmon/test/nfdump_test_v3/test_output_cmp

#line to be ignored
file_ignore = 1084, 8, 15, 55, 188, 189, 190

#line 188 will be ignored from file, but will be compared with regex
file_regex_188 = .*
```

Most frequent reasons for unexpected fails

- Forget to use prepare.sh
- Forget to use stop_exporters.sh
- Bad template for exporter/bad path to it (use absolute path, deamonized exporter doesn't really work with relative paths)
- More tests for exporter are running simultaneously, tests for exporter have to run over promiscs switch so have to be run sequely
- You don't have latest Flowmonexp5 or Nfdump tools
- Your VM is running on different than 192.168.3.110 host
- After every update of nftools or flowmonexp is needed to run prepare.sh/stop_exporters.sh like on clean machine.
- If there are some sequence errors try to run test again with normal speed (not top speed)

Příloha C

Manual for ADS testing framework (Pro potřeby zadavatele v AJ)

Description

- Framework for ADS testing
- It's using ads-nf-loader for loading nffiles as batches to ADS.
- Framework is checking reporting modules
  - Syslog - extern syslog server for listening is running on port 11514
  - Email - this module is using python smtpd module for capturing emails from ADS on port 10025
  - SNMP - Not implemented yet
  - Custom script - Not implemented yet
- Framework is reading standard /var/log/daemon for starts and ends of batches

Installation/prerequisites

- Framework and all other script must be run as root.
- Before running framework there is a script *prepare.sh*. This script set up ADS configuration for testing. Set external syslog server, email server etc.
- Root folder has to be in `/home/flowmon`.
- All config files have to in folder *tests*.

File structure

```
ADS_tests/                  (test root folder, has to be in /home/
   flowmon/)
 -test_ADS                  (main python script)
 -prepare.sh                (preparing script)
 -syslog_daemon.py          (external syslog server daemon)
 -modules/                  (folder with modules for test\_ADS script)
   -checking_modules/       (modules for report checking)
```

```
      -email.py
      -syslog.py
      -snmp.py
      -custom_script.py
      -shared_functions.py
    -parser.py              (module for config file parsing)
    -process.py             (module for test evaluation)
  -tests/                   (folder with all confguration files)
    -example_test/
      -config.ini
      -nfcapd_files/        (folder with nfcapd files)
       -year/
        -month/
         -day/
          -file.nfcapd
      -config.xml
  -test_ADS_outputs/        (test outputs)
    -example_test/
      -error.log
      -email.log
      -syslog.log
      ...
```

Testing procedure

- Framework and all other script must be run as root.
- Run `./prepare.sh`
- Put config file to folder *tests*
- Folder *nfcapd_files* with nfcapd files must have structure `/nfcapd_files/year/month/day/`
- Run `./test_ADS tests/yourFolder/yourConfigFile.ini`

Configuration file

```
#As input need config file with following structure (lines starting with
    # are comments, () are just explanations in config file they shouldn'
    t be

#required section
[common_settings]
#required options
reset=""/data/all   (data for clearing data, all for reset2factory, for
    no reset use 0,"" or just blank space)
SNMP=0/1
syslog=0/1
custom_script=0/1
email=0/1
statistics=0/1

#optional options
config=path   (XML file for upload)
timewait=x    (time in hours, for email/custom script waiting (0 -
    critical, immediate, 1 - hight priority, 6 - medium priority, 12 -
    low priority))

#optional section
```

```
[event_X]   (can be more sections, X can be string or number, but has to
    be unique)
#required options
#count can contain operands <, >, <=, >=, =, !=
count=X
eventtype=type  (substring...for example SCANS)

#optional options
timestamp=time  (in form 2016-08-18 05:28:32, check exact date)
source=ip     (ip adress)
detail=detail   (in form of regular expresion)
targets=IP, IP...
```

notes for configuration file:

- all options in event_X are just substring except detail
- detail is in form regular expresion

Example

```
[common_settings]
#required options
reset         = data
SNMP          = 0
syslog        = 1
custom_script = 0
email         = 0
statistics    = 0

[event_1]
#required options
count       = >1
eventtype   = SCANS

timestamp   = 2016-09-08 08:52:00
detail      = Description:\'TCP Null\';SuccessfulCount:\'0\';
#targets    = 10.200.10.249, 10.20.22.4

[event_2]
#required options
count     = >=2
eventtype = DIVCOM
```

Return values

- 255 when error occures (bad configuration, not available port for external syslog..)
- 0 when comparing was OK
- 1 when modules output doesn't match with ini.file expresions
- more detailed errorlog.log is created