VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ROZPOZNÁVANÍ APLIKACÍ V SÍŤOVÉM PROVOZU

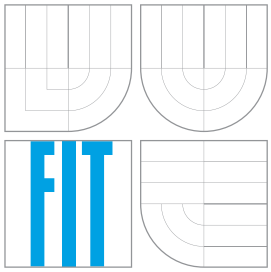
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN ŠTOURAČ

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ROZPOZNÁVANÍ APLIKACÍ V SÍŤOVÉM PROVOZU

NETWORK-BASED APPLICATION RECOGNITION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN ŠTOURAČ

VEDOUCÍ PRÁCE
SUPERVISOR

Mgr. KAMIL MALINKA, Ph.D.

BRNO 2014

Abstrakt

Tato práce seznámí čtenáře s různými druhy metod, které jsou v současné době využívány k detekci aplikací, komunikujících skrze počítačovou síť. Další část se zabývá výběrem vhodné metody detekce a implementace prototypu detekčního algoritmu pro zakomponování do existujícího produktu, včetně otestování jeho úspěšnosti detekce. Vybraný detekční algoritmus se opírá o využití statistických dat získaných ze síťových toků v komunikaci. Výsledné řešení není závislé na tom, zda-li komunikace probíhá šifrovaně, či nikoliv. Dále práce obsahuje různé varianty možností zakomponování funkčnosti detekce síťových aplikací do současného produktu Kernun UTM firmy Trusted Network Solutions a.s., s využitím nastudovaného detekčního algoritmu. Nejvhodnější návrh je vybrán a popsán podrobněji. V závěru je nastínění dalšího směru vývoje navázáním na současný stav a návrhy pro další možná zlepšení.

Abstract

This thesis introduces readers various methods that are currently used for detection of network-based applications. Further part deals with selection of appropriate detection method and implementation of proof-of-concept script, including testing its reliability and accuracy. Chosen detection algorithm is based on statistics data from network flows of tested network communication. Due to its final solution does not depend on whether communication is encrypted or not. Next part contains several possible variants of how to integrate proposed solution in the current architecture of the existing product Kernun UTM — which is firewall produced by Trusted Network Solutions a.s. company. Most suitable variant is chosen and described furthermore in more details. Finally there is also mentioned plan for further developement and possible ways how to improve final solution.

Klíčová slova

Internet, TCP, UDP, IP, Firewall, detekce aplikací, klasifikace provozu, HTTP, HTTPS, Facebook, Skype, K-Means, K-Nearest Neighbour, počítačová síť, KERNUN, TNS, UTM.

Keywords

Internet, TCP, UDP, IP, Firewall, application detection, traffic clasification, HTTP, HTTPS, Facebook, Skype, K-Means, K-Nearest Neighbour, computer network, KERNUN, TNS, UTM.

Citace

Jan Štourač: Rozpoznávaní aplikací v síťovém provozu, diplomová práce, Brno, FIT VUT v Brně, 2014

Rozpoznávaní aplikací v síťovém provozu

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana Mgr. Kamila Malinky, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Štourač
27. května 2014

Poděkování

Za odborné vedení, cenné rady a náměty při psaní této práce bych chtěl poděkovat Mgr. Kamilu Malinkovi, PhD. Dále bych zde rád poděkoval především mé rodině a blízkým za trpělivost a nikdy nekončící podporu jak při psaní této práce, tak v celém průběhu mého dosavadního studia. Můj dík patří i kolegům z TNS, za jejich vstřícnost při vypracovávání této práce.

Obsah

Kapitola 1

Úvod

Obsahem této práce je nastudovat současné možnosti klasifikace síťového provozu a detekce síťových aplikací s ohledem na aktuální trendy, které se této problematiky týkají. Dále seznámit se se základními principy těchto technik a zjistit, jaké jsou jejich výhody a nevýhody, případně zda-li budou využitelné i v dlouhodobějším horizontu. Zároveň jsou uvedeny některé nástroje a také komerční řešení, které danou problematiku pomáhají řešit nebo ji přímo implementují.

Dalším krokem je výběr zájmových tříd aplikací, které budeme chtít umět rozpoznávat a na kterých bude výsledná implementace také testována. S ohledem na vybrané třídy aplikací a také s přihlédnutím k očekávaným výsledným vlastnostem implementace, týkající se jejího konečného nasazení, je vybrána vhodná metoda detekce a konkrétní algoritmus.

Vybraný způsob detekce je následně implementován v podobě prototypu, který má za úkol ověřit, výsledky vzorového řešení a ověřit, zda-li bude v praxi vůbec použitelný. Zároveň se jedná o vyzkoušení implementace před samotnou integrací do cílového produktu. Pro možnosti porovnání jsou implementovány dvě varianty — zjednodušená verze a cílová verze. V rámci cílové verze byly implementovány dvě metody hodnotící funkce. Výsledné implementace jsou otestovány nad předem zachyceným reálným síťovým provozem.

Další kapitola se zabývá diskuzí nad několika možnými variantami, jak začlenit funkčnost detekce aplikací do současného firemního produktu Kernun UTM. U jednotlivých variant jsou brány v potaz jejich výhody a nevýhody, přičemž je vybrána jedna varianta jako nejvhodnější. Tato je rozepsána podrobněji a stala se základem pro počátek implementace výsledného řešení do cílového produktu Kernun UTM.

Poslední část se zabývá zhodnocením úspěšnosti implementovaného prototypu a dosažených cílů. Dále je diskutována možnost rozšíření funkčnosti detekce ještě o blokování vybraných tříd aplikací rovněž do navržené architektury. V samotném závěru je pak diskutován možný budoucí vývoj a návrhy na zlepšení řešení pro dosažení ještě lepších výsledků.

Kapitola 2

Co jsou to síťové aplikace a proč je rozpoznávat

Komunikace a možnost komunikovat byla odjakživa důležitou součástí lidských komunit a jejich rozvoje. S tím, jak se společnost vyvíjela a utvářela stále větší a větší celky, tato potřeba dále rostla. Dalo by se říci, že dnešní doba více než kdy jindy přímo stojí a padá s možností mezi sebou rychle a efektivně vzájemně komunikovat a vyměňovat si informace. Ať už za účelem práce, zábavy, výměny zkušeností, vzdělání, vědy a podobně. K masivnímu rozvoji tohoto trendu přispěl bouřlivý vývoj komunikačních technologií v průběhu minulého století. Obzvláště s rozvojem informačních technologií začala vznikat celá řada komunikačních kanálů, které jsou dnes dostupné prakticky komukoliv na světě a které dříve byly naprosto nemyslitelné. Mezi nejzásadnější prostředky se řadí celosvětová síť Internet. Díky tomuto nástroji jsme dnes schopní komunikovat v reálném čase s osobou na druhé straně planety nebo dokonce i mimo ni, pokud se nachází na místě s konektivitou a má adekvátní přijímač. Internet sám o sobě se skládá z nepřeberného množství menších sítí, které jsou vzájemně propojeny mezi sebou a vytváří tak celosvětovou síť, poskytující nejrůznější služby, které denně velká většina z nás využívá. Pro to, aby všechny součásti správně fungovaly, je nutná určitá správa jednotlivých sítí a zařízení v ní zapojených. S tím souvisí potřebná kontrola a zabezpečení jak těchto sítí, tak i koncových zařízení samotných, proti neoprávněným přístupům, manipulacím s daty, konfiguracím zařízení i sítě samotné nebo neoprávněným zneužíváním.

2.1 Síťové aplikace a síťový provoz

Jak již bylo zmíněno, v současné době hraje významnou roli v komunikaci celosvětová síť Internet. V rámci této sítě komunikují paralelně miliony lidí po celém světě. Uskutečňují spolu telefonní hovory, videohovory, chatují a podobně. Ke komunikaci jim slouží jakýkoliv počítač, který obsahuje nutný hardware, má konektivitu k internetu a odpovídající software. Právě tento software, který využívá připojení k síťové infrastruktuře nazýváme síťovými aplikacemi. Navíc se stálým postupným přesouváním jednotlivých služeb směrem od koncových stanic do internetového cloudu[1] se dokonce dají považovat za síťové aplikace

[1] Cloud (Cloud Computing) — je metoda poskytování služeb a počítačových programů, fyzicky uložených na serverech v Internetu, nikoliv na PC příslušného uživatele. Ten k nim přistupuje přes síť pomocí příslušného klienta, či webového prohlížeče. Tato oblast se stává velice oblíbenou, jelikož takto lze přistupovat ke svým datům odkudkoliv, kde je k dispozici připojení k Internetu, bez nutnosti řešit synchronizaci mezi

i různé webové stránky, jako jsou Facebook, Gmail, Google dokumenty, Twitter a mnoho dalších. Web, který je složen z takovýchto aplikací, se označuje jako Web 2.0 [1]. To, co tyto aplikace odlišuje od jiných webových stránek, je hlavně integrace více funkčních prvků, které z těchto stránek vytváří víceúčelové nástroje, jež tak těmto stránkám dodávají punc jakési aplikace. Facebook například nabízí možnost přímé komunikace svých uživatelů, zobrazuje aktuální dění na síti, dovoluje přehrávat videa, hudbu a podobně. Google dokumenty pak v sobě integrují textový editor, tabulkový procesor a další funkce, které jsme donedávna byli zvyklí vídávat pouze v desktopových aplikacích. Typicky se jedná o aplikace typu klient—server, kdy server obsluhuje většinu logických částí a klient pomocí webového prohlížeče zprostředkovává výsledky uživateli. Z pohledu klasifikace síťového provozu se však stále jedná o webové aplikace, které při komunikaci vykazují podobné vlastnosti, jako klasický webový provoz.

2.2 Kontrola síťového provozu

Důvodů, proč pro správce sítě může být zajímavé mít možnost rozpoznat aplikace, které v jeho síti komunikují, se dá vymyslet poměrně mnoho a lze pro ně najít využití v různých situacích. Příkladem může být problém s kapacitou síťové infrastruktury, kdy již kapacita sítě nestačí potřebám jejích uživatelů. Takovou situaci lze řešit jednoduše posílením prvků v síti a navýšením tak celkové kapacitu, kterou síť poskytuje. To ale samozřejmě bez předchozí znalosti provozu v síti a znalosti způsobů využívání sítě (například jaké aplikace v příslušné části sítě spolu komuikují a množství záteže, jež vyvolávají) může být neefektivní a třeba i neúčinné řešení. Taková data totiž mohou posloužit jako odrazový bod k zoptimalizování využití počítačové sítě, nastavení lepších hodnot Quality of service (QoS), či rozhodnutí o omezení nebo úplném odstavení určitých síťových aplikací. V konečném důsledku mohou oddálit nutnost povýšení prvků v síti a ušetřit tak náklady. Dalším příkladem může být omezení používání některých aplikací v síti z důvodů vnitřní bezpečnostní politiky. Například zamezení využívání cloudových synchronizačních služeb pro zabránění úniku citlivých dat mimo síť a podobně. Především ale tyto informace mohou dobře posloužit k tomu, aby administrátor měl přehled, co se v síti děje, jaký druh provozu kde běží a jestli to odpovídá očekávání. V průběhu času navíc může tato data srovnávat, pozorovat změny a ty následně analyzovat dále. Takovéto analýzy mohou posloužit i jako základ pro odhalení nebezpečného provozu. Mnohdy platí, že v případě napadení sítě nějakým nebezpečeným softwarem tento začne generovat novou neočekávanou komunikaci a spojovat se na různé body jak v interní, tak i externí síti. Tato změna chování v síti může posloužit jako první signál, že něco není v pořádku.

Příkladů pro využití by se našlo jistě mnohem více a mohou být různě specifické pro konkrétní síťovou infrastrukturu a společnost. Tím jak dochází k postupnému přesunu práce z jednotlivých koncových stanic do internetového cloudu, a tedy i k navýšení hustoty provozu v síti, je jednoduše nutné mít přehled, co se v počítačové síti organizace odehrává a mít možnost tyto děje včas identifikovat a případně ovlivňovat. Samozřejmostí je také to, že případné sledování a rozpoznávání aplikací nesmí mít žádný vliv na fungování sítě, alespoň do té doby, dokud si to nepřejeme. Je nutné používat neinvazivní techniky, aby nebylo narušeno pohodlí uživatelů ani kapacita sítě.

Pravdou je, že snaha o rozpoznávání aplikací, jejich povolování, omezování a zakazování v příslušné počítačové síti probíhalo vždy, historicky převážně z bezpečnostních důvodů.

různými zařízeními.

Každý správce si je vědom aplikačních portů, na kterých různé aplikace standardně poslouchají a komunikují. Podle tohoto portu se dala poměrně jednoznačně identifikovat konkrétní aplikace nebo služba. V posledních letech však spousta faktorů omezila přesnost takovéhoto způsobu identifikace. Hlavní příčinou je rozšíření používání firewallů a praktiky povolování komunikace jen přes některé porty (například HTTP: TCP 80, SMTP: TCP 25, SMTP: UDP 53 a další) kvůli vyššímu zabezpečení sítě, čímž se tyto porty staly více dostupné než jiné. To má za příčinu jejich velkého nadužívání ostatními aplikacemi, ať už přímým použitím daného portu nebo pomocí tunelování v odpovídajícím protokolu. Touto technikou se příslušné aplikace snaží dosáhnout větší šance na konektivitu s vnější sítí. Jiné aplikace pak používají dynamické porty, čímž ulehčují aplikační vrstvě od demultiplexování mezi jednotlivými instancemi. Příkladem mohou být různé streamovací protokoly jako SIP, H.323 nebo hry pro více hráčů a podobně. Některé aplikace nakonec používají náhodné porty přímo za cílem jejich nesnadné identifikace, jako například různí P2P klienti.[2] Dá se očekávat, že tento trend se bude do budoucna stále více rozšiřovat a stále více správců sítí bude narážet na problémy tímto trendem nastolené. Přestože dosavadní způsob práce s porty v sítích zůstane pravděpodobně ještě dlouho zachován a podporován, potřebujeme zároveň další možnosti, jak obsah uvnitř sítě monitorovat, analyzovat, klasifikovat a detekovat různé aplikace. Na scénu tak přicházejí metody, které se při identifikaci jednotlivých služeb neomezují pouze na aplikační port, ale analyzují datový tok samotný zkoumáním příslušných paketů a přenášených dat. Takovému přístupu se říká Deep Packet Inspection (DPI) [3, 4]. Výhodou tohoto přístupu je získání spousty dalších informací, relevantních ke klasifikaci síťového provozu a detekci aplikací. Velkým problémem je ale stále rostoucí popularita používání šifrovaného spojení. V takových případech je tato metoda prakticky nepoužitelná, pokud nejsme schopni toto šifrování nějakým způsobem obejít. Proto jsou rovněž rozvíjeny metody, které provádějí statistickou analýzu metadat o přenášených paketech a tocích v síti, na základě které jsou pak nějakým způsobem s určitou úspěšností schopné provoz klasifikovat a případně i detekovat různé komunikující síťové aplikace.

Kapitola 3

Rozpoznávání aplikací v Internetu

V předchozí kapitole byly nastíněny důvody, jež nás motivují k rozpoznávání síťových aplikací a také byl naznačen měnící se trend v podobně snižující se rozlišitelnosti těchto aplikací pomocí aplikačního portu. Vznikají tedy snahy o vytvoření jiných technik a postupů, které tuto rozlišitelnost opět vrátí na původní úroveň, případně ji ještě vylepší. V této kapitole jsou popsány některé dosavadní techniky a v současnosti používané metody, které lze k této činnosti využít.

Samotné způsoby, jakými detekovat jednotlivé síťové aplikace jsou z velké části závislé na samotné síťové technologii a infrastruktuře, která je ke komunikaci použita. Tato práce se zaměřuje na aplikace fungující v síti Internet, kde je jako referenční model používán model ISO/OSI [5]. Veškeré možné způsoby detekce tudíž musí reflektovat způsob implementace tohoto modelu a technologií nad ním postavených.

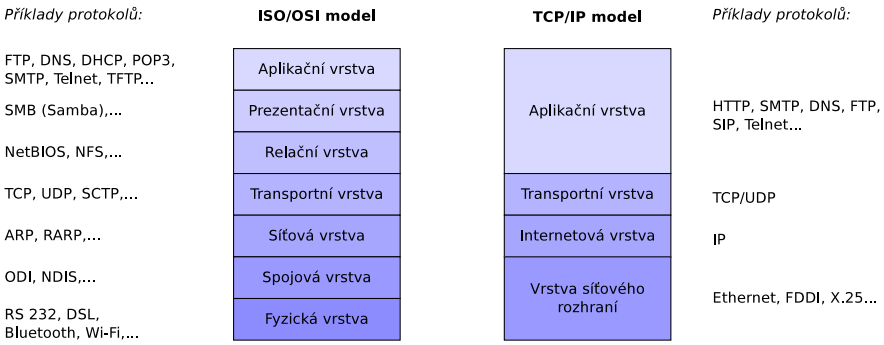
3.1 Referenční model ISO/OSI

Krátce se pozastavme nad stručným shrnutím základních technologií a protokolů, používaných v infrastruktuře sítě Internet. Popsání většího množství těchto technologií není zdaleka v ambicích této práce. Technologie a protokoly, které budou nezbytné pro bezprostřední pochopení obsahu této práce budou popsány v příslušných sekcích. Zároveň se jedná o ty nejzákladnější prvky ISO/OSI modelu.

Pokud chceme komunikovat mezi počítači nebo obecně mezi nějakými dvěma zařízeními, je nutné je propojit tak, aby byly schopné předávat si vzájemně data. Komunikace mezi počítači, potažmo mezi jednotlivými aplikacemi, které na nich běží, není sama o sobě triviální záležitost. Je nutné zajistit uložení dat do vhodného formátu a přidat takové informace, aby data mohla bezpečně doputovat k cíli, kde si je cílová stanice převezme, správně interpretuje a na základě přidaných informací rozpozná, zda nedošlo při přenosu k narušení integrity dat a následně určí, jaké konkrétní aplikaci byla data adresována. Proto vzniknul model ISO/OSI [5]. Tento model byl vypracován organizací ISO jako jako snaha o standardizaci počítačových sítí nazvané OSI (Open Systems Interconnection). V roce 1984 byl model přijat jako mezinárodní norma ISO 7498 a jeho hlavní úlohou je poskytnout základnu pro vypracování norem pro účely propojování systémů. Norma nespecifikuje samotnou implementaci ani realizaci systémů, ale uvádí všeobecné principy sedmivrstvé síťové architektury. Popisuje jednotlivé vrstvy, jejich funkce a služby. Nejsou zde zařazeny žádné protokoly, které by vyžadovaly zbytečně mnoho detailů. Především kvůli své složitosti se ale model ISO/OSI v praxi příliš neujal. Používá se tedy především jako referenční model. Naopak v rámci dnes

nejrozšířenější celosvětové počítačové sítě Internetu, se rozšířil model TCP/IP, někdy také označovaný jako Internetový model. Na obrázku lze vidět porovnání vrstev obou modelů s příklady používaných protokolů na úrovni příslušné vrstvy 3.1.

Model TCP/IP se nakonec ujal téměř na všech úrovních komunikace v rámci Internetu a v praxi se více než osvědčil. Technologie nad ním postavené se používají dnes a denně a dovolují flexibilní komunikaci mezi jednotlivými síťovými i koncovými uzly. Vzniklo tak velké množství komunikačních protokolů, které se striktně, či někdy poněkud neostře, řadí do příslušných vrstvev komunikace v tomto modelu.



Obrázek 3.1: Srovnání modelů ISO/OSI a TCP/IP na jednotlivých úrovních.

3.2 Detekce síťových aplikací

Každá metoda pro detekci síťových aplikací musí vycházet z vlastností použitých technologií, nad kterými jsou příslušné aplikace postaveny a jejich chování. To znamená, že máme k dispozici dostupné prostředky na různých úrovních TCP/IP modelu a s tím i jejich vlastnosti. Jelikož různé aplikace pracují na různých úrovních tohoto modelu, liší se i dostupné informace, jež lze přímo získat, aplikaci od aplikace. Dalším zdrojem k rozpoznávání je také různé chování aplikace na síti. To znamená různé způsoby, jakými se spojuje se serverem, případně dalšími klienty, jak moc a jakým směrem využívá datovou linku, množství dat, která přenáší, frekvence a velikost přeposílaných dat, pravidelné posílání určitého datového obsahu v komunikaci a podobně. Různé aplikace také mohou, a používají, stejné komunikační protokoly, případně mohou využívat i stejné knihovny pro síťovou komunikaci, které následně zapříčiňují podobné vzory chování. To může následně zkomplikovat jejich správné rozpoznání detekčním algoritmem a snížit tím úspěšnost klasifikace a detekce aplikací.

V této části se sluší zmínit rozdíl mezi pojmy klasifikace a detekce, jak je vnímán v této práci. I když může být rozdíl mezi nimi někdy velmi neostrý, klasifikací je obecně myšleno rozdělování síťové komunikace do několika různých skupin, které reprezentují nějakou společnou třídu síťových aplikací — typicky se jedná o rozdělování podle druhu síťového provozu nebo použitého aplikačního protokolu. Detekcí je pak myšlena detekce konkrétní síťové aplikace. V následujícím textu budou různě používány tyto dva pojmy, přičemž u převážné většiny jmenovaných algoritmů se mluví o klasifikaci, jelikož většina článků se zabývala právě klasifikací síťového provozu do různých skupin. To však neznamená, že není možné příslušný algoritmus využít pro detekci konkrétní aplikace vhodným naučením na testovacích datech.

Základní přehled aktuálně dostupných způsobů klasifikace síťového provozu a detekce síťových aplikací, jejich rozdělení a trendy jakými se pravděpodobně bude situace vyvíjet, provedu v následujících odstavcích této kapitoly. Tyto informace jsem čerpal z dostupné literatury, vědeckých publikací, dokumentací nekomerčních a OSS[1] řešení, i dostupných dokumentací a zdrojů ke komerčním řešením. Pěkný souhrn na toto téma také obsahuje článek A survey of Internet Traffic Identification [6]. Rovněž stručně popíši některé zajímavé vědecké články na toto téma.

3.2.1 Detekce podle aplikačního portu

Jedná se o původní metodu, kdy síťovou aplikaci identifikuje její aplikační port a použitý protokol. Na základě hodnot této dvojice se pak provádí povolování či zakazování komunikace aplikace s vnějším světem. Aplikační port je 16-ti bitové číslo, nabývající hodnot od 0 do 65535. Tento interval je rozdělen na tři části: 0 – 1023 jsou takzvané známé porty, 1024 – 49151 jsou porty registrované a 49152 – 65535 jsou porty dynamické nebo privátní.

První interval známých portů (0 – 1023) jsou porty rezervované často používaným aplikacím a protokolům, jako například HTTP [7], SSH [8], Telnet [9], SMTP [10], POP3 [11] a podobně. Dvojice port a aplikace je definována organizací IANA (Internet Assigned Numbers Authority [12]). Téměř všechna tato čísla jsou přiřazena. Na většině systémech je pro komunikaci na těchto portech vyžadováno oprávnění na úrovni systémového administrátora. Nemůže je tak použít neprivilegovaný proces pro svoji potřebu.

Druhý interval (1024 – 49151) jsou porty registrované rovněž u organizace IANA. Na rozdíl od předchozího intervalu, kde jsou porty již určené, lze v tomto intervalu registrovat vlastní protokol/aplikaci, která bude příslušný port využívat. Patří sem například OpenVPN, NFS, RADIUS, MySQL a další. Na rozdíl od předchozích mohou na většině systémů pomocí těchto portů komunikovat i aplikace s nižšími právy než s právy systémového administrátora.

A poslední třetí interval (49152 – 65535) jsou porty dynamické nebo pro privátní použití. Tyto porty nemohou být registrované u organizace IANA a jsou určeny pro dočasná krátká spojení, komunikaci privátních aplikací a podobně.

Všechny výše zmíněné porty se identifikují také na základě použitého protokolu na úrovni transportní vrstvy ISO/OSI. Tím je myšleno to, že různé protokoly mohou mít registrovaný, případně využívat, stejné číslo portu pro různé aplikace. Markantní množství komunikace v síti Internet je dnes na úrovni transportní vrstvy prováděno pomocí TCP nebo UDP protokolu. Právě alespoň pro tyto dva protokoly mají typicky nejpoužívanější síťové aplikace rezervovány stejné číslo portu — čili například komunikace HTTP probíhá standardně na portu 80 jak pro TCP, tak i pro UDP protokol.

Na základě povolování a zakazování aplikačních portů fungují mimo jiné paketové filtry a stavové firewally. Tato zařízení jsou schopna detekovat různé síťové aplikace pomocí použitého čísla portu a protokolu na úrovni transportní vrstvy ISO/OSI. Na základě této informace a zjištění, zda-li je příslušný port otevřen, případně přes něj probíhá komunikace tak jsou schopné zjistit, jaká aplikace v síti komunikuje a uzavřením daného portu pak příslušnou aplikaci odstavit z provozu. Příkladem paketového firewallu je například pf [13] z operačního systému OpenBSD a stavového paketového firewallu například ipfw [14] z operačního systému FreeBSD.

[1] Open Source Software — jedná se o software, jehož zdrojový kód je volně dostupný k nahlédnutí, úpravám a dalšímu použití podle licence, pod kterou je příslušný OSS vydán

V dnešní době je tato metoda, detekce a blokování síťových aplikací na základě aplikačního portu, již nedostatečná, protože její přesnost rapidně klesla (Karagiannis a spol. již v roce 2004 naměřili méně než 70% přesnost [15]). Stalo se tak z důvodů velké obliby omezování konkrétních portů na hraničních přístupových bodech do sítě, čímž se některé porty staly více dostupné než ostatní. Jedná se například o porty pro HTTP (80), HTTPS (443), SMTP (25) a podobně [6]. To mělo pak logický vliv na větší zájem o komunikaci přes takové porty, ať už snahou o jejich přímé použití či zabalení vlastní komunikace do příslušného protokolu. Dalším impulsem byla velká obliba NAT připojení. Tedy připojení, která se provádí pro úsporu IPv4 adres, případně pro skrytí interní podoby sítě před vnějším světem. Další nevýhodou pro tuto metodu jsou aplikace, které jsou schopné komunikovat na různých portech a dynamicky mezi nimi přepínat, jako například různé P2P aplikace, Skype nebo hry pro více hráčů a podobně. Je tedy zřejmé, že tímto směrem se nelze dále ubírat a je nutné najít lepší způsoby, jak provoz v síti klasifikovat.

3.2.2 Detekce podle obsahu přenášených dat

Zatímco předchozí způsob detekce sledoval pouze čísla použitého aplikačního portu a typ transportního protokolu, tento postup (takzvaný Deep Packet Inspection — DPI [3, 4]) se důkladněji zaměřuje v podstatě na všechna smysluplná data v komunikaci, která lze rozumně využít. Jedná se samozřejmě o informace z hlaviček jednotlivých protokolů, ale především na samotná data v paketech — takzvaný payload — jež jsou podstatou samotného přenosu. Jsou to data, která doputují až k samotné cílové aplikaci, jež je přečte a interpretuje odpovídajícím způsobem. Cílem této metody je daná data vzít, analyzovat a detekovat z nich příslušnou aplikaci. V tomto postupu se využívá hledání vzorů, které identifikují konkrétní aplikaci, rozpoznávání aplikačních protokolů nebo statistické informace o obsahu. Tato metoda přináší spoustu relevantních údajů pro klasifikaci a je tak možné markantně zvýšit přesnost detekce. Právě proto je tento způsob detekce velmi často používaný v praxi — například v aplikacích Snort [16], Bro [17], ve svých produktech jej mimo jiné využívají například firmy Check Point [18], Palo Alto Networks [19] a Procera Networks [20].

Bohužel ne vždy jsou data z payloadu dostupná. Důvodem může být šifrovaná komunikace, která se především v posledních letech značně nasazuje a je nutné počítat s jejím dalším rozšiřováním. V některých případech může být problém i s nepatrně vyšší náročností algoritmu, jelikož dochází k analýze samotných dat v přenášených paketech, což klade vyšší nároky na rychlost a výkonnost použité platformy, aniž by utrpěla propustnost zařízení. Dalším aspektem může být právní rovina. Jelikož dochází k tak důkladnému prohledávání síťového toku, může být v některých státech toto chování, hlavně v závislosti na dalším způsobu nakládání s těmito daty, považováno za narušování soukromí či omezování svobody jedince [21].

Samotné způsoby, jakými dochází ke tvorbě a použití vzorů pro rozlišování různých síťových aplikací je více. V jednodušším případě se může jednat jen o posloupnosti bajtů, které jsou očekávány v přenášených datech a které příslušnou aplikaci dostatečně dobře identifikují. Do této kategorie však také spadají metody, kdy je příslušný síťový tok průběžně sledován, přijímaná aplikační data z něj zpět sestavována a teprve na nich je prováděna detekce aplikace. V takovém případě může posloužit i rozlišování použitého aplikačního protokolu, což, pokud je takový protokol rozeznán, dává ještě větší možnost porozumět dané aplikaci a obsahu dat, která jsou přenášena. Tato metoda z velké části spoléhá na porozumění samotné aplikaci. Z toho plyne i nutnost znalosti o detekovaných aplikacích pravi-

delně aktualizovat, jelikož průběžně může docházet ke změnám chování, úpravám použitých aplikačních protokolů, snahám o ztížení možností detekce a podobně.

3.2.3 Detekce na základě informací o datových tocích

Obě předchozí metody detekce se zaměřovaly jen na informace dostupné přímo v přenášených datech — ať už aplikačních nebo v hlavičkách použitých protokolů. Nebraly v potaz možné informace, které jsou dostupné z chování síťové komunikace různých aplikací. Takový způsob se primárně nezabývá přenášenými daty, ale spíše informacemi o samotném přenosu. Komunikace síťových aplikací je založena na výměně síťových paketů. Takto spolu komunikují typicky dvě zařízení, které si příslušné pakety vyměňují. Sousledností takových paketů posílaných mezi dvojicí zařízení vzniká takzvaný síťový tok. K identifikaci síťového toku typicky slouží tato pětice informací: zdrojová a cílová IP adresa, zdrojový a cílový aplikační port a použitý protokol na úrovni transportní vrstvy. K této pětici se ještě často sleduje čas posledního přijatého paketu v takovém toku. Důvodem je zjišťování, zda-li je příslušný tok stále aktivní. Většinou je definován nějaký čas, po který, když není v příslušném síťovém toku přenesen žádný paket, je tento tok označen jako neaktivní. Právě na vlastnosti takových síťových toků se dívá tento způsob detekce.

Data sloužící k získání informací o chování konkrétního síťového toku, jsou v omezené míře brána z jednotlivých hlaviček použitých protokolů v paketech toku — typicky do úrovně transportní vrstvy ISO/OSI modelu — jako jsou: velikost přenášených dat, použitý protokol, a podobně. Hlavním zdrojem informací jsou však data o přenášených paketech jako takových – například velikosti jednotlivých paketů, časové rozdíly mezi příchozími pakety, počet paketů poslaných od klienta na server a naopak, délka samotného spojení a podobně. Jedná se tedy o zkoumání metadat samotného síťového toku, která jej identifikují v síti a komunikaci. Všechna tato data se pak dají různě využívat a stavět na nich statistické modely, podle kterých lze rozeznat určitý druh provozu. Výhodou tohoto způsobu je to, že nenahlíží do samotného obsahu paketu, který nemusí být vždy dostupný — například z důvodu šifrované komunikace, právních a jiných důvodů.

Někdy se při detekcích založených na této metodě záměrně a úplně vynechávají informace obsažené v hlavičkách paketů, se snahou spolehnout se pouze a jen na charakteristické chování jednotlivých toků a klasifikovat provoz pouze na základě statistik z tohoto chování získaných. Motivací k takovému přístupu je mnohem větší flexibilita, jelikož data z hlaviček (především IP adresy a aplikační porty) se často vztahují ke konrétní počítačové síti a v různých sítích mohou být jejich hodnoty různé a tím ovlivnit celkovou spolehlivost detekce. Podobná situace může nastat i při zásazích do struktury sítě. Vynecháním takto specifických informací tedy snižujeme riziko a nutnost opětovného učení se provozu na nové síti. Ovšem to stále nevylučuje nutnost získání dostatečně obsáhlého vzorku trénovacích dat, jelikož některé aspekty konkrétní sítě lze jen těžko odstraňovat — například kvalita a rychlost linky, zařízení, které mají různé velikosti MTU[2], čímž ovliňují velikosti přenášených paketů a podobně.

S nástupem rozšiřování používání šifrované komunikace se stále více snah vkládá do výzkumu způsobů detekce síťových aplikací, které nezávisí na obsahu payloadu paketů tak, jak bylo popsáno v předchozí sekci 3.2.2.

[2]Maximum transmission unit (MTU) — určuje maximální velikost dat v bytech pro jeden rámec konkrétního komunikačního protokolu nebo který je příslušné síťové zařízení schopné vyslat.

3.3 Existující řešení

Klasifikace síťového provozu je v praxi velmi zajímavé téma a implementaci této vlastnosti nabízí spousta komerčních firem. Existují i nekomerční a OSS řešení. U jednotlivých metod jsem již zmínil příklady nějakých nekomerčních či komerčních řešení, přičemž lze vidět, že většina z nich se dnes zabývá právě metodou detekce na základě obsahu přenášených dat v síťovém provozu a některé využívají i detekci na základě informací o datových tocích. Naopak lze vidět, že stále méně řešení se spoléhá na detekci pouze na základě aplikačních portů. Dále některá z těchto řešení stručně popíšu.

3.3.1 Komerční řešení

Metodu detekce na základě obsahu přenášených dat nebo-li DPI, využívají například řešení firem Check Point, Palo Alto Networks či Procera Networks. Tyto firmy spravují velké databáze s informacemi o síťových aplikacích, které se snaží svými produkty detekovat, případně blokovat. Tyto databáze musí být pravidelně udržovány s tím, jak se případně mění vlastnosti detekovaných aplikací, které mohou změnit detekované vzory tak, že pomocí vzorku ze zastaralé databáze není možné příslušnou aplikaci již detekovat. Jedná se o komerční řešení, tedy podrobnější technické informace nejsou dostupné.

O využívání metody detekce na základě informací o datových tocích v praxi jsem narazil pouze v informacích ke komerčnímu řešení od společnosti Palo Alto Networks — AppID. Tato jejich hybridní metoda detekce aplikací využívá právě jak informace o datových tocích, tak i DPI. Síťový provoz je nejprve tříděn na základě IP adres a portů, posléze je využita metoda DPI pomocí sad vzorků pro jednotlivé aplikace z rozsáhlé databáze. V případě šifrovaného provozu a pokud je uplatňována politika dešifrování provozu na zařízení, uplatní se toto i na šifrovaný (v tuto chvíli nešifrovaný) provoz. Pro síťový tok, pro který nebyla detekována žádná aplikace se využijí heuristiky a právě behaviorální analýzy, tedy detekce na základě informací o datových tocích, na jejichž základě je komunikace zařazena do nějaké z definovaných tříd provozu. Informace o takovémto novém a nedefinovaném provozu pak mohou být poslány společnosti pro bližší analýzu.

3.3.2 Snort

Snort je OSS síťové IDS, případně IPS zařízení, které je poměrně dobře rozšiřitelné pomocí vlastních plug-inů a pravidel, jež v něm lze nakonfigurovat. Funguje na základě sledování toku v síti a podle nahraných pravidel rozpoznává, zda-li se jedná o nebezpečnou komunikaci a generuje odpovídající hlášení o nastalých problémech. Dá se tedy dobře využít k detekci pomocí DPI nad procházejícím provozem napsáním vlastních specifických pravidel, reagujících na výskyt konkrétních vzorků dat v komunikaci. Pro jeho správnou funkčnost je nutné jej korektně nakonfigurovat v závislosti na podobě sítě, kde je nasazován a druhu provozu, který v ní probíhá. Také je nutné pravidelně aktualizovat sadu pravidel, na základě nichž se rozhoduje. Mimo psaní vlastních pravidel je možné pro Snort psát vlastní moduly s rozšířenou funkčností s využitím Data AcQuisition knihovny (DAQ) [22]. Tato knihovna nahrazuje přímé volání funkcí při zachycení paketu pomocí knihoven jako například PCAP, abstraktní vrstvou, která podstatně zjednodušuje implementaci vlastní reakce na zachycený paket. Výhodou takto napsaných modulů je také to, že mezi nimi může být přepínáno za běhu bez nutnosti nového překladu celé aplikace.

Nedávnou novinkou, kterou avizovalo Cisco, je přidání podpory takzvaného OpenAppID [23, 24] do aplikace Snort. Jedná se o specifický jazyk a modul, který do Snortu

přidává možnost jednoduchým způsobem implementovat detekci aplikací pomocí předpřipravených definicí. OpenAppID je způsob popisu webových aplikací. Myšlenkou je nabízet tyto definice volně a nechat komunitu, aby sama vytvářela definice nové a aktualizovala stávající. Tedy to, co dnes jednotlivé komerční firmy spravují jako proprietární intelektuální vlastnictví, by mělo být díky práci komunity dostupné zdarma pro každého, který implementuje použití aplikace Snort ve své síti. Dá se předpokládat, že pokud toto řešení komunita přijme, velice rychle vznikne rozsáhlá sada takových pravidel, čímž může nakonec překonat i databáze některých komerčních řešení.

3.3.3 Bro

Aplikace Bro je OSS analyzátor síťového provozu s velmi rozsáhlými schopnostmi. Dokáže analyzovat živý nebo předem nahraný síťový provoz a generovat události. Pokud se stane určitá akce, například při zachycení nového HTTP requestu, TCP spojení nebo i ukončení některého z procesů samotné Bro aplikace, je vygenerována událost. U události se nerozlišuje, jestli se jedná o špatnou nebo neškodnou. Událost slouží pouze jako vstup pro další část aplikace — sadu skriptů — která na základě vygenerovaných událostí vykonává nastavenou politiku. Tyto skripty jsou psané ve vlastním skriptovacím jazyce Bro.

Pro rozlišení síťových protokolů využívá Bro jak detekci na základě známých portů, tak i na základě dynamických portů s využitím připravených vzorů, ale i základní jednoduché behaviorální analýzy.

Obecně lze říci, že převážná většina řešení v případě detekce aplikací využívá právě DPI. Avšak vzhledem k tomu, že cílem této práce je pokusit se detekovat aplikace bez ohledu na to, jestli je provoz šifrovaný či ne, není možné přímo, bez jistých úprav, žádnou z výše uvedených aplikací využít.

3.4 Související výzkumné publikace

Jak jsem zmínil dříve — klasifikace síťového provozu a detekce aplikací v něm je pro praxi velice zajímavé téma — tedy se věnuje této problematice poměrně mnoho vědeckých článků, v nichž se objevují různé přístupy s využitím různých algoritmů. V základu se jedná vždy o využití některé z předchozích metod, případně nějaké z jejich kombinací s využitím různých algoritmů tak, aby výsledek odpovídal požadovaným vlastnostem a parametrům výsledného systému. Proto rozdělení detekce v předchozích odstavcích lze brát spíše jako striktní přehled možností, které jsou v praxi různě kombinovány. V následujících odstavcích bych rád shrnul některé z prací a přístupů, které jsem v rámci přípravy na toto téma nastudoval.

Mnoho současných prací si uvědomuje rozšíření šifrovaného provozu a také důsledky, jaké z toho plynou pro detekci aplikací a klasifikaci provozu v síťi. Alshammari a Zincir-Heywood [25] se v jejich práci zaměřují na klasifikaci šifrovaného provozu s využitím strojového učení. Zároveň se pro zjednodušení omezují pouze na identifikaci toků SSH [8] a Skype aplikací. Jejich cílem bylo dosažení co nejrobustnějšího modelu, který není nijak závislý na konkrétní síťi. Pro získání znalostní báze — modelu — využili pět učících algoritmů: AdaBoost, Support Vector Machine, Naïve Bayesian, RIPPER a C4.5. Z jejich výsledků je patrné, že nejlepších hodnot dosáhla implementace pomocí učícího algoritmu C4.5. Nejhorší výsledek této implementace dosáhl detekční úspěšnosti pro SSH provoz 83,7 % a false

positive 1,5 %, kdy učení probíhalo na jedné síti, ale testování na jiné. Tím se podařilo odstínit případné odchylky, které mohly být závislé na původní síti. Nejlepší výsledek pak byla detekční úspěšnost 97 % a false positive 0,8 %. Při detekci provozu toků aplikace Skype dosáhli hodnot detekce 98,4 % a false positive 7,8 %. Úroveň detekce, na jakou dosáhli, je poměrně slibná, avšak omezili se na detekci pouze dvou aplikací, takže je otázka, zda by se stejná úroveň udržela i s přidáním detekce více aplikací. Rovněž testování neprobíhalo online za provozu, ale na nasbíraných vzorcích dat, tedy rychlost detekce nebyla úplnou prioritou výsledného řešení.

Další práce zabývající se klasifikací šifrovaného provozu od Bar-Yanai a kol. [26] se zaměřuje na klasifikaci šifrovaného i nešifrovaného provozu pomocí statistických metod nad síťovými toky, která by byla schopná pracovat v reálném čase. Využívají k tomu klasifikátor založený na hybridní kombinaci algoritmů K-Means [27] a K-Nearest Neighbors [28]. Spojením výhod obou přístupů dosáhli na řešení využitelné pro efektivní klasifikaci provozu v reálném čase, které vyzkoušeli na stroji Cisco SCE 2020, přičemž se ale nezmiňují o maximální použitelné rychlosti linky. Provoz třídili do devíti skupin: HTTP, SMTP, POP3, Skype, EDonkey, BT, Šifrovaný BT, RPT, ICQ. Přesnost jejich algoritmu u každé ze zmíněných skupin byla nejhůře 94 % a nejlépe téměř 100 %. Zároveň dosáhli podobných výsledků u nešifrovaného a šifrovaného BitTorrentu, což naznačuje, že jejich metoda je stejně dobře použitelná na šifrovaný i nešifrovaný provoz.

Caniny a kol. [29] navrhli řešení, využívající hardwarovou akceleraci nad NetFPGA [30], díky kterému dosáhli na skvělé rychlosti klasifikace. Řešení úspěšně otestovali na rychlosti 1 Gb/s s maximální rychlostí procházejících paketů o velikosti 173 Kp/s. Provoz klasifikují do 13-ti tříd a pro vytvoření znalostní báze použili rozhodovací algoritmus C4.5. Zajímavým řešením pro urychlení vyhledávání v tabulce toků bylo vytvoření dvouúrovňové cache. Nejprve se hledá tok pouze podle IP adresy, aplikačního portu a protokolu. Tedy podle trojice namísto pětice, která definuje tok. Tento přístup hájí pozorováním, že v síti v příslušném časovém okamžiku existuje pro takto definovanou trojici pouze jedna příslušná aplikace, která touto cestou komunikuje. Teprve pokud není v první úrovni cache nic nalezeno, tak se začne hledat v druhé úrovni, kde se již vyhledává pomocí standardní pětice, která identifikuje síťový tok.

Wang a kol. [31] dosáhli ve své práci vysoké úspěšnosti detekce mimo jiné i díky pomocí technik DPI. Jejich implementace automaticky se učícího klasifikátoru pracuje na základě kombinace statistických vlastností síťových toků a naučených vzorů, které hledají v jednotlivých tocích a jež reprezentují konkrétní třídu aplikací. Nejprve provedou agregaci toků do několika skupin pomocí algoritmu X-means [32]. Následně se v jednotlivých skupinách toků snaží nalézt společné vzory, na základě kterých by dále probíhala klasifikace nového síťového toku. Samotné vzory jsou v jejich práci definovány jako vektor posloupností bytů z datové části paketu. Tedy očekávají v toku nějaké množství různých sekvencí bytů, které jsou pro danou aplikaci v toku typické. Jimi dosažená přesnost klasifikace je více než 99,8 %. Výjimkou je pouze třída Edonkey, která dosáhla přesnosti 98,3 %. Navržené řešení je příkladem hybridního přístupu, kdy jsou využívány jak statistické informace o síťových tocích, tak samotná přenášená data v paketech. Nevýhoda tohoto postupu je v omezení rozumného použití na šifrovaný provoz. V takovém případě je totiž datový obsah paketů šifrovaný a nalézt vhodné vzory pro klasifikace je tak v tomto případě značně obtížnější. Naproti tomu ale v jejich výsledcích figuruje HTTPS jako samostatná třída, což naznačuje, že minimálně v nějaké omezené míře je i jejich metoda pro šifrovaný provoz použitelná.

3.5 Referenční síťové aplikace

Na začátku práce bylo naznačeno, co je myšleno pod pojmem síťová aplikace 2.1. Jedná se tedy o počítačovou aplikaci, která ke své činnosti využívá komunikaci skrze počítačovou síť. Takových aplikací existuje obrovské množství — což lze vidět například v katalozích síťových aplikací firem Palo Alto Networks [33] či Check Point [34] a neustále vznikají nové. Obecně můžeme některé aplikace sdružovat do společných tříd podle účelu, ke kterému byly vyvinuty. Příkladem takové třídy aplikací je webový provoz nebo-li HTTP/HTTPS provoz, SMTP nebo-li e-mailová komunikace a podobně. Konkrétní aplikací je pak například webový prohlížeč Internet Explorer, Google Chrome, respektive pro e-mailovou komunikaci Mozilla Thunderbird, atd. Avšak vzhledem k trendům v oblasti webové komunikace a využívání informačních technologií se stále více masově používaných aplikací přesouvá do cloudu a jsou nabízeny formou webové aplikace spouštěné ve webových prohlížečích. Tento vývoj způsobuje, že je stále více žádané mít možnost různé tyto webové aplikace od sebe rozpoznávat. Termínem webová aplikace je míněna taková aplikace, jejíž běh probíhá ve webovém prohlížeči a komunikuje se serverem. Na rozdíl od standardních webových prezentací/stránek obsahuje rozšířené funkce, které byly do nedávné doby vlastní pouze pro klasické desktopové aplikace. Příkladem takových webových aplikací je třeba Facebook, Google dokumenty, Gmail a další.

Cílem této práce není postihnout detekci celého takto rozsáhlého množství síťových aplikací. Snahou je vybrat vhodný mechanismus, který by byl schopný efektivně klasifikovat síťový provoz do níže uvedených kategorií a detekovat vybrané síťové aplikace, a to takovou technikou, která v ideálním případě půjde rozšířit i na další, především webové, aplikace.

Následující seznam vyjmenovává referenční třídy a síťové aplikace, které nás v této práci z pohledu klasifikace a detekce zajímají:

- HTTP [7] — webový provoz
- HTTPS [35] — šifrovaný webový provoz
- Facebook — webová aplikace, pracující nad HTTP/HTTPS
- Google dokumenty — webová aplikace, pracující nad HTTP/HTTPS
- BitTorent — peer-to-peer distribuční protokol pro distribuci souborů
- Skype — proprietární komunikační protokol pro komunikaci v reálném čase (videokonference, chat)
- SMTP [10] — protokol pro přenos emailových zpráv
- SSH [8] — secure shell
- DNS [36] — protokol pro překlad doménových jmen
- SNMP [37] — protokol pro sběr dat o zařízeních na síti a správu sítě

Z velké části se jedná o třídy síťového provozu, rozdělené podle aplikačního protokolu. Za poznámku však stojí právě aplikace Facebook a Google documenty. V těchto případech se totiž na rozdíl od ostatních výše uvedených aplikací nejedná o aplikační protokoly, ale jde o webové aplikace, fungující nad HTTP/HTTPS aplikačním protokolem. Jde tedy o jakési podtřídy právě tříd HTTP/HTTPS. Zvolená metoda detekce by měla umožnit úspěšné rozpoznání výše uvedených tříd a aplikací. Zároveň je cílem, aby bylo možné rozšířit toto rozpoznávání i na další aplikace, především webové, jako například Google mail a další.

Kapitola 4

Zvolené detekční techniky

V předchozí kapitole 3.4 jsem zmínil několik vědeckých prací, zabývajících se klasifikací síťového provozu. Většina z nich brala v potaz přítomnost šifrovaných přenosů, jelikož je využití šifrování dat dnes poměrně široce využívané a do budoucna se pravděpodobně bude jeho využití nadále zvyšovat. Proto se nelze spoléhat na podrobné hledání informací přímo v samotných datových částech paketů (DPI), jelikož k těmto datům díky šifrování není jednoduše přístup. Většina technik se tedy snaží klasifikovat síťový provoz na základě metadat o provozu samotném — snaží se počítat statistiky a vlastnosti provozu a na základě toho určit do jaké kategorie spadá, případně přímo o jakou aplikaci se jedná.

Výsledné řešení by se tedy mělo zaměřit právě na detekci způsobem, který zvládá šifrovaný i nešifrovaný provoz stejnou nebo alespoň podobnou měrou. Zároveň s ohledem na využití, je nutné, aby bylo schopné klasifikovat provoz pokud možno v reálném čase. Na základě těchto informací mě při výběru návrhu řešení, které zvolit pro implementaci proof-of-concept, ze zmíněných prací nejvíce zaujalo řešení Bar-Yanai a kol. [26]. V jejich práci se zaměřují na klasifikaci šifrovaného i nešifrovaného provozu právě pomocí statistických metod nad datovými toky provozu s přihlédnutím k tomu, aby výsledné řešení bylo schopné pracovat v reálném čase. Podle jejich závěrů jim jejich řešení pracuje stejně dobře jak nad šifrovaným, tak i nad nešifrovaným provozem a dosáhli velice slibných výsledků klasifikace ve všech třídách, do kterých klasifikovali 3.4 síťový provoz. Třídy, do kterých provoz klasifikovali, se částečně překrývají s tím, jak by měla klasifikovat výsledná implementace této práce. Rozdílem je však to, že jejich třídy provozu neobsahují žádné webové aplikace. U nich pravděpodobně budou hodnoty statistických ukazatelů komunikace velice podobné, jelikož se jedná o komunikaci nad stejnými protokoly. Zde se tedy dá očekávat, že bude mít řešení nedostatky, které bude nutné odstranit. Toto řešení blíže popíšu dále.

4.1 Klasifikace na základě informací o datových tocích

Jak jsem uvedl výše, jako nejslibnější návrh klasifikace provozu po nastudování existující literatury, z pohledu cílového využití mé práce, se jevil přístup popsaný v článku Realtime Classification for Encrypted Traffic [26] Bar-Yanai a kol. Jejich cílem bylo vytvořit klasifikátor, který by fungoval nezávisle na tom, zda-li jsou klasifikovaná data šifrovaná, či nikoliv. Bylo tedy nutné zaměřit se na statistické vlastnosti přenášených paketů a jimi tvořených síťových toků, a ignorovat samotná přenášená data. Dalším jejich cílem byla snaha o rychlou klasifikaci, která by byla schopna korektně pracovat i při klasifikaci provozu v reálném čase. Ve své práci zmiňují, že svoje řešení vyzkoušeli na stroji Cisco SCE 2020, kde nijak zá-

sadně běžící klasifikace nezatěžovala celkový systém. Bohužel se už ale nezmiňují o rychlosti použité datové linky a rychlostí přenášených dat.

Práce jimi navrženého algoritmu se rozděluje na dvě části — část trénovací a část klasifikační. V trénovací části se pomocí předpřipravených dat, u kterých známe náležitost ke správné aplikaci či třídě provozu, vytvoří vhodná reprezentace statistických dat. Tato databáze znalostí je pak využívána v druhé — klasifikčaní — části, kdy již dochází ke zpracovávání přijímaného provozu a jeho klasifikaci, případně detekci konkrétní síťové aplikace.

V této části následuje podrobnější popis klasifikačního algoritmu, který Bar-Yanai a kol. v rámci své práce navrhli.

4.1.1 Sledovaná statistická data

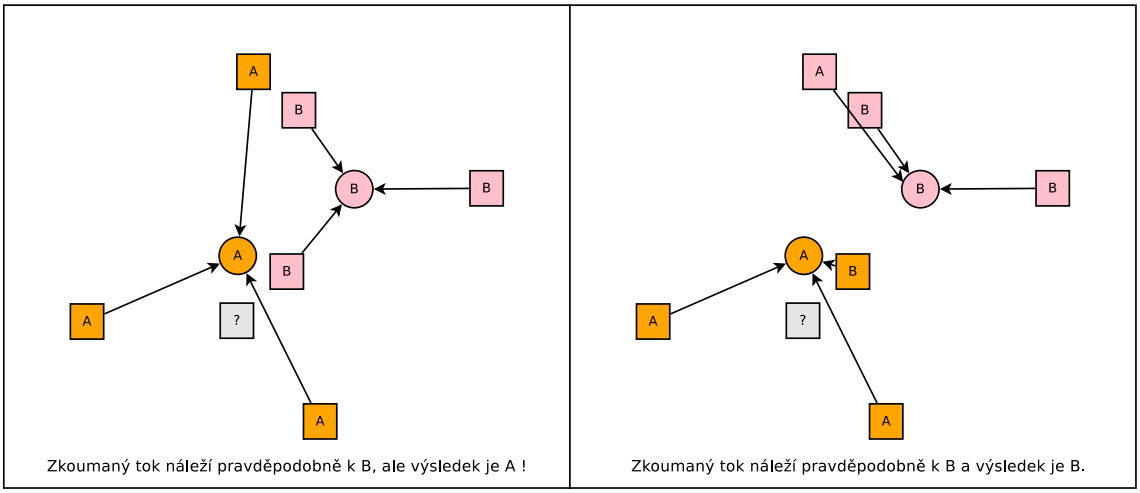
V každém síťovém toku mezi sebou komunikují dvě zařízení, kde jeden z nich se označuje jako server a druhý jako klient. To zařízení, které iniciovalo a začalo síťový tok — to je vyslalo první paket daného flow druhé straně — je označováno jako klient. Každý síťový tok, který v komunikaci probíhá je vnímán jako vektor sady parametrů $\overline{V}_x = \langle V_1, ..., V_d \rangle$, kde každý parametr V_i obsahuje nějakou statistickou hodnotu toku x, například průměrnou velikost paketů v toku. Tyto parametry jsou vybrány s ohledem na požadavek klasifikace v reálném čase, tedy jsou vybrány takové vlastnosti, které lze počítat relativně rychle a ideálně tak, že stačí každý příchozí paket procházet pouze jednou. Zároveň bylo nutné určit hranici, kdy se již dá předpokládat, že je k dispozici dostatečné množství dat pro to, aby bylo možné příslušný síťový tok klasifikovat. Důvodem je to, že některé toky mohou trvat velice dlouho s obrovským množstvím paketů. Tuto hodnotu Bar-Yanai a kol. ve své práci určili maximálním počtem paketů v toku na hodnotu 100, případně méně, pokud tok tolik paketů neobsahuje. Zde je kompletní seznam jimi sledovaných vlastností v každém sledovaném toku: počet paketů od klienta, počet paketů od serveru, celkový počet paketů, očekávaná velikost paketu klienta, očekávaná velikost paketu serveru, průměrná rychlost paketů od klienta, průměrná rychlost paketů od serveru, rozptyl velikostí paketů od klienta, rozptyl velikostí paketů od serveru, celkový počet bytů od klienta, celkový počet bytů od serveru, poměr downloadu a uploadu, průměrné množství bytů od klienta na dávku[1], průměrné množství bytů od serveru na dávku, průměrné množství paketů od klienta na dávku, průměrné množství paketů od serveru na dávku a typ transportního protokolu — TCP nebo UDP.

V předchozím odstavci byla popsána struktura vektoru každého toku, který vstupuje k použití do detekčního algoritmu. Nyní je třeba tyto hodnoty správně použít. Základem jejich metody je použití hybridního algoritmu, využívajícího geometrické klasifikační algoritmy K-Nearest Neighbors (K-NN) [28] a K-Means [27]. Oba z těchto algoritmů mají své výhody a nevýhody, které se jejich vzájemnou kombinací pokusili Bar-Yanai a kol. odstranit. Zatímco K-NN algoritmus je velmi spolehlivý co se týče klasifikace, jeho náročnost lineárně stoupá s velikostí trénovacích dat. To přináší problém, jelikož více trénovacích dat obecně zlepšuje kvalitu klasifikace, ale v kombinaci s tímto algoritmem by to současně znamenalo neudržitelné zvětšení výpočetní náročnosti. Naproti tomu algoritmus K-Means nedosahuje tak kvalitních výsledků při klasifikaci příchozího síťového toku, ale oproti KNN nepotřebuje tolik prostředků a je celkově méně náchylný na velikost trénovacích dat. Právě tyto vlastnosti přiměly autory k návrhu hybridního algoritmu, vzájemně kombinující zmíněné algoritmy způsobem, kdy jsou využity jejich přednosti a jejich nevýhody částečně potlačeny.

[1] Navazující části síťového toku, oddělené mezipaketovými mezerami o velikosti 1 vteřiny a více.

4.1.2 Trénovací část algoritmu

Trénovací část se dělí na dvě fáze. Pro úspěšné splnění trénovací fáze je třeba mít vzorky komunikace, kde jsou jednotlivé toky označeny správným druhem aplikace nebo třídou provozu. Bez těchto informací není možné provést trénovací fázi a algoritmus by neměl na čem stavět. První fáze proběhne tak, že na každou sadu toků jedné třídy/aplikace (například HTTP, SMTP,...) se provede vytvoření skupin toků pomocí algoritmu k–means. Tak vznikne sada skupin, kde každá skupina obsahuje jen toky, příslušející jedné třídě nebo aplikaci. Samozřejmě může platit to, že jedné aplikaci náleží více skupin avšak ne to, že v jedné skupině jsou toky různých aplikací nebo tříd. Tyto skupiny se však mohou vzájemně překrývat, což by v následujících fázích klasifikace mohlo vést ke klasifikačním chybám. Například, pokud by se těžiště dvou skupin, náležící každá jiné třídě nebo aplikaci, nacházela dostatečně blízko, tak jejich jednotlivé toky se mohou vzájemně překrývat. To znamená, že příslušná třída provozu nebo aplikace jsou si v podobě sledovaných statistických dat poměrně blízké. Pokud bychom okamžitě využili před chvílí vyrobené skupiny a použili ke klasifikaci skupinu s nejbližším těžištěm, může se stát, že provedeme klasifikaci chybně. Lépe tuto situaci ilustruje obrázek 4.1.



Obrázek 4.1: Diagram znázorňující přiřazení nového toku do nejbližší třídy. Vlevo je vidět, jak může dojít k chybě, pokud nejsou trénovací toky zařazeny k nejbližšímu těžišti.

Proto se vypočtou těžiště těchto jednotlivých skupin a sjednotí se do jedné množiny. Tím získáme množinu těžišť, které vycházejí ze skupin, vytvořených z diskrétních tříd a aplikací. Nad touto novou množinou těžišť provedeme přiřazení všech toků, které jsme v předchozí části použili, takovým způsobem, že příslušný tok přiřadíme nejbližšímu těžišti z množiny. Nyní ale již bez ohledu na to, o jakou třídu provozu nebo aplikaci se jedná. Po této fázi již tedy nemusí platit to, že v jedné skupině (myšleno množina toků, spadající k jednomu těžišti) mohou být pouze toky jedné aplikace/třídy. Naopak je dost pravděpodobné, že v této chvíli je v každé ze skupin sada toků, které náleží k různým aplikacím/třídám. Celý tento postup popisuje pseudokód 4.1.

Algoritmus 4.1 Dvoufázové rozdělení trénovacích dat do skupin.

for all $X_i \in \{X_1, ..., X_l\}$ **do**
  $C_i = \text{k–means}(X_i, k)$;
end for
$C = C_1 \cup C_2 \cup .... \cup C_l$;
$X = X_1 \cup X_2 \cup .... \cup X_l$;
for all $x_i \in X$ **do**
  přiřaď x_i k nejbližšímu těžišti z množiny;
end for

Získali jsme tak množinu těžišť, z nichž každé obsahuje množinu síťových toků s jejich statistickými daty. Výhodou je, že máme menší množinu těžišť, které lze využít pro rychlé nalezení přibližné třídy a teprve v rámci množiny toků, spadajících k danému těžišti, hledat konkrétní aplikaci nebo třídu provozu. Další část algoritmu bude tyto informace využívat k rychlé a efektivní klasifikaci.

4.1.3 Klasifikační část algoritmu

Tato část je rovněž rozdělena na dvě fáze. V první fázi najdeme v naší znalostní bázi, kterou jsme vyrobili v trénovací fázi, těžiště, které je nejbližší zkoumanému síťovému toku x. Tím jsme přibližně určili třídu zkoumaného toku a zároveň výrazně omezili množství toků, které by bylo jinak nutné srovnávat. Nyní nastupuje druhá fáze klasifikační části. Odpovídající množinu toků c_j, která náleží k nalezenému nejbližšímu těžišti zkoumaného toku nyní využijeme k určení přesnější třídy zkoumaného toku. To provedeme nalezením toku x_i, který je nejbližší námi zkoumanému toku x. Podle typu aplikace, ke které náleží tok x_i lze rozhodnout, která aplikace pravděpodobně generuje nově klasifikovaný tok x. Celý tento postup popisuje pseudokód 4.2.

Algoritmus 4.2 Klasifikace nového toku nad naučenými daty.

$j = argmin_j \parallel x - c_j \parallel, c_j \in C$;
$nb = argmin_x, \parallel x_i - x \parallel, x_i \in c_j$; return label(nb);

Kapitola 5

Implementace proof of concept detekčních technik

V předchozí kapitole jsem se věnoval popisu vybraných detekčních technik a vysvětlení jakým způsobem fungují. Tato část se bude zabývat implementací prototypu, který by měl ověřit, že lze opravdu takové metody využít a že vůbec v našem prostředí fungují. Cílem zde tedy je vytvořit prototyp klasifikátoru, který by byl schopný korektně klasifikovat síťový provoz do tříd, respektive síťových aplikací, zmíněných dříve zde 3.5. Zároveň jsou kladeny požadavky, aby této činnosti byl schopný přímo za provozu. Využity budou techniky, popsané v předchozí kapitole. Výstupem klasifikátoru by měla být co největší úspěšnost v detekci provozu.

Ověření, zkoušení, výstupy a závěry z této části budou velice přínosné před započetím implementace samotného cílového řešení. Na tomto prototypu by se měly odhalit případné nedostatky, které by mohly zkomplikovat budoucí nasazení.

5.1 Návrh řešení

Implementaci prototypu jsem provedl iterativními kroky. Abych mohl provést nějaké srovnání s jednodušší variantou, provedl jsem nejprve implementaci zjednodušené verze popsaného návrhu, kterou jsem posléze dokončil do cílové podoby. Vznikla tedy zjednodušená varianta, která využívá pro svůj běh pouze algoritmu K-Nearest Neighbor a následně cílová varianta, kombinující oba zmíněné algoritmy — K-Means i K-Nearest Neighbor.

Jako implementační jazyk pro implementaci tohoto prototypu byl zvolen skriptovací jazyk Perl, především pro jeho flexibilitu, možnosti rychlého vytvoření konceptu a dostupnosti knihovny pro práci s PCAP soubory, ve kterých jsem měl uložený testovací provoz. Dá se očekávat, že rychlost řešení implementované ve skriptovacím jazyce nebude dostatečná. Cílové řešení bude s největší pravděpodobností implementováno v jazyce C/C++, který poskytuje větší výkon a lépe zapadá do architektury produktu Kernun UTM. Právě v tomto produktu by měla být výsledná implementace nasazena. Jakým způsobem a konkrétní podoba, je řešeno samostatně v další kapitole 6.

5.1.1 Zjednodušená varianta

Při implementaci zjednodušené varianty prototypu jsem sice vycházel ze zmíněné práce Bar-Yanai a kol. [26], avšak oproti jejich návrhu jsem provedl podstatná zjednodušení, která znamenala využití pouze algoritmu K-Nearest Neighbor. Ve výsledku se bude tato

implementace hodit pro možné srovnání s cílovou implementací a k diskuzi, jakým způsobem dojde k optimalizaci nebo degradaci parametrů výsledného hybridního algoritmu oproti jednodušší verzi.

Jelikož Algoritmus K-Nearest Neighbor se vyznačuje poměrně přesnými výsledky při klasifikaci, očekává se, že jeho úspěšnost bude o něco lepší než u cílové implementace. Podle informací z článku by však tento rozdíl neměl být výrazný. Na druhou stranu však díky své lineární náročnosti v závislosti na velikosti trénovacích dat můžeme čekat, že jeho vykonávání bude trvat podstatně delší dobu než u cílového řešení.

Jelikož se jedná o algoritmus, který ke své činnosti potřebuje vytvořit znalostní bázi z trénovacích dat podobně, jako cílové řešení, je jeho činnost rozdělena na dvě fáze: trénovací a klasifikační.

Úkolem trénovací fáze je vyrobení znalostní báze z trénovacích dat, která bude využita v následující klasifikační fázi. Implementace funguje na principu načítání síťových dat, u kterých je známa příslušnost k odpovídající aplikaci, případně třídě provozu. Tato data jsou předem předpřipravena a měla by být důsledně filtrována, jelikož odchylka v této části pak znamená další chyby při klasifikaci v reálném provozu. Program tedy čte příchozí síťová data, vyrábí si vnitřní reprezentaci síťových toků, kde každému toku má přiřazenou informaci, jaké třídě/síťové aplikaci náleží. Zároveň s touto informací si samozřejmě udržuje i sledované statistické informace o daném toku, včetně pomocných proměnných tak, jak bylo popsáno při prezentaci algoritmu v sekci 4.1.1. Sledovaná data jsou tedy stejná, jako u cílového řešení. Na rozdíl od něj se však tyto informace nesdružují do žádných sad podle příslušné aplikace nebo třídy, ale jsou vedeny společně. Po této fázi není prováděna další práce nad těmito daty a klasifikátor má k dispozici znalostní bázi, kterou je schopen využít pro klasifikaci neznámých síťových dat.

$$d^2 = (V_{i0} - V_{j0})^2 + (V_{i1} - V_{j1})^2 + (V_{i2} - V_{j2})^2 + (V_{i3} - V_{j3})^2 + \dots$$

Obrázek 5.1: Popis výpočtu euklidovské vzdálenosti.

Klasifikační fáze funguje také velice jednoduše. Klasifikátor opět přijímá síťová data, která však již v tuto chvíli nemají označení, jaké síťové aplikaci náleží. Na základě těchto dat si program opět vytváří tabulku síťových toků. Opět po zpracování všech vstupních dat provádí algoritmus klasifikaci nad vytvořenou tabulkou neznámých toků a znalostní bází. Jednoduše vybírá pro každý neznámý tok takový tok ze znalostní báze, který je mu podle hodnotící funkce nejblíže. V této implementaci je hodnotící funkce pouze triviální výpočet euklidovských vzdáleností srovnávaných toků. Euklidovskou vzdáleností je myšleno součet kvadrantů rozdílů všech sledovaných vlastností u obou porovnávaných toků, viz 5.1. Na základě toho určí třídu/aplikaci, která vygenerovala příslušný tok.

5.1.2 Cílová varianta

Při implementaci cílové varianty jsem již plně vycházel ze zmíněné práce Bar-Yanai a kol. [26]. Návrh řešení klasifikátoru využívá statistické metody počítané nad síťovými toky se snahou stejné úspěšnosti klasifikace jak nešifrované, tak šiforvané komunikace a možností použití pro klasifikaci v online režimu, tedy možností klasifikovat síťovou komunikaci za provozu bez znatelné latence. Využívá kombinace algoritmů K-Means a K-Nearest Neighbor.

Samotná implementace prototypu musí respektovat fungování popsaného algoritmu. Ten při své činnosti prochází dvěmi fázemi: trénovací a klasifikační. Jak již bylo zmíněno, tak trénovací fázi je třeba provést na začátku při inicializaci algoritmu, aby vznikla znalostní báze, na základě které pak bude samotná klasifikace prováděna. Poté se přejde do klasifikační fáze, která zpracovává příchozí síťový provoz a generuje výsledky.

Trénovací fáze je část, kdy jsou klasifikátoru předkládány síťová data, která jsou označena do jaké třídy, případně jaké síťové aplikaci, náleží. Z těchto dat je tak klasifikátor schopen vytvořit rozpoznávací znalostní bázi, kterou využívá při samotném provozu na datech, kde žádnou informaci o tom, do jaké třídy/jaké síťové aplikaci přísluší, nemá. Ve vybraném návrhu probíhá trénovací fáze podle diagramu 5.2. V prvním kroku je pro každou síťovou aplikaci zvlášť provedena následující sekvence kroků:

1. Při zpracování vstupních paketů dochází k vytvoření vnitřní reprezentace tabulky síťových toků, kdy každý vstupní tok má informaci o tom, jakou aplikací byl vygenerován.

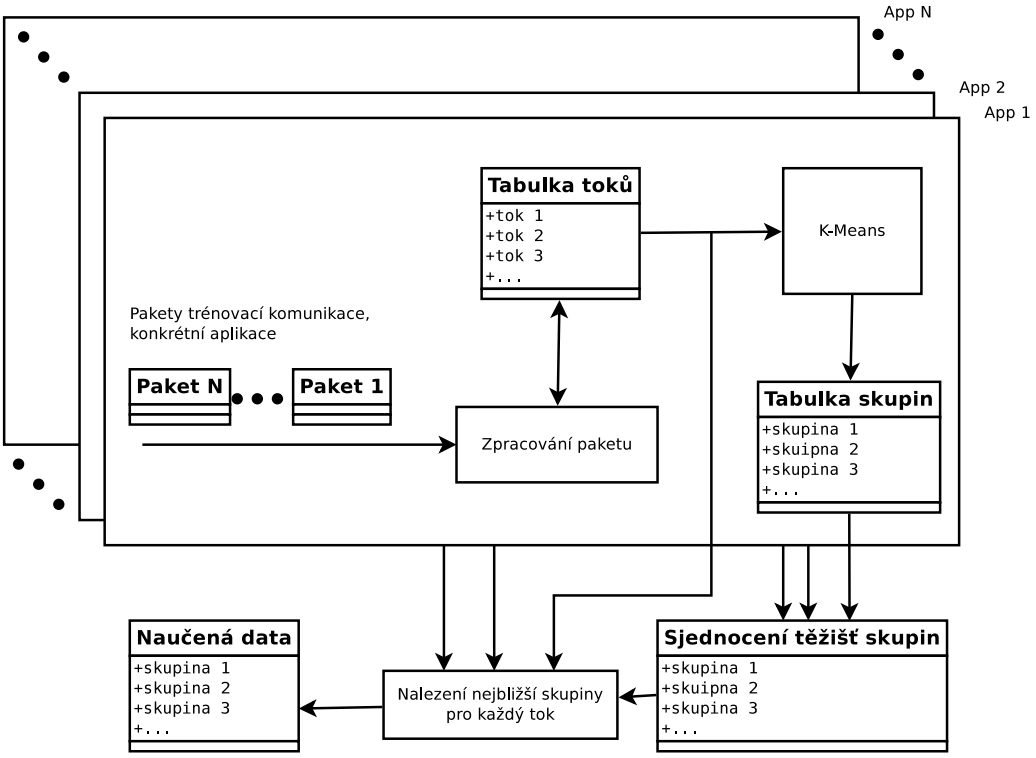
2. S každým příchozím paketem je tento předzpracován a přiřazen k síťovému toku, ke kterému náleží. Pokud se odpovídající síťový tok zatím nenachází v tabulce toků, je vytvořen tok nový.

3. Při každém zpracování nového paketu jsou pro odpovídající tok aktualizovány hodnoty, jež se uchovávají pro statistickou analýzu, tak i pomocné proměnné, které tyto hodnoty usnadňují počítat v průběhu celého trvání toku. Každý síťový tok je sledován pouze do určitého maximálního množství paketů, které určují horní hranici získávání informací o něm. Tato hodnota je na doporučení výsledků referenční práce nastavena na 100 paketů, ale jelikož ji v implementaci používám jako konstantu, tak může být, pro potřeby testování, jednoduše změněna.

4. Jakmile jsou všechny dostupné pakety pro danou aplikaci zpracovány, na výsledné tabulce toků se pomocí algoritmu K-Means provede rozčlenění do skupin. Tímto způsobem získáme určitý počet skupin toků, kde platí, že v každé skupině náleží příslušné toky právě jedné síťové aplikaci.

Tyto operace jsou provedeny pro každou aplikaci zvlášť. Tím získáme pro každou aplikaci množinu toků, rozdělených do několika skupin. Výsledné skupiny toků mezi různými aplikacemi se mohou vzájemně překrývat. Jak ukazuje tento obrázek 4.1, tak taková situace by mohla vést k chybám při klasifikaci. V dalším kroku je proto provedeno sjednocení těžišť všech skupin toků do jedné množiny a následně pro všechny toky přiřazeno nové těžiště takové, pro které platí, že je pro daný tok nejblíže podle hodnotící funkce. V tuto chvíli již neplatí, že každá skupina obsahuje toky jedné aplikace.

Klasifikační fáze je část, kdy klasifikátor má vytvořenou znalostní bázi a je mu na zpracování předáván síťový provoz, vygenerovaný neznámými aplikacemi. Ve vybraném návrhu probíhá klasifikační fáze podle diagramu 5.4. Vstupní příchozí provoz je klasifikátorem zpracováván opět po paketech, které jsou předzpracovány a použity pro tvorbu vnitřní reprezentace tabulky síťových toků, jako tomu bylo v trénovací fázi. Po zpracování všech vstupních paketů je pro každý tok z výsledné tabulky toků nalezena nejbližší skupina z dostupné rozpoznávací databáze. Nejbližší skupinou se myslí taková, jejíž těžiště je podle hodnotící funkce nejblíže příslušnému toku. Jakmile je taková skupina nalezena, ještě je nutné ke klasifikovanému toku nalézt nejbližší síťový tok z dané skupiny pomocí K-Nearest Neighbors



Obrázek 5.2: Diagram trénovací fáze.

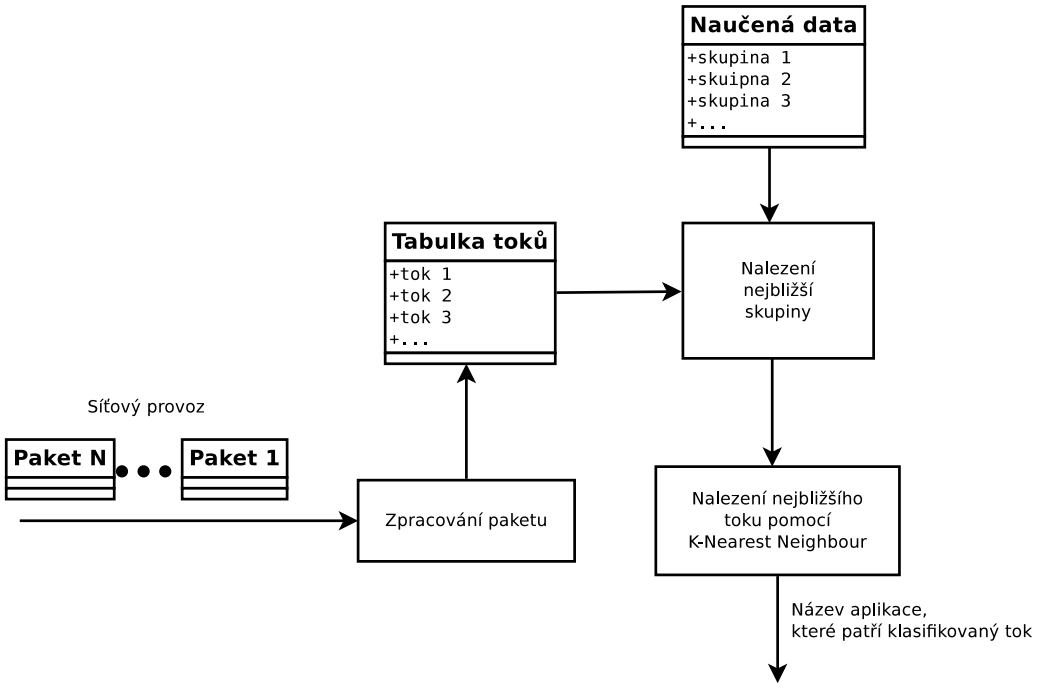
algoritmu. Výsledné přiřazení aplikace klasifikovanému toku je provedeno na základě hodnoty aplikace nalezeného nejbližšího toku pomocí K-Nearest Neighbors algoritmu. V tomto posledním bodě jsem však provedl oproti návrhu z článku drobnou úpravu. Namísto hledání nejbližšího jednoho toku pomocí K-Nearest Neighbor provedu hledání nejbližších pěti toků. Výsledná třída, kam nový tok zařadím je pak taková, která je v této pětici zastoupena nejčastěji, potažmo v případě shody počtů je její prvek blíže, než jiná varianta. Číslo pět bylo vybráno empirickým ověřením.

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Obrázek 5.3: Normalizační funkce použitá v hodnotící funkci.

Implementace hodnotící funkce v triviální podobě výpočtu euklidovské vzdálenosti v tomto případě již nevyhovovala, jelikož velice rapidně klesla úspěšnost detekce — viz výsledky dále. Proto jsem přistoupil k drobné úpravě v podobě normalizace jednotlivých sledovaných statistických hodnot a teprve následnému určení euklidovské vzdálenosti srovnávaných toků. Normalizaci jsem provedl jednoduchým postupem, kdy jsem od hodnoty příslušné sledované veličiny odečetl nejmenší hodnotu výskytu této veličiny v trénovacích datech a tento rozdíl vydělil maximálním rozsahem hodnot této veličiny v trénovacích datech — tedy rozdíl ma-

ximální a minimální hodnoty 5.3. Teprve takto normalizované hodnoty jsem využíval pro další výpočty v hodnotící funkci. Výsledky fungování obou variant lze vidět dále.



Obrázek 5.4: Diagram klasifikační fáze.

Jak již bylo výše zmíněno, tak toto řešení využívá i algoritmus K-Means. Ten ke své činnosti potřebuje zadat počet skupin, do kterých má rozdělovat příslušné vzorky. Empirickým zkoušením jsem došel na nejvhodnější číslo výchozího množství skupin pro aplikaci jako 8. Tato hodnota je použita jako výchozí hodnota proměnné *num_of_init_clusters* ve zdrojovém kódu. Ideální hodnota tohoto čísla však může být odlišná pro různé druhy trénovacích dat. Při zkoušení bylo vidět, že při dalším zvýšení tohoto čísla začala již přesnost detekce nepatrně klesat. Pravděpodobně to bylo způsobeno větší fragmentací trénovacích dat do více skupin, kdy se mohla projevit nedokonalost hodnotící funkce, která vybrala pro upřesňující detekci nevhodnou skupinu. Naopak se však při zvýšení počtu skupin obvykle maličko zlepšila rychlost detekce. Opět je to důsledek větší fragmentace trénovacích dat, kdy klasifikátor nemusel procházet tak mohutné množiny dat.

5.2 Testovací platforma

V předchozí sekci je popsána implementace vybrané detekční metody, včetně zjednodušené verze. Pro získání základních informací o jejich úspěšnosti a náročnosti klasifikace, je nutné provést sadu testů nad testovacími daty. Všechny výsledky, uvedené v následujících sekcích budou vycházet ze zde popsané testovací platformy, uvedené v tabulce 5.1 a způsobů spouštění aplikace v průběhu testování.

CPU	Intel Core i5 M 480 @ 2,67GHz, 2 jádra, HyperThreading
RAM	8 GB single channel
Pevný disk	320 GB, 5400 RPM
Operační systém	FreeBSD 9.1, 64-bit
Interpret	Perl 5.14.2

Tabulka 5.1: Testovací hardware, na kterém probíhalo měření výkonnosti.

5.3 Testovací sada dat

Pro vytvoření testovací sady dat byl využit nástroj Wireshark [38], který slouží k zachytávání síťové komunikace a jeho ekvivalent pro příkazový řádek tcpdump. Zachytávání probíhalo částečně na podnikové síti, kde bylo připojeno přibližně 9 klientských PC a na lokální síti s jedním klientským PC. Výsledné soubory s nachytaným reálným provozem byly posléze ručně zkontrolovány tak, aby neobsahovaly očividný nechtěný provoz, který se netýkal zachytávané aplikace — například DNS provoz při zachytávání HTTP provozu a podobně. Při zachytávání vzorků pro každou aplikaci byla v případě lokální sítě s jedním PC vyvinuta snaha docílit toho, aby na sledovaném síťovém rozhraní probíhala komunikace právě a jen dané aplikace. Toho bylo třeba dosáhnout například při získávání vzorků pro aplikaci Facebook nebo Google dokumenty, jelikož komunikace těchto aplikací jsou ve výsledném zachyceném provozu již těžko dohledatelné. Takovým způsobem byly nachytány vzorky dat pro jednotlivé aplikace. Žádné filtrování extrémě krátkých nebo naopak dlouhých toků nad zachycenými daty nebylo prováděno. Tabulky 5.2 a 5.3 tak zobrazují zevrubnou podobu testovací sady dat.

Aplikace	Počet toků	Počet paketů
HTTP	633	22 666
HTTPS	567	42 884
Facebook	264	23 251
Google dokumenty	344	57 785
BitTorent	404	332 222
Skype	199	11 871
SMTP	651	24 001
SSH	200	373 553
DNS	742	1 489
SNMP	362	2 680

Tabulka 5.2: Trénovací sada dat.

5.4 Výsledky detekce

Pro všechna prováděná měření a testování jsem využil jak stejný testovací hardware, tak stejná testovací data. Snažil jsem se v průběhu testování zachovat co možná nejpodobnější podmínky při všech měření. Testování probíhalo v offline režimu, tj. na předem připravených souborech se zachyceným provozem pomocí nástroje Wireshark [38]. O provozu bylo známo, jaké síťové aplikaci náleží. Ve všech spuštěných testech byl klasifikátor naučen pomocí stejné sady vstupních vzorků 5.2, které obsahovaly dostatek toků pro naučení

Aplikace	Počet toků	Počet paketů
HTTP	346	14 407
HTTPS	88	8 406
Facebook	91	10 369
Google dokumenty	96	17 194
BitTorent	365	39 989
Skype	85	8 545
SMTP	138	2 668
SSH	108	23 316
DNS	622	1 250
SNMP	55	406

Tabulka 5.3: Testovací sada dat.

všech detekovaných síťových aplikací. Následně byly takto naučenému klasifikátoru postupně předkládány soubory se zachyceným provozem konkrétní aplikace 5.3, na kterých se zkoušela úspěšnost a rychlost detekce. Každé měření pro každou aplikaci proběhlo celkem třikřát a do výsledných tabulek, na které je odkazováno vždy u popisu příslušné varianty, byly zaneseny aritmetické průměry jednotlivých výsledků. Tímto způsobem se omezil vliv možného okamžitého výkyvu ve výkonu testovací platformy.

5.4.1 Zjednodušená varianta

Tabulka 5.4 zobrazuje výsledky úspěšnosti detekce a rychlosti implementované zjednodušené varianty. Je patrné, že rychlost klasifikace je u všech měření téměř stejná. Neplatí tedy, že pro různé třídy a aplikace vychází rychlost klasifikace různě. Přesto by se to u některých aplikací očekávat dalo. Platí totiž, že některé aplikace nebo třídy obsahují toky dlouhé typicky jen pár paketů (například DNS), což oproti delším spojením (například SSH) znamená, že hodnoty sledovaných statistických veličin jsou podstatně menší a tudíž s nimi prováděné matematické operace nepatrně rychlejší. Pravděpodobně je však tento rozdíl velice mizivý a je potlačen výkonným hardwarem, na kterém probíhalo testování.

Úspěšnost detekce je pro různé aplikace podstatně odlišná. Nejlépe dopadla detekce aplikace SSH s úspěšností více než 97 %. Naopak nejhůře dopadla detekce aplikace Facebook s úspěšností něco přes 24 %. Je vidět, že tato zjednodušená varianta má své rezervy, které by bylo třeba odladit. Zcela jistě by mohlo pomoci zvýšení počtu trénovacích dat. Avšak obával bych se v tomto případě další degradace již tak pomalé rychlosti klasifikace.

Za pozornost také stojí porovnání výsledků detekce aplikací Facebook a Google dokumenty. Tyto jediné jsou z naší sady, oproti ostatním (které reprezentují samotné aplikační protokoly), webové aplikace, což je v podstatě podmnožina HTTP a HTTPS provozu. V obou případech se dá říci, že především HTTPS provozu, jelikož obě dvě komunikují přes šifrované HTTP. Dala se očekávat nižší úspěšnost detekce, jelikož část toků je pravděpodobně dost podobná obecnému HTTPS provozu, což se potvrdilo — ze všech aplikací dosahují nejhorších výsledků. Na druhou stranu ale minimálně u aplikace Google dokumenty bylo dosaženo více než 56 % úspěšnosti, což naznačuje, že možnost detekce pomocí této metody není zcela vyloučena.

Celková zátěž klasifikátoru na testovanou platformu se pohybovala mezi 20 a 35 % výkonu procesoru. Paměťová náročnost se pohybovala od úvodních 40 MB až k přibližně 53 MB. Rychlost klasifikace v řádech jednotek toků za vteřinu není příliš dobrý výsledek

Aplikace	Úspěšnost detekce [%]	Trvání detekce [s]	Rychlost detekce [toků/s]
HTTP	68,79	39,53	8,75
HTTPS	65,90	10,03	8,77
Facebook	23,08	10,33	8,81
Google dokumenty	53,13	10,95	8,77
BitTorent	67,12	41,70	8,75
Skype	68,24	9,86	8,62
SMTP	79,71	15,61	8,84
SSH	97,22	12,53	8,61
DNS	94,69	71,70	8,67
SNMP	90,91	6,24	8,81

Tabulka 5.4: Výsledky detekce zjednodušené varianty klasifikátoru.

a v reálném provozu by byl značně nedostačující.

5.4.2 Cílová varianta

Nyní se pojďme podívat na výsledky měření cílové varianty. Jak jsem se zmínil dříve, pro tuto cílovou variantu bylo nutné upravit implementaci hodnotící funkce. V tabulkách vynesu výsledky implementací obou dvou hodnotících funkcí, aby bylo možné srovnávat, jakým způsobem se podařilo zlepšit vlastnosti detekce.

Tabulka 5.5 zobrazuje výsledky detekce a rychlosti implementovaného řešení, s použitím stejné hodnotící funkce, jako u předchozí zjednodušené verze. Na první pohled lze pozorovat viditelné zvýšení rychlosti klasifikace. U většiny aplikací se jedná o více než dvojnásobek až trojnásobek původní rychlosti, avšak u klasifikace aplikace SSH je toto zrychlení více než osminásobné. Zrychlení klasifikace bylo očekávatelné, jelikož navržený hybridní algoritmus odstraňuje nevýhodu algoritmu K-Nearest Neighbor a to právě jeho nedostatečnou rychlost, která lineárně roste s velikostí trénovacích dat. Rychlost detekce mohla být navýšena díky využití algoritmu K-Means, který provádí rozřazení trénovacích dat do menších skupin a tím pádem není nutné vždy procházet celou bázi trénovacích dat.

Aplikace	Úspěšnost detekce [%]	Trvání detekce [s]	Rychlost detekce [toků/s]
HTTP	92,49	15,47	22,37
HTTPS	25,00	4,54	19,37
Facebook	6,59	3,55	25,62
Google dokumenty	14,58	3,26	29,42
BitTorent	42,70	11,04	32,87
Skype	21,18	3,46	24,57
SMTP	15,94	6,78	20,35
SSH	64,81	1,58	68,26
DNS	65,59	27,96	22,24
SNMP	14,55	3,54	15,52

Tabulka 5.5: Výsledky detekce hybridním algoritmem — jednoduchá hodnotící funkce.

Co se týká úspěšnosti detekce, tak je zde vidět rapidní propad oproti předchozí zjed-

nodušené variantě. Nejlépe dopadla detekce aplikace HTTP s úspěšností více než 92 %. Naopak nejhůře dopadla opět detekce aplikace Facebook s úspěšností tentokrát pouze něco přes 6,5 %. Důvod k takovému propadu se dá vysvětlit nedokonalou hodnotící funkcí. Oproti předchozí verzi je optimalizováno procházení znalostní báze takovým způsobem, že je nejprve nutné vybrat skupinu, ve které se bude hledat nejbližší tok. Pokud se tento prvotní výběr skupiny provede špatně, je nutně negativně ovlivněno další hledání.

Celková zátěž klasifikátoru na testovanou platformu se pohybovala opět mezi 20 a 35 % výkonu procesoru. Paměťová náročnost se v tomto případě pohybovala mezi 40 MB až 50 MB. Rychlost klasifikace v tomto případě stoupla na jednotky desítek toků za vteřinu. Tato rychlost stále není příliš přijatelná v reálném provozu, avšak je zde vidět náznak toho, že tato metoda má své rezervy, které je možné odstranit a výslednou rychlost ještě navýšit.

Nakonec si pojďme ukázat výsledky měření cílového návrhu s vylepšenou hodnotící funkcí o normalizaci sledovaných statistických veličin. Výsledky obsahuje tabulka 5.6. Rychlost klasifikace v tomto případě opět klesla téměř na stejné hodnoty, jako měla zjednodušená varianta. U některých aplikací je to nepatrně nad tuto hodnotu, u některých naopak ještě nepatrně pod. Zdá se, že výkonnostní nárůst, který způsobilo rozdělování toků ve znalostní báze do více skupin, téměř zcela smazala úprava hodnotící funkce o normalizaci položek, se kterými pracuje. Přidáním pomocných výpočtů a tím, že je tato funkce používána velice často, tak rychlost klasifikace klesla zpět na původní hodnoty zjednodušené verze. Zcela zásadní se tedy jeví provést optimalizaci výpočtů v hodnotící funkci.

Aplikace	Úspěšnost detekce [%]	Trvání detekce [s]	Rychlost detekce [toků/s]
HTTP	86,99	38,68	8,94
HTTPS	69,32	9,54	9,22
Facebook	28,57	8,79	10,36
Google dokumenty	57,29	8,80	10,91
BitTorent	65,38	40,05	9,09
Skype	29,41	10,26	8,28
SMTP	65,22	17,69	7,80
SSH	54,63	11,81	9,14
DNS	22,19	81,22	7,66
SNMP	63,64	7,22	7,61

Tabulka 5.6: Výsledky detekce hybridním algoritmem — normalizující hodnotící funkce.

Při pohledu na úspěšnost detekce je vidět znatelné zlepšení oproti předchozí variantě — tedy s hybridním algoritem, avšak triviální hodnotící funkcí. Nejlépe dopadla detekce aplikace HTTP s úspěšností téměř 87 %. Naopak nejhůře dopadla detekce aplikace DNS s úspěšností něco málo přes 22 %. Lze tedy vidět, že upravení hodnotící funkce mělo velice znatelný vliv na celkovou úspěšnost detekce. Přesto by si však hodnotící funkce zasloužila ještě další zlepšení, jelikož oproti zjednodušené variantě, využívající pouze algoritmus K-Nearest Neighbor, úspěšnost klasifikace není lepší. Na druhou stranu se dá u této varianty předpokládat lepší připravenost na větší sadu trénovacích dat, která by dále neměla výpočty této metody příliš zpomalovat.

Stejně jako u zjednodušené varianty, i zde se podívejme na výsledky detekce aplikací Fa-

cebook a Google dokumenty. Jak jsem již psal, tak tyto jediné jsou z naší sady, oproti ostatním (které reprezentují samotné aplikační protokoly), webové aplikace. Tedy jsou podmnožinou HTTP a HTTPS provozu. Z tohoto důvodu se u nich očekávala nižší úspěšnost detekce, což se potvrdilo. Ovšem pokud klasifikátor funguje alespoň trochu dobře, dá se také předpokládat, že pokud špatně vyhodnotí tok, který patří aplikaci Facebook nebo Google dokumenty, s velkou pravděpodobností by měl tedy takový tok označit buď jako HTTP nebo HTTPS provoz. V tabulce 5.7 jsou vypsány výsledky klasifikace po jednotlivých aplikacích, včetně jejich false positive, tedy chybě vyhodnocených výsledků výsledků. Z tabulky je patrné, že jak u aplikace Facebook, tak i u aplikace Google dokumenty jsou false positive u HTTP a HTTPS provozu poměrně vysoké. Také je z tabulky patrné, že právě HTTP provoz je dost často false positive i u ostatních aplikací.

[%]	HTTP	HTTPS	Fcbk	Gdocs	BitTor.	Skype	SMTP	SSH	DNS	SNMP
HTTP	86,99	2,31	3,47	1,45	0,29	0,29	3,76	1,45	0	0
HTTPS	11,37	69,32	9,09	4,55	2,27	2,27	1,14	0	0	0
Fcbk	17,58	37,36	28,57	6,59	7,69	2,20	0	0	0	0
Gdocs	22,92	9,38	6,25	57,29	1,04	0	1,04	2,08	0	0
BitTor	10,44	0,82	2,20	2,47	65,38	3,57	0	0	15,11	0
Skype	24,71	2,35	12,94	3,53	4,71	29,41	1,18	0	14,12	7,06
SMTP	11,59	6,52	2,90	1,45	0	11,59	65,22	0,72	0	0
SSH	28,70	2,78	1,85	7,41	0	4,63	0	54,63	0	0
DNS	29,58	0	28,46	19,77	0	0	0	0	22,19	0
SNMP	27,27	0	0	0	9,09	0	0	0	0	63,64

Tabulka 5.7: Výsledky detekce hybridním algoritmem — normalizující hodnotící funkce. Včetně false positives.

Celková zátěž klasifikátoru na testovanou platformu se pohybovala mezi 15 a 30 % výkonu procesoru. Paměťová náročnost se pohybovala mezi 43 MB až 55 MB. Rychlost klasifikace, která dosahuje v průměru těsně pod deset toků za vteřinu, je třeba ještě navýšit.

5.5 Zhodnocení obou variant

V této části jsem otestoval tři varianty implementace prototypu detekčního modulu. Zjednodušená verze kupodivu dosahuje nejlepších výsledků a rychlostně je srovnatelná s cílovou variantou, které však musela být vylepšena hodnotící funkce, což mělo za následek snížení rychlosti klasifikace (tím se právě obě varianty rychlostně srovnaly). Zjednodušená varianta má však nevýhodu v podobě lineární závislosti náročnosti na rostoucí množině trénovacích dat. Tento neduh by z větší části neměl platit u cílové varianty. Každopádně v obou případech je rychlost klasifikace v řádu deseti toků za vteřinu pro praktické použití využitelná velmi omezeně. Je třeba provést optimalizaci hodnotící funkce, která má na svědomí nejvíce režie, jelikož je v rámci výpočtu volána velmi často. Zároveň je třeba tuto funkci i vylepšit tak, aby se ještě navýšila úspěšnost detekce algoritmu. Rovněž zvýšení rychlosti přinese cílová implementace, která bude provedena v jazyce C/C++.

Ohledně použitelnosti vybraného algoritmu pro naše potřeby se dá říci, že použitelný je. Jak již bylo zmíněno, je třeba zapracovat jak na úspěšnosti detekce, tak na její rychlosti. Co se týká podezření, že by tento algoritmus, který je založený na práci se statistickými daty síťových toků, mohl mít problém s detekcí různých webových aplikací, tak to se částečně

potvrdilo. Jak aplikace Facebook, tak Google dokumenty neměly příliš vysokou úspěšnost v detekci (i když zároveň ne nejhorší). Právě jako false positive byl v těchto případech nejčastěji označen provoz HTTP a HTTPS. Na druhou stranu ale minimálně u aplikace Google dokumenty bylo dosaženo více než 57 % úspěšnosti, což naznačuje, že detekce pomocí této metody není úplně vyloučena. Osobně věřím, že při bližším zaměření na hodnotící funkci a případně vzatí potaz dalších statistických ukazatelů se může úspěšnost detekce konkrétních webových aplikací ještě daleko zlepšit.

Kapitola 6

Návrh zakomponování do produktu Kernun UTM

Předchozí kapitola se zabývala popisem a implementací prototypu, který měl ověřit použitelnost vybraného detekčního algoritmu. Závěrem předchozí kapitoly je, že zvolený algoritmus a způsob je použitelný, avšak rohodně nutně potřebuje provést řadu úprav, které jednak zvýší jeho úspěšnost, tak i rychlost klasifikace. Zároveň by nebylo vhodné, omezit se při návrhu zakomponování cílového řešení do současné architektruy produktu pouze na tento typ detekce — tedy detekce na základě statistických informací o síťových tocích. Bylo by vhodné architekturu navrhnout tak, aby respektovala i možnost použití technik, založených na zkoumání obsahu přenášených paketů (DPI). Po zkušenostech s implementací prototypu detekčního modulu se tedy v této kapitole zaměřím na způsob, jakým by bylo možné tuto funkcionalitu začlenit do stávajícího produktu firmy Trusted Network Solutions a.s.

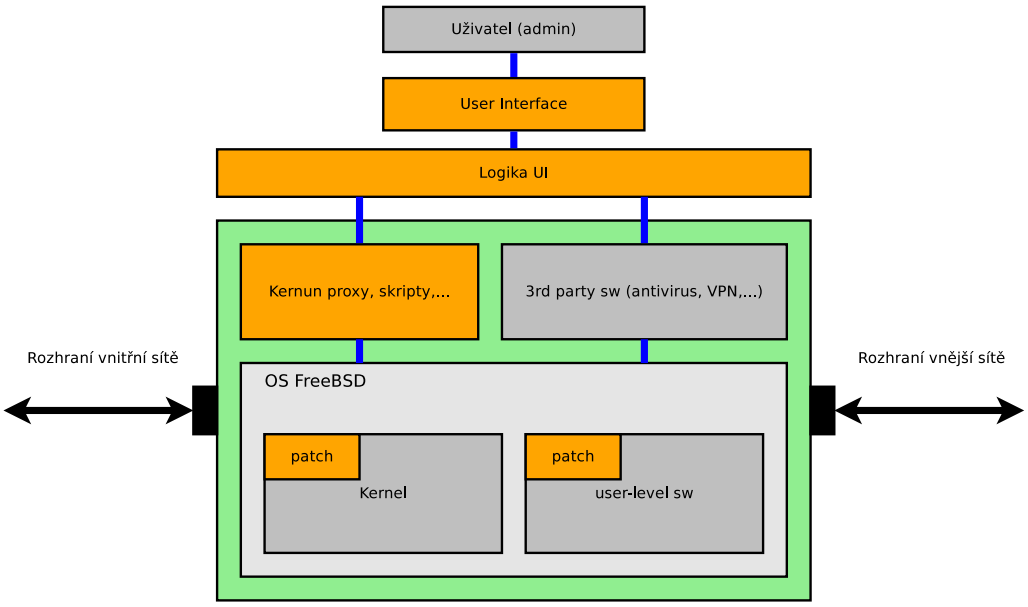
Kernun UTM [39] je produkt, vyvíjený českou firmou Trusted Network Soulutions a.s., která se zabývá vývojem a správou síťových bezpečnostních řešení. Jedná se o proxy firewall s prvky aplikačního firewallu, postavený nad operačním systémem FreeBSD. Jeho základ tvoří konfigurovatelné proxy pro různé protokoly (HTTP, DNS, FTP, SMTP, apod.), dále využívá stavový paketový filtr (pf) pro jemnější řízení provozu — prioritizace, omezování rychlosti — sadu různých proprietárních skriptů a dalších programů pro zajištění specifických vlastností produktu. Také podporuje některé aplikace třetích stran, dodávaných na základě výběru konkrétní varianty výrobku a podle specifikace požadavků zákazníka, jako je například antivirus, antispam, VPN a podobně. Pro zajištění požadovaných vlastností jsou samotné jádro systému FreeBSD a některé aplikace z user-level vrstvy upraveny proprietárními patchi[1]. Celý tento systém lze konfigurovat, ovládat a kontrolovat jeho stav pomocí grafického uživatelského rozhraní nebo pomocí řádkového uživatelského rozhraní, které jsou funkčně ekvivalentní.

6.1 Současná architektura Kernun UTM

Základní pohled na strukturu celého systému ilustruje obrázek 6.1. Prakticky všechny podrobnosti o způsobu práce se systémem a jeho rozhraní se lze dočíst v uživatelské příručce [40], která je dodávána spolu se systémem.

[1] Patch (záplata) — zásah do zdrojového kódu, který opravuje chyby, přidává nové vlastnosti nebo upravuje chování příslušného softwaru.

Správce systému přistupuje k zařízení přes grafické nebo řádkové uživatelské rozhraní, odkud může jednoduše a pohodlně konfigurovat všechny součásti, sledovat a měnit jejich stav, analyzovat systémové a síťové záznamy, generované statistiky provozu a podobně. Grafické uživatelské rozhraní je implementováno jako nativní aplikace v Qt frameworku [41], v současné době překládané s podporou systémů GNU/Linux, FreeBSD a Microsoft Windows. Jedná se o grafickou nadstavbu nad řádkovým uživatelským rozhraním — KAT - Kernun Administration Tool. Toto rozhraní je pak ovládáno v shellu jako klasická konzolová aplikace.



Obrázek 6.1: Architektura Kernun UTM.

Z pohledu systému se samotná konfigurace rozděluje do dvou úrovní. První úroveň představuje konfigurace, reprezentující přepis nastavení, dostupných z uživatelského rozhraní. Zde se ukládají v jednotném a lidsky relativně dobře čitelném formátu hodnoty, které nastavil správce systému pomocí uživatelského rozhraní. Použitým formátem k tomuto účelu je jazyk *cml*[2], který byl pro toto použití interně vyvinut. Jeho výhodou je přehlednost zápisu a rozdělení konfiguračních součástí do různých logických sekcí. Obsah této konfigurace je pak následně transformován na takzvanou nízkoúrovňovou konfiguraci, která představuje právě druhou úroveň konfigurace zmíněnou dříve. To jsou samotné konfigurační soubory pro paketový filtr, jednotlivé proxy, pomocné skripty, služby systému (sshd, ntpd...) a podobně. Administrátor je však od této vlastnosti odstíněn a o samotou transformaci se stará logika uživatelského rozhraní.

Hlavními konfiguračními součástmi jsou paketový filtr a jednotlivé proxy, dostupné pro různé protokoly. Tyto aplikace povolují, omezují, zakazují, prioritizují a směrují síťový provoz, který prochází přes zařízení. Paketový filtr je součástí samotného operačního systému, který je přes uživatelské rozhraní patřičně konfigurován. Probíhající síťový provoz je v rámci

[2]cml (Configuration Meta-Language) — je způsob zápisu konfigurace zařízení Kernun UTM. Více o tomto jazyku se lze dočíst v uživatelské příručce produktu Kernun UTM [40]

IP stacku systému buď předán paketovému filtru nebo postoupen na vyšší úroveň, kde mohou poslouchat instance síťových proxy. K dispozici jsou následující typy proxy: DNS, FTP, H323, HTTP, IMAP4, POP3, SIP, SMTP, TCP a UDP. Lze nakonfigurovat více instancí od každé z těchto proxy, z nichž může každá poslouchat na různých síťových rozhraních, IP adresách a různém rozsahu portů. Každá z těchto instancí pak může mít rozdílnou konfiguraci, podle potřeby příslušné části sítě. Další součástí systému jsou pomocné skripty a skripty generující statistiky využití sítě a zařízení. Ty zahrnují přístupy jednotlivých uživatelů, množství přenesených dat, informace o bezpečnostních incidentech a podobně.

Samotný produkt je vyvíjen nad operačním systémem FreeBSD, který je bezpečným a spolehlivým systémem, vhodným pro serverové nasazení. Dostupnost velkého množství síťových nástrojů a rozsáhlá podpora síťových technologií pak usnadňuje vývoj komplexnějších síťových řešení nad jeho základy. Otevřená licence dovoluje přebírat zdrojové kódy jádra samotného systému i dalších aplikací a volně je modifikovat a rozšiřovat. Sada takových úprav je i součástí produktu Kernun UTM. Jedná se o zásahy do samotného jádra systému i některých dalších systémových aplikací. Tyto změny rozšiřují a upravují některé části, týkající se síťové infrastruktury, jako například podporu řízení šířky pásma.

Pro nasazení ve větších sítích, kde se využívá autentizace připojovaných zařízení a uživatelů, podporuje Kernun UTM autentizaci pomocí metod Kerberos [42] a NTLM [43]. Doporučovaným a preferovaným způsobem autentizace je metoda Kerberos, především díky vyšší náročnosti NTLM v některých případech, nedostatečné bezpečnosti šifrování použitím RC4 šifry, podporou vzájemné autentizace a jednodušší správou konfigurace prvků v síti [44].

Jelikož některé organizace požadují ještě lepší kontrolu nad datovými toky v jejich síti, byla od verze Kernun UTM 3.8.2 přidána podpora HTTPS inspekce. To jednoduše znamená, že zařízení Kernun UTM vystupuje v komunikaci při šifrovaném spojení jako prostředník, který provádí při samotném přenosu dat jejich rozšifrování, kontrolu a opětovné zašifrování před přeposláním klientovi. Jedná se tedy o implementaci klasického MITM (Man in the middle [45]) útoku, avšak v tomto případě sloužícímu ke sledování sítě za účelem její ochrany. Uživatelé toto poznají jednoduše tím, že stránky, které využívají HTTPS spojení se jim nepředstavují vlastním původním HTTPS certifikátem, ale certifikátem, který je vyrobený správcem sítě, využívajícím místní certifikační autoritu. Ve výsledku je tak zařízení schopné číst a analyzovat data komunikace, která by jinak probíhala šifrovaně. Tato vlastnost se může hodit pro budoucí snahu o detekci aplikací, které skrz zařízení komunikují, jelikož není omezena dostupnost obsahu samotné komunikace. Avšak v rámci mé práce, která se zaměřuje na detekci pouze v rámci statistické analýzy není mnou tato vlastnost využívána. Je však vhodné s touto možností počítat při návrhu architektury, aby bylo možné při případném budoucím rozšiřování tuto vlastnost při detekci aplikací jednodušše využít.

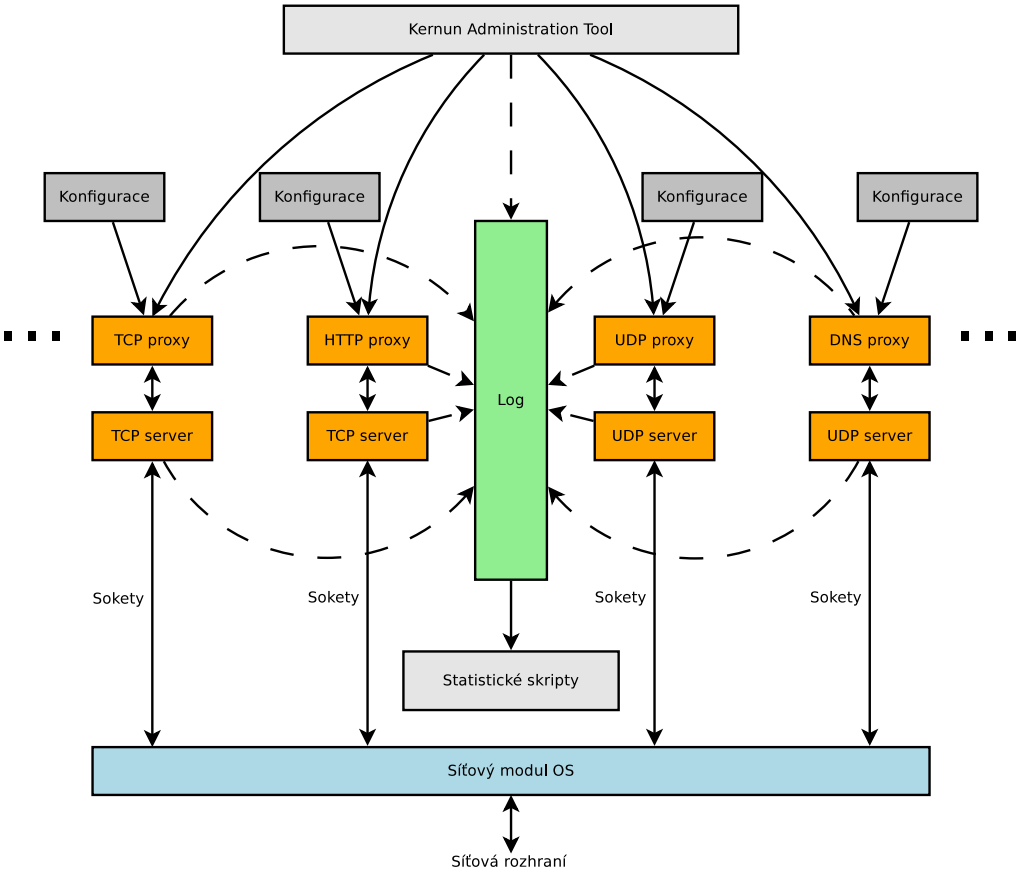
6.1.1 Architektura proxy

Z celkové architektury je pro detekci síťových aplikací nejzajímavější ta část systému, která je nejblíže samotným síťovým rozhraním, na kterých probíhá komunikace. Proto je vhodné ještě před dalším pokračováním popsat podrobněji architekturu implementace samotných proxy, které Kernun UTM nabízí.

Jak již bylo zmíněno, tak administrátor má k dispozici několik druhů proxy, podle potřebného aplikačního protokolu, přičemž vždy je pro přenos použit TCP nebo UDP protokol nižší vrstvy. Toho je využito při návrhu. Existující architektrua obsahuje implementaci dvou

typů serverů — TCP a UDP, které vytvářejí potřebné síťové sockety, na kterých dokáží komunikovat a vytvářet nové procesy pro obsluhu klientů. Samotné proxy pak implementují odpovídající chování a komunikaci těchto serverů tak, jak je třeba z hlediska příslušného aplikačního protokolu. Takto navržená architektura velice efektivně omezuje duplicitu kódu a dovoluje sdílení stejného kódu mezi více funkčními částmi.

Konfiguraci si proxy čte při spouštění z odpovídajícího konfiguračního souboru. Na jeho základě vytvoří odpovídající síťové sockety a provede další konfigurační kroky, definované v konfiguraci. Posléze čeká na připojení od klientů a provádí obsluhu jejich spojení. V průběhu své činnosti provádí zápis o spojeních do logovacích souborů, které jsou posléze využívány pro tvorbu statistik, zkoumání síťové komunikace a zdrojem informací v případě prováděných úprav struktury nebo fungování sítě. Aktuální architekturu implementace proxy popisuje obrázek 6.2.



Obrázek 6.2: Současná podoba architektury proxy.

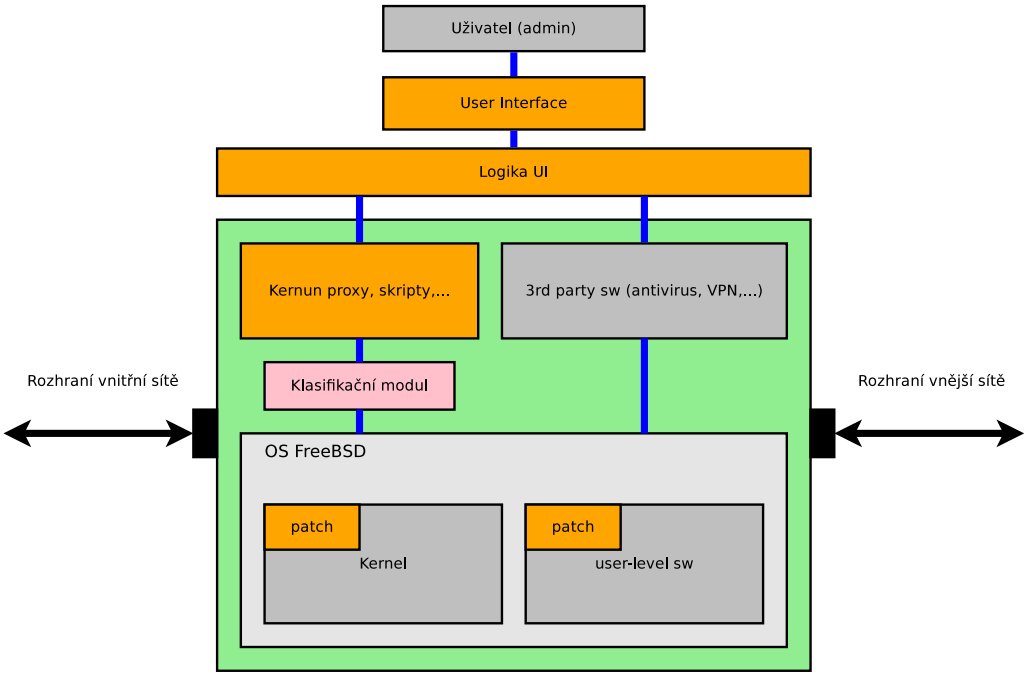
6.2 Možné způsoby zakomponování detekčního systému do Kernun UTM

Navržený detekční systém, založený na dříve popsaném algoritmu, pro svoji práci potřebuje informace o síťových tocích, které v síti probíhají a na základě nichž provádí samotnou detekci. Potřebuje mít přístup k jednotlivým TCP či UDP paketům, na jejichž základě si vytváří a udržuje tabulku síťových toků, nad kterou provádí další výpočty související s hledáním nejvhodnějšího kandidáta na aplikaci, jíž příslušná komunikace patří. V případě využití detekce na základě DPI mohou být dalším zdrojem dat pro detekci samotná data v TCP a UDP paketech. Přestože v popsaném algoritmu, který implementuji tyto informace nevyužívám, jelikož naším cílem je zaměřit se primárně na detekci na základě statistických informací o síťových tocích, tak pro případné budoucí rozšíření je vhodné s tímto možným požadavkem počítat a při tvorbě samotného návrhu jej brát v potaz. Jako zdroj dat pro tyto další výpočty je nutná znalostní báze, která obsahuje dříve naučené vlastnosti a parametry síťových toků, které patří konkrétní aplikaci. Nutným požadavkem je také možná škálovatelnost výsledného návrhu tak, aby byl schopen spolehlivě fungovat i při vyšší zátěži systému a síťové komunikace. Kromě detekce samotných aplikací by měl návrh počítat i s možností budoucího jednoduchého rozšíření o blokování takto detekovaných aplikací. Cílem této sekce je navrhnout nejvhodnější zakomponování detekčního modulu do současné architektury produktu Kernun UTM, případně navrhnout úpravy současné architektury takovým způsobem, aby bylo možné detekční modul lépe zakomponovat. Bude rozebráno několik možností, z nichž nejvhodnější bude vybrána a popsána podrobněji. Pro zjednodušení nás v aktuálním návrhu nezajímá případ, kdy je využíván paketový filtr 'pf', který je také součástí produktu Kernun UTM. Rozšíření podpory detekčního modulu i v případě využívání paketového filtru je plánováno až po ověření a ustálení návrhu detekčního modulu a jeho navázání v samotných proxy.

Z výše uvedeného vyplývá, že pro správnou funkčnost detekčního modulu je nutné, aby měl přístup k samotným paketům, které jsou přenášeny po síti a z nichž pak bude sbírat informace pro samotnou detekci. Současná architektura produktu Kernun UTM je v tomto směru dostatečně flexibilní a bude možné detekční modul zakomponovat do produktu bez jejích větších zásahů a úprav. Zcela jistě bude vhodné, aby výsledků detekce mohly využívat jak samotné proxy, tak i další proprietárně vyvinuté programy, které tak v návaznosti mohou poskytnout více informací o provozu přes zařízení. Obrázek 6.3 ilustruje předpokládané umístění detekčního modulu do architektury Kernun UTM. Podrobnější návrhy budou rozebrány v dalších sekcích.

6.2.1 Integrace do společné části proxy

Jako přímočaré řešení se nabízí zakomponovat klasifikační modul přímo do samotného kódu proxy. Obrázek 6.2 popisuje základní pohled na současnou podobu proxy, využívaných v Kernun UTM. Předpokládá se, že nejvíce užitečným bude implementovaný detekční modul v TCP, UDP a HTTP proxy. Je tomu tak proto, že například samotná SMTP proxy přirozeně rozpoznává, že v ní opravdu probíhá SMTP komunikace (v opačném případě zahlásí chybu v samotném komunikačním protokolu a typicky ukončí spojení) a jemnější rozpoznávání není v tuto chvíli cílem. Podobně tak i další proxy, specifické pro konkrétní protokol. V případech TCP a UDP proxy se však dostáváme na úroveň komunikace, kde mohou komunikovat opravdu různé aplikace. Podobně HTTP proxy. I když v tomto případě se jedná opět o komunikaci pouze pomocí HTTP protokolu, mohou skrze ni komunikovat

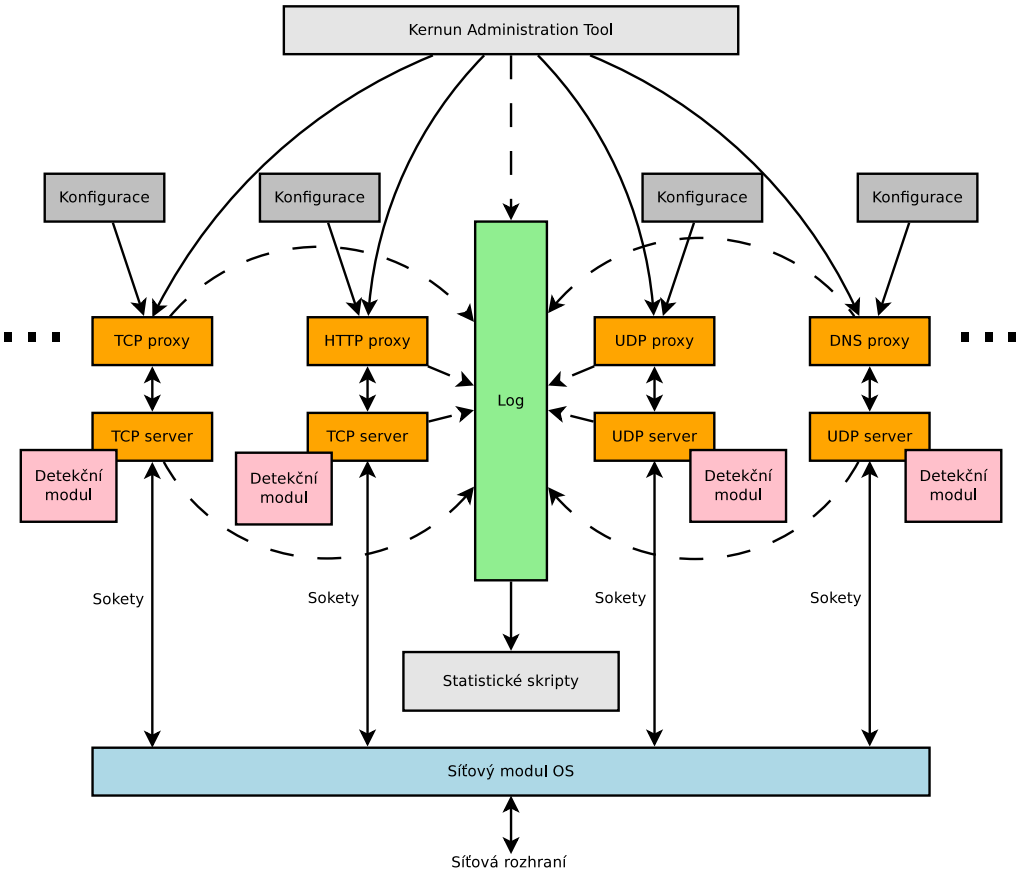


Obrázek 6.3: Návrh zakomponování klasifikačního modulu do produktu Kernun UTM.

různé webové aplikace takové, jaké jsme definovali dříve v této práci.

Při pohledu na současnou podobu implementace proxy a při úvaze o vhodnosti místa zakomponování detekčního modulu, kterou jsme provedli v předchozím textu, je nejvhodnější zasadit detekční modul na vhodné místo přímo do vykonávání kódu TCP a UDP serveru. Výhodou tohoto řešení je, že tuto část kódu sdílí všechny proxy, tedy okamžitě po zakomponování a úpravách v komunikaci mezi serverem a samotnou proxy je možné využít detekčního modulu ve všech dostupných proxy. Další výhodou je, že v takovém případě má detekční modul přímý přístup k přenášeným datům a navíc pro každou instanci proxy vznikne samotná instance detekčního modulu, což usnadňuje možnost jednoduchého škálování. V takovém případě je nutné, aby znalostní báze, na základě které probíhá detekce, byla sdílena na jednom místě mezi všemi těmito instancemi, aby se zabránilo zbytečnému plýtvání zdrojů. Detekční modul musí být konfiguračně povolován pro každou instanci proxy zvlášť, přičemž ve výchozím stavu je jeho použití zakázáno, aby zbytečně nedocházelo k nechtěnému degradování výkonu systému. Následné začlenění detekčního modulu do současného logovacího systému a úpravou odpovídajících skriptů se zajistí propagace výsledků detekce a jeho průběžný stav do generovaných statistik. Navrhnované úpravy do současné podoby implementace proxy a s ní spojených součástí vystihuje obrázek 6.4.

Takto navržené řešení však bohužel není vhodné realizovat pro námi navržený detekční algoritmus, který je založen na statistických informacích o síťových tocích, kde k vyhodnocení provozu potřebuje znát relativně přesný čas přijetí konkrétních paketů v síťovém toku. Jelikož jsou na úrovni proxy využívány klasické BSD sockety, kde metoda pro zpracování přijatého nebo odesílaného paketu je volána v pozdější fázi průchodu síťovým modulem operačního systému, čímž může dojít k ovlivnění přesnosti časové značky přijetí či odesílání



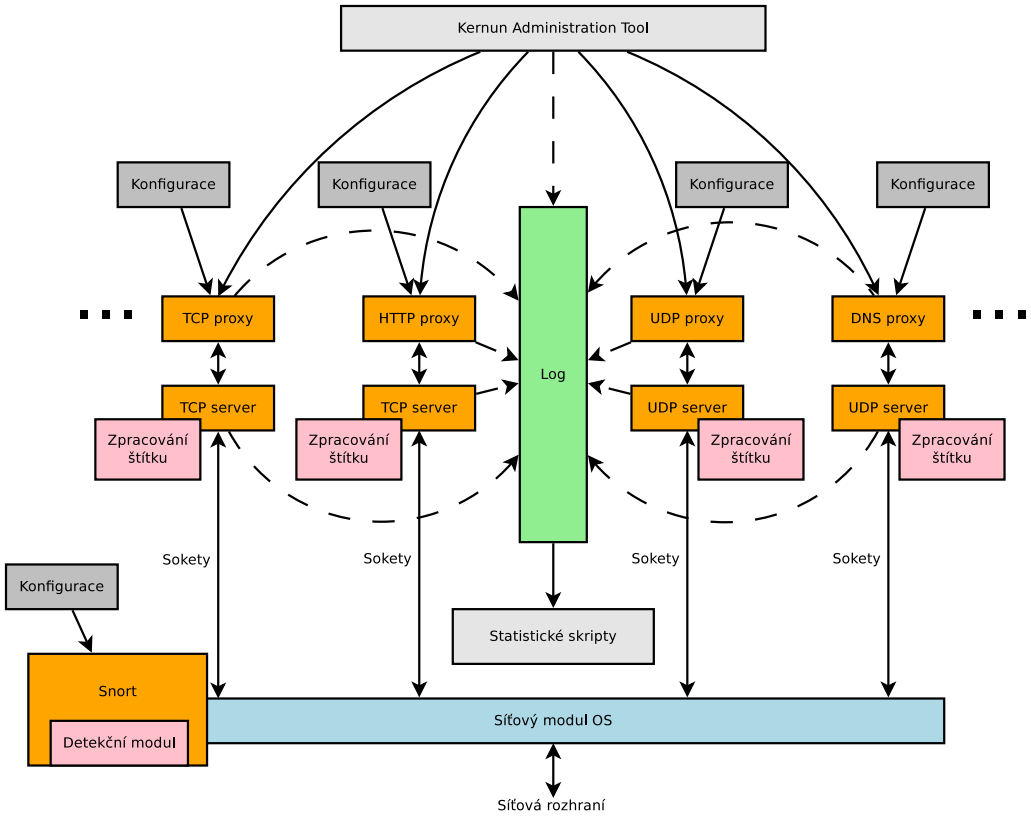
Obrázek 6.4: Návrh zakomponování klasifikačního modulu do architektury proxy.

paketu, tak tato data nejsou k dispozici. Zároveň na této úrovni nejsou přijímány pakety vrstvy síťového rozhraní, ale již pakety internetové vrstvy, takže přicházíme o část informace. Takovýto návrh je vhodný, pokud by detekční algoritmus byl založen pouze na detekci na základě přenášených dat a využíval především metod DPI. Takové informace jsou zde dostupné. Případné rozšíření o možnosti blokování vybraných aplikací je pomocí tohoto návrhu možné, jelikož detekční modul přímo v kódu proxy detekuje aplikaci a tato informace je posléze předána spolu s příslušným paketem, o kterém následně přímo proxy samotná rozhodne na základě konfigurace, zda-li bude tento paket propuštěn nebo zablokován.

6.2.2 Využití aplikace Snort s vlastní rozšiřující knihovnou

Produkt Kernun UTM v sobě již jako součást obsahuje aplikaci Snort, která slouží jako IDS, potažmo IPS systém. Funguje na základě sledování toku v síti a podle nahraných pravidel rozpoznává, zda-li se jedná o nebezpečnou komunikaci. Následně generuje odpovídající hlášení o problémech v síti. Pro jeho správnou funkčnost je nutné jej korektně nakonfigurovat (lze provádět přímo v grafickém rozhraní Kernun UTM) v závislosti na podobě sítě, kde je nasazován a pravidelně aktualizovat sadu pravidel, na jejichž základě se rozhoduje. Mimo psaní vlastních pravidel je možné pro Snort psát vlastní DAQ moduly, s rozšířenou

funkčností. Více jsem se o tom zmínil v kapitole 3.3.2.



Obrázek 6.5: Návrh zakomponování klasifikačního modulu s využitím aplikace Snort.

Díky možnosti rozšířit funkčnost Snortu o vlastní moduly by bylo možné vyrobit vlastní modul, využívající implementaci výše uvedeného algoritmu pro detekci aplikací, který by na příslušném zařízení prováděl analýzu procházejících paketů. Při úspěšné detekci by pak příslušné toky mohl označit tagem/štítkem, patřícím příslušné aplikaci, případně speciální značkou, která označuje, že k úspěšné detekci zatím nedošlo nebo se jedná o dosud neznámou aplikaci. Do modulu proxy serveru by pak bylo třeba přidat možnost tyto štítky číst a na základě konfigurace se k nim zachovat. Vzhledem k povaze aplikace Snort by implementace tohoto řešení umožňovala jednoduché rozšíření i o případnou detekci aplikací na základě obsahu přenášených dat — DPI. Zároveň samotná aplikace Snort umoňuje modifikovat a blokovat vybraný provoz, tedy s použitím vlastního modulu by bylo možné jednoduše rozšířit návrh i o blokování zvolených aplikací. Bylo by dokonce možné převést kompletní detekci aplikací a jejich blokování jen do samotného modulu aplikace Snort. Avšak nebylo by možné využívat rozšířených možností celé platformy, například pro řízení provozu pro konkrétního koncového uživatele (různým uživatelům bychom chtěli různě povolovat nebo monitorovat jaké síťové aplikace využívají na základě nastavené politiky). Informaci, kdo je cílovým uživatelem má totiž k dispozici až příslušná proxy. Proto by standardně provádění blokování mělo být řešeno až v rámci konkrétní proxy samotné. Tedy aplikace Snort a speciálně vyvinutý rozšiřující modul by prováděl pouze detekci aplikací a označování příslušných

paketů odpovídajícím štítkem, přičemž až v samotné proxy by se na základě této informace rozhodovalo, jak bude s paketem dále naloženo. Obrázek 6.5 graficky znázorňuje toto řešení.

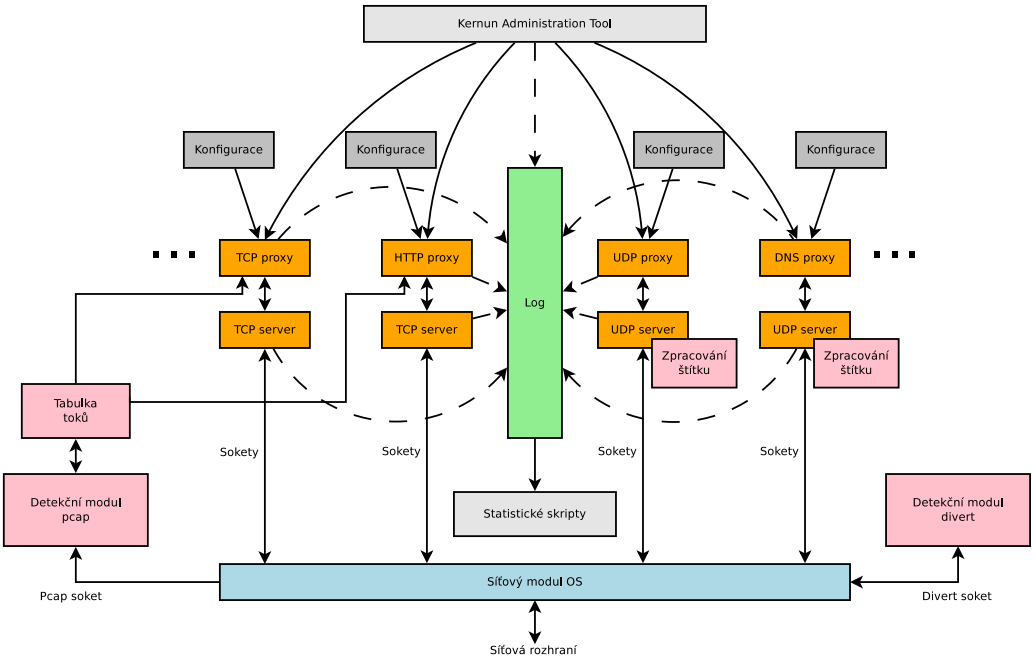
Tento návrh řešení poskytuje veškeré potřebné informace, které jsou nutné pro implementovaný detekční algoritmus. Vzhledem k tomu, že Snort využívá pro řízení síťového provozu divert sockety, které jsou umístěny níže v síťovém modulu OS, dovolují přesnější zachycení časové značky, kdy byl příslušný paket přijat nebo odeslán. Zároveň nabízí rozšiřitelnost o další metody, které by byly založeny na DPI, případně možnost rozšíření o blokování vybraných aplikací. Nevýhodou tak zůstává závislost na řešení od třetí strany, kdy se z ní využívá pouze mizivá podmnožina celkových jejich dostupných vlastností. To by mohlo být problémem při případném dalším škálování výkonu. Za určitých okolností bychom mohli chtít, aby na různých rozhraních pracovaly různé instance detekčních modulů. To by znamenalo nutnost spouštět více instancí samotné aplikace Snort s naším rozšiřujícím modulem. Rovněž by tato závislost nebyla vhodná v případě, kdy by bylo v budoucnu z jakéhokoliv důvodu rozhodnuto, že produkt Kernun UTM již nadále aplikaci Snort nebude obsahovat. V takovém případě by bylo třeba implementaci přepsat — v té chvili pravděpodobně do samostatného modulu, využívajícího přímo divert sockety.

6.2.3 Klasifikace jako samostatný modul

Jelikož zakomponování klasifikačního modulu přímo do kódu proxy serveru není díky absenci dostatečně přesné časové značky u čteného paketu možné, nabízí se alternativa tuto činnost převést do zvláštního modulu. Čtení by v tomto případě probíhalo na úrovni pcap socketu [46], případně divert socketu [47], které jsou pro navrhovanou implementaci vhodnější. Struktura datového typu `pcap_pkthdr`, který je používán při sledování provozu pomocí pcap soketů již přímo obsahuje časovou značku, určující čas přijetí příslušného paketu. U divert socketu je tento čas sice nutný získat ručně, ale díky tomu, že manipulace pomocí divert socketu se odehrává na nižší úrovni v řetězci síťového modulu operačního systému, je tento čas přesnější, než čas, získaný v případě poslouchání na klasickém BSD socketu.

Při využití pcap socketu by to pak v praxi mohlo vypadat tak, že v konfiguraci zařízení přibude další komponenta, která bude na vybraných síťových rozhraních číst pomocí pcap socketů příchozí a odchozí data a na základě nich provádět detekci. Výsledky pak bude zapisovat do tabulky toků, kterou bude udržovat každá instance komponenty zvlášť. Informace o tom, jaká aplikace konkrétní síťový tok generuje by si proxy musela vyzvedávat samostatně od správné instance detekčního modulu, který provádí detekci na odpovídajícím soketu. Pro komunikaci a dotazování se do tabulky toků na druh aplikace, která příslušný tok generuje, se nabízí možnost využití sdílené paměti, dedikovaného souboru nebo lokálního soketu. Případné rozšíření implementace o detekci nejen na základě statistické metody, ale i o detekci s využitím přenášených dat je i v případě tohoto návrhu možné, jelikož pcap sockety dovolují přístup k přenášeným datům přijatého paketu. Možnost blokování vybraných spojení na této úrovni možná sice není, jelikož pcap sockety toto neumožňují, ale je možná na úrovni proxy. Příslušná proxy totiž již ve chvíli zpracovávání paketu, díky možnosti přístupu do odpovídající tabulky toků, je schopna určit, jaké aplikaci příslušný paket náleží a na základě politiky rozhodnout, zda-li bude paket postoupen dále nebo bude zablokován. Návrh znázorňuje levá část obrázku 6.6.

Řešení pomocí divert socketů dovoluje nejen příslušný příchozí a odchozí paket číst, ale také jej upravovat, případně úplně zahodit. Předávání informací o detekované aplikaci by tedy v tomto případě mohlo být jednodušší než v předchozím případě, jelikož by stačilo přidat ke každému paketu tag nebo štítěk, který by identifikoval rozpoznanou aplikaci,



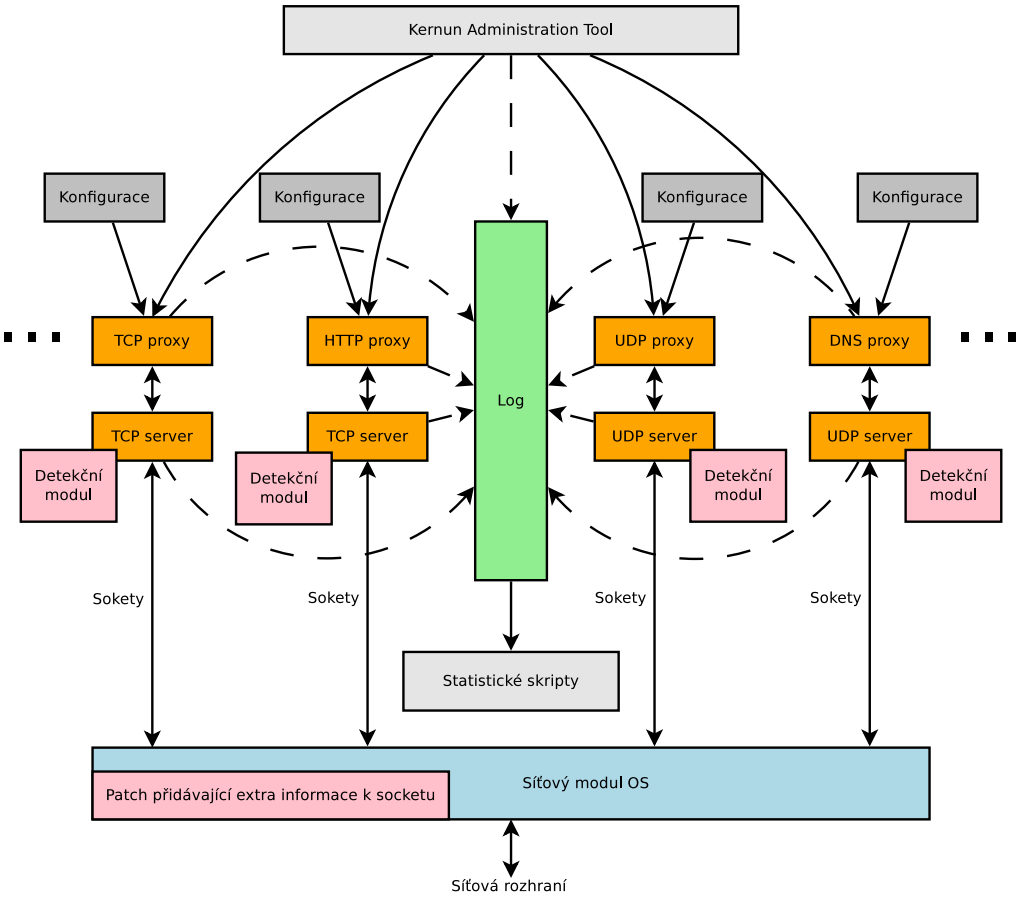
Obrázek 6.6: Návrh implementace klasifikační logiky jako samostatného modulu. Levá polovina obrázku znázorňuje návrh při použití pcap socketů, pravá polovina pak použití divert socketů.

případně speciální značku, která by označovala, že aplikace nebyla dosud rozpoznána — stejný způsob, jako byl navrhnutý v 6.2.2. Tento návrh řešení je alternativou pro řešení pomocí aplikace Snort, jelikož ta samotná využívá k řízení síťového provozu na zařízení rovněž divert sockety. Výhodou oproti němu je v tomto případě nezávislost na samotné aplikaci Snort. Případné rozšíření tohoto řešení o detekci na základě obsahu paketů je podobně jako v případě využití pcap socketů stejně jednoduché. Blokování je samozřejmě nejvhodnější opět řešit až na úrovni konkrétních proxy, avšak tento mechanizmus dovoluje provést blokování již na úrovni samotného detekčního modulu, ať už by důvod k takovému kroku byl jakýkoliv. Další nezbytností při tomto řešení je nutnost zapnutí systémového firewallu ipfw (případně paketového filtru pf) s odpovídajícími pravidly, směrujícími provoz na divert sockety. Návrh znázorňuje pravá část obrázku 6.6.

Výhodou řešení pomocí samostatného modulu je poměrně snadná možnost škálování výkonnosti klasifikace a oproti návrhu s využitím aplikace Snort nižší režie. Při výběru mezi pcap a divert sockety se jeví jako výhodnější využít divert sockety, díky kterým by odpadla nutnost komunikace příslušné proxy s odpovídajícím detekčním modulem, který by jí musel poskytovat aktuální informace z tabulky síťových toků, což znamená režii navíc. Naproti tomu řešení pomocí divert socketů, kdy je ke každému paketu přímo přiřazen štítek, obsahující výsledek detekčního modulu, tuto režii odbourává a zároveň odpadá nutnost propagovat výše tabulku síťových toků a její obsah.

6.2.4 Úprava síťového stacku jádra systému FreeBSD

Jak již bylo uvedeno v sekci 6, produkt Kernun UTM je vyvíjen nad operačním systémem FreeBSD. Díky jeho otevřenosti a dostupným zdrojovým kódům se jako další možnost nabízí provedení úpravy zpracování příchozích i odchozích paketů přímo v jádře. Ke každému socketu se připojí potřebné informace, které jsou potřeba pro použití v detekčním algoritmu. Každý socket tedy bude udržovat tabulku se síťovými toky, které přes něj procházejí. Při každém přijetí nebo odeslání paketu je třeba informace v tabulce toků aktualizovat. Tyto informace je pak nutné poskytnout do uživatelské části systému tak, aby je byla schopná příslušná proxy přečíst při svém samotném zpracovávání paketu a na základě toho se pak rozhodnout, co dále provede. Tento návrh řešení vychází z prvního návrhu popsaném zde 6.2.1 a odstraňuje jeho nedostatky, kvůli kterým jej není vhodné použít pro vybraný detekční algoritmus.



Obrázek 6.7: Návrh implementace detekčního modulu s pomocí úpravy jádra kernelu.

Detekční modul se tak v tomto případě rozdělí na dvě části. První část tvoří samotná úprava zpracování přijímaných i odesílaných paketů v jádře systému, spolu s implementací podpory pro čtení těchto informací z uživatelské části systému. Druhou částí je pak úprava v proxy serveru, kdy dochází při každém zpracovávaném paketu ke zjištění, k jakému toku

daný paket náleží, z příslušného socketu získání informací o daném toku a zavolání modulu, který na základě těchto informací detekuje aplikaci, která daný síťový provoz generuje. Zde je třeba zdůraznit, že samotné výpočty, týkající se detekce aplikace a další možné složitější algoritmy není vhodné přidávat do samotného jádra (čili do první zmíněné části). Mohlo by totiž dojít k citelné degradaci výkonu zpracovávání síťových dat samotným jádrem systému.

Ohledně případného rozšíření detekce i o detekci na základě přenášených dat v samotných tocích, je možné uvažovat i v tomto navrhovaném řešení. V části, kdy dochází ke zpracování paketů na straně proxy, je přístup k datům příslušného paketu, které pro takovou detekci mohou být využity. Koncept blokování vybraného provozu tento přístup také umožňuje, pouze však v režii samotné proxy, což je ale dostatečné.

Jelikož v současné verzi produktu již existují úpravy zdrojových kódů operačního systému FreeBSD a Kernun UTM je provozován nad modifikovaným, speciálně přeloženým jádrem s upraveným konfiguračním souborem, není z tohoto pohledu žádný problém rozšířit tento patch o další změny, které by zahrnovaly implementaci zmiňovaného návrhu. Návrh řešení pomocí úpravy v jádře poskytuje všechny potřebné informace, které jsou pro algoritmus detekce aplikací potřebné. K dispozici je také poměrně přesný čas zachycení konkrétního paketu. Také díky přesunu detekčního modulu do základního kódu proxy je usnadněno škálování, jelikož s každou novou instancí proxy se bude vázat nová instance detekčního modulu. Nevýhodou může být to, že příslušné extra informace jsou k socketu přidány zvlášť přímo v síťovém modulu jádra, mimo vlastní implementaci detekčního algoritmu. To příliš nekoresponduje s vhodným způsobem návrhu architektury pro případ, kdy by bylo plánováno využití kombinace více různých způsobů detekce, případně výběru mezi nimi. V případě budoucí implementace detekce aplikací i pomocí jiných metod by tak tato úprava mohla zbytečně zavádět režii do zpracování přijímaných a odesílaných paketů. Proto by bylo vhodné mít možnost ji jednoduše konfiguračně vypínat a zapínat podle potřeby.

Tuto variantu jsem zvolil jako nejvhodnější řešení a v následujících odstavcích se o ní více rozepíšu.

6.3 Vybraný způsob implementace

Po zvážení výhod a nevýhod jednotlivých variant jsem pro provedení implementace zvolil poslední zmíněný návrh — Úprava síťového stacku jádra systému FreeBSD, popsaný v sekci 6.2.4. Tato varianta nabízí začlenění detekčního modulu do kódu proxy, aniž by bylo nutné vyrábět novou komponentu, jejíž stav by bylo třeba samostatně sledovat. Z uživatelského pohledu se tak bude přidání funkčnosti detekce aplikací jevit jako další konfigurační položka příslušné proxy. Detekční modul tedy bude konfigurovatelný pomocí standardní konfigurace Kernun CML v rámci konfigurace samotné proxy. Vhodná by mohla být i jedna globální sekce, udávající konfiguraci společně pro všechny proxy. Každá proxy bude mít vlastní instanci detekčního modulu, což zjednodušuje škálování výkonu. Druhou částí integrace detekčního modulu je nutná úprava síťového modulu operačního systému FreeBSD. Úprava v jádře bude obsahovat pouze nejnutnější změny, které doplní potřebné údaje, jež potřebuje detekční algoritmus ke své činnosti.

Dalšími argumenty pro výběr této varianty byl vhodný potenciál pro další rozšiřování a úpravy. Zároveň lze toto řešení vhodně modularizovat a začlenit do současné architektury systému Kernun UTM.

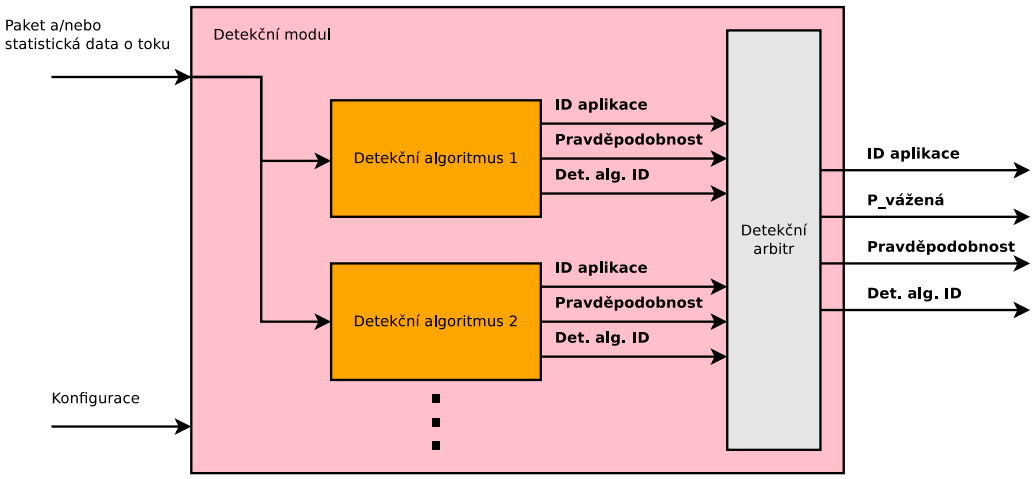
6.3.1 Navrhovaná architektura

Obrázek 6.8 a sekce 6.2.4 zobrazuje a popisuje architekturu způsobu integrace detekčního modulu do produktu Kernun UTM. Jedná se vlastně o modifikovanou verzi návrhu na obrázku 6.4, popsaného v sekci 6.2.1. Návrh se skládá ze třech základních částí:

1. Vytvoření detekčního modulu, který je instanciován v kódu proxy a který bude využívat extra přidané informace díky modifikaci jádra. V rámci této implementace musí být zahrnuta i podpora pro počáteční inicializaci detekčního modulu.
2. Úprava síťového modulu jádra systému FreeBSD takovým způsobem, aby se zajistily potřebné informace pro detekční algoritmus ve správný okamžik.
3. Úprava konfigurace pro současnou komponentu proxy tak, aby reflektovala změny a přidanou funkčnost.

6.3.2 Architektura detekčního modulu

Detekční modul a jeho podoba reflektuje vybraný návrh začlenění do současné architektury produktu Kernun UTM. Zároveň je však třeba uvažovat o jeho dalším možném rozšiřování a flexibilitě ke změnám tak zásadním, jako například výměně samotného detekčního algoritmu. Z tohoto důvodu návrh počítá s modularitou jednotlivých částí a jejich možném jedoduchém nahrazení, případně rozšíření. Návrh architektury zobrazuje obrázek 6.8.



Obrázek 6.8: Návrh architektury detekčního modulu.

Před samotným začátkem používání detekčního modulu je třeba jej správně nakonfigurovat. V rámci konfigurace je nutné vybrat požadovaný typ detekce respektive zvolené detekční algoritmy, stejně jako inicializovat přednaučené údaje, ze kterých pak jednotlivé metody vycházejí při své činnosti. Vstupem detekčního modulu jsou informace o příchozím paketu a odpovídajícím síťovém toku. Jelikož návrh zohledňuje jak možnost začlenit detekční algoritmy založené na statistických vlastnostech probíhajících toků, tak i na metodách, využívající k detekci samotný obsah přenášených dat, je třeba mít možnost dodat

oba typy informací. Samotné jádro tvoří moduly s detekčními algoritmy, které implementují samotné metody detekce. Výstupem každého z těchto modulů jsou vektory s předpokládanými aplikacemi a odpovídající pravděpodobností správnosti detekce. Tedy výstupem jsou trojice [id_app; probability; id_algorithm], seřazené právě podle míry pravděpodobnosti od nejvyšší k nejnižší. Jednotlivé prvky mají význam následující: *id_app* — jednoznačný identifikátor aplikace nebo třídy provozu, kterému příslušný paket náleží; *probability* — pravděpodobnost, s jakou tento paket náleží aplikaci, specifikovanou pomocí položky *id_app*; *id_algorithm* — jednoznačný identifikátor algoritmu, který byl pro tuto konkrétní detekci použit.

Právě z důvodu začlenění podpory použití v detekčním modulu více detekčních algoritmů, je třeba přidat další prvek, který bude vybírat mezi všemi výsledky na základě zvolené strategie. Proto je třeba mezi výstup jednotlivých detekčních algoritmů a výstup samotného detekčního modulu vložit blok detekčního arbitra. Ten umožňuje na základě zvolené konfigurace různě řadit výstupy z jednotlivých detekčnícho algoritmů. K dispozici jsou následující možnosti výběru výsledku:

1. Respektování výsledků jednotlivých detekčních algoritmů přímo bez dalších zásahů.
2. Váhové úpravy výsledků podle příslušných detekčních algoritmů.
3. Heuristika založená na kombinaci výsledků z různých detekčních algoritmů.

První možnost mluví sama za sebe. Jedná se o způsob, kdy jsou výsledky ze všech detekčních algoritmů jednoduše seřazeny podle pravděpodobností správné detekce. Celkovým výsledkem je tedy trojice [id_app; probability; id_algorithm] s největší hodnotou položky *probability*.

Druhá možnost přidává schopnost ovlivnit výsledky detekce preferencemi vybraných algoritmů tím, že každému z algoritmů přiřadíme určitou váhu. Samotné hodnoty pravděpodobnosti jsou pak násobeny tímto váhovým koeficientem a teprve následně seřazeny stejným způsobem, jako tomu bylo u první možnosti.Tato varianta dovoluje také specifikovat množinu detekčních algoritmů, které budeme opravdu považovat za relevantní a zbylým nastavit prioritu rovnou nule. To znamená, že i když jsou tyto další algoritmy nakonfigurovány a fungují, nejsou při výsledném rozhodování brány v potaz, jelikož výsledná pravděpodobnost jejich detekce je násobena nulovým koeficientem. Jejich výstup tak může sloužit například pro sledování výsledků, aniž by ovlivnily aktuální konfiguraci detekce. To se může hodit v případech testování nového nebo upraveného detekčního algoritmu.

Třetí možností je návrh využití výsledků z více různých detekčních algoritmů implementováním heuristik, které by byly schopné zpřesnit detekci a zvýšit pravděpodobnost správného výsledku. Tento způsob je prozatím návrh, který by bylo v praxi třeba ověřit. Každopádně potenciál pro zvýšení úspěšnosti zde jistě je. Například porovnáním výsledků různých detekčních algoritmů mezi sebou pro jednu aplikaci se dá lépe určit pravděpodobnost správné detekce a podobně.

Celkovým výstupem detekčního modulu bude tedy vektor, obsahující čtveřici informací [id_app; probability_weight; probability; id_algorithm]. Oproti původní trojici informací, popsaných výše, obsahuje navíc informaci *probability_weight*, což je váhově ovlivněná pravděpodobnost náležitosti příslušného paketu k příslušné aplikaci specifikovanou *id_app*. Váhově ovlivněna znamená podle přednastavených preferencí ke konkrétním detekčním algoritmům a jejich vah. Lépe je to možné vidět na obrázku 6.8.

6.3.3 Úpravy v síťovém modulu FreeBSD

Použitý detekční algoritmus v této práci je založen na detekci na základě statistických informací o síťových tocích. Obecně jsou zde tyto statistické metody preferovány díky jejich nezávislosti na přenášených datech. Pro správnou funkčnost potřebují mít přístup k informacím o velikostech samotných paketů i poměrně přesným časovým značkám, identifikující přijetí paketu na zařízení. Implementace detekčního modulu je provedena zakomponováním do kódu samotných proxy, kde je zjišťování přesného času přijetí paketu ovlivněno několika úrovněmi průchodu přes síťový modul operačního systému i přes část kódu samotných proxy. Také na této úrovni není přístup k nižší síťové vrstvě přijatých paketů, jejíž informace nelze na této úrovni již získat. Je tedy nutné nalézt jiný způsob, jak tyto informace detekčnímu modulu v uživatelské části systému zprostředkovat. Zvolenou variantou se stala úprava síťového modulu jádra FreeBSD.

Pro splnění požadavků, které jsou třeba dodat do uživatelské části, aby detekční algoritmus mohl správně fungovat, je nutné dodat správné informace o sledovaných síťových tocích. Tyto jsou sledovány na úrovni síťové vrstvy. Úprava síťového modulu tedy spočívá v modifikaci kódu, který zpracovává příjem ethernetových rámců na síti pro protokoly IP a IPv6. Jiné protokoly nebyly brány v potaz. Přidaný kód provádí při každém přijatém paketu úpravu tabulky síťových toků, kterou si udržuje, na základě právě přijatého paketu. Tabulka je sestavována na základě pětice: zdrojová IP adresa, cílová IP adresa, zdrojový aplikační port, cílový aplikační port a protokol transportní vrstvy (předpoklad je TCP nebo UDP). Tato tabulka toků je pak dostupná příslušné proxy, která při zpracovávání aktuálních dat ze síťového soketu vyhledá v tabulce odpovídající záznam síťového toku, který předá detekčnímu modulu, v rámci něhož jsou spuštěny nakonfigurované detekční algoritmy. Na základě těchto dat detekční algoritmus provede detekci a vrátí proxy výsledek v podobě vektorů s nejpravděpodobnějšími aplikacemi.

Samotná úprava zpracování přijatých ethernetových rámců by měla být provedena v souboru *sys/net/if_ethersubr.c*, konkrétně ve funkci *ether_demux*(). Právě zde jsou zpracovávány přijímané ethernetové rámce a přidání dalšího kódu by mělo být jednoduché. V tomto kroku by mělo docházet k aktualizaci hodnot v udržované tabulce síťových toků. Je taté třeba přidat novou položku do struktury *if_data* v souboru *sys/net/if.h*. Tato struktura obsahuje informace o příslušném rozhraní a je dostupná z uživatelského prostoru — tedy z kódu proxy. Položka, která by měla být přidána je ukazatel na instanci tabulky toků, které náleží příslušnému síťovému rozhraní. Současně s touto úpravou je nutné zajistit inicializaci této položky při vytváření rozhraní a uvolnění paměti, při rušení rozhraní. To by mělo být provedeno ve funkcích *if_alloc*(), kde by měla být položka inicializována a *if_free_internal*(), kde by mělo dojít k uvolnění paměti a zrušení tabulky síťových toků pro příslušné rozhraní. Datová struktura pro tabulku síťových toků, kterou bude třeba udržovat, by měla být hash-tabulka, kde jednotlivé klíče pro data jsou údaje o zdrojové a cílové IP adrese, zdrojovém a cílovém aplikačním portu a typu aplikačního protokolu. Hash-tabulka umožňuje velice rychlé vyhledávání, takže je vhodnější než jiné datové struktury. Každý záznam pak obsahuje odkaz na datovou strukturu, reprezentující údaje o konkrétním síťovém toku. Jsou zde vedeny sledované statistické informace, nutné pro výpočet detekčního algoritmu, časy přijetí úvodního i posledního paketu a podobně. Každý síťový tok má dobu životnosti, po kterou když nepřijde žádný paket, tak je odpovídající tok z tabulky vymazán.

6.3.4 Konfigurace

V rámci konfigurace je třeba minimálně zajistit, aby bylo možné modul povolit nebo zakázat. V případech, kdy jeho potřeba není nutná bude vhodné jej vypnout, aby zbytečně nedocházelo k režii, vedoucí k plýtvání systémovými prostředky a případné degradaci výkonu celého zařízení. Návrh počítá s možností jednoduchého začlenění více detekčních algoritmů. Proto by úprava konfigurace měla zohledňovat i možnost volby detekčního algoritmu. Zároveň s tím je třeba zahrnout i konfigurační možnosti pro modul detekčního arbitru tak, jak byl navržený v 6.3.2. Tedy je zapotřebí mít možnost nastavení výběru typu výsledku a váhy jednotlivým použitým detekčním algoritmům. Dále je třeba mít možnost zvolit předpřipravenou heuristiku, která spoléhá na výstup výsledků z více algoritmů současně.

6.4 Stav cílové implementace v produktu Kernun UTM

Nad rámec rozsahu této práce jsem začal pracovat nad zapracováním navrhnutých změn do produktu Kernun UTM. I když je tato část stále spíše na začátku a nejsou zatím dostupné žádné reálné výsledky, přesto přinesla několik podstatných postřehů, které jsou zmíněny dále v sekci 7.3.

6.4.1 Zvolený jazyk a technologie

Pro implementaci cílového řešení jsem, na rozdíl od implementace proof-of-concept prototypu, nezvolil skriptovací jazyk Perl, ale kombinaci jazyka C a C++. Důvody k tomuto rozhodnutí jsou především v požadavcích na vysoký výkon výsledné implementace, které tyto jazyky jistě oproti jazyku Perl nabízejí, ale také to, že navržené řešení počítá se zakomponováním přímo do kódu samotných proxy, které jsou napsány v jazyce C a úpravách jádra operačního systému, který je rovněž psán v tomto jazyce. Volbu C++ jsem pak použil proto, jelikož nabízí možnosti objektového programování a obsahuje větší množství standardních knihoven s podporou předpřipravených šablon datových struktur.

6.4.2 Reálná implementace

Přestože cílová implementace do produktu Kernun UTM vychází z návrhu, představeného v předchozích ostavcích, tak v první iteraci provádím implementaci trochu zjednodušeně, aby se urychlil vývoj a mohlo se dříve testovat a ověřovat navrhnuté řešení. V úvodní iteraci nebude implementován modul rozhodovacího arbitra — respektive bude řešen jen jako jednoduchá prázdná skříňka, která pouze přeposílá vstupní informace na výstup. Podobně v první iteraci nepočítám s dalšími alternativními detekčními algoritmy, vyjma toho, který byl popsán v této práci.

Dokončení prací na implementaci tohoto návrhu je plánováno v co nejbližším horizontu, aby bylo možné jej zpřístupnit pro interní testování v rámci společnosti TNS. V průběhu tohoto testování by se měly odladit všechny případné nalezené nedostatky a rovněž by mělo docházet k postupné optimalizaci jak výkonnosti, tak i spolehlivosti samotné detekce. V další fázi by se měla implementace dokončit podle návrhu a po dalším kole testování teprve bude možné považovat implementaci za dostatečně zralou a bude možné ji uvolnit jako veřejně přístupnou funkcionalitu.

Kapitola 7

Zhodnocení dosažených výsledků provedenou implementací

V předchozí kapitole jsem popsal návrh zakomponování funkcionality detekce aplikací do produktu Kernun UTM. Podle tohoto návrhu byly již započaty úpravy produktu. Zatím jsou však ve stavu, kdy není možné provést testování s reálnými výsledky. V průběhu práce jsem však implementoval prototyp, nad kterým jsem tyto testy provedl. V této části se zaměřím na zhodnocení implementovaných technik a zamyslím se nad možnými dalšími rozšířeními a možnostmi navázání na současný stav práce. Zhodnocení možnosti o budoucí rozšíření i o blokování vybraných aplikací, jak bylo zmiňováno a uvažováno již v začátcích návrhu, provedu v samostatné sekci.

7.1 Implementace a výsledky nad reálným síťovým provozem

V průběhu práce jsem implementoval prototyp detekčního modulu, který provádí klasifikaci provozu nad zachyceným reálným síťovým provozem. Více informací o provádění a výsledcích těchto testů lze najít v sekci 5.4.2. Úspěšnost detekce v těchto testech nebyla příliš vysoká, avšak bylo vidět, že detekce nějakým způsobem funguje a že případným vylepšením hodnotící funkce, případně rozšířením trénovacích dat by se mohla úspěšnost dále zvýšit.

Dalším krokem k ověření navrhnutého řešení je test nad reálným síťovým provozem za běhu. Jedná se tedy o test, kdy detekční modul již poslouchá přímo na síťovém rozhraní testovacího počítače, čte procházející provoz a provádí detekci. K tomuto účelu jsem upravil implementovaný prototyp tak, aby dokázal právě poslouchat na zvoleném síťovém rozhraní a průběžně na výstup vypisoval výsledky detekce. Testování probíhalo na stejném hardware, jako testování, prováděné v sekci 5.4.2 a byla použita i stejná trénovací data. Po spuštění detekčního modulu jsem generoval různý provoz na příslušném síťovém rozhraní a pozoroval, jaké výsledky detekční modul průběžně vypisuje. Bohužel výsledky měření nejsou příliš přesvědčivé. Poměrně slušně funguje detekce webového provozu — HTTPS, HTTP. Při spuštěném detekčním modulu a prohlížení webových stránek jsem dostával relativně odpovídající výsledky s tím, že zde byly i nějaké false positives v podobě například aplikace Facebook nebo Google dokumenty. Vzhledem ale k tomu, že se též jedná o webové aplikace, tak toto false positive není příliš negativní. Při zkoušení detekce SSH připojení se již projevila nezrálost detekčního modulu. Výsledkem bylo především false positives v podobě HTTP a HTTPS komunikace. Přesto však nakonec detekční modul SSH aplikaci v provozu

poznal. Podobných nepřesvědčivých výsledků se mi dostalo i u dalších aplikací.

Po provedení otestování nad reálným živým provozem lze konstatovat, že úspěšnost detekce ani zdaleka nedosahuje takových výsledků, jako v případě testování nad připravenou sadou dat (ikdyž také z reálného provozu), použitou při testech v sekci 5.4.2. Na vrub to lze přičíst jak nezrálosti samotné implementace detekčního algoritmu, tak pravděpodobně nízkému množství trénovacích dat, které pro úspěšnou detekci nad skutečně živým provozem bude nutné ještě podstatně navýšit.

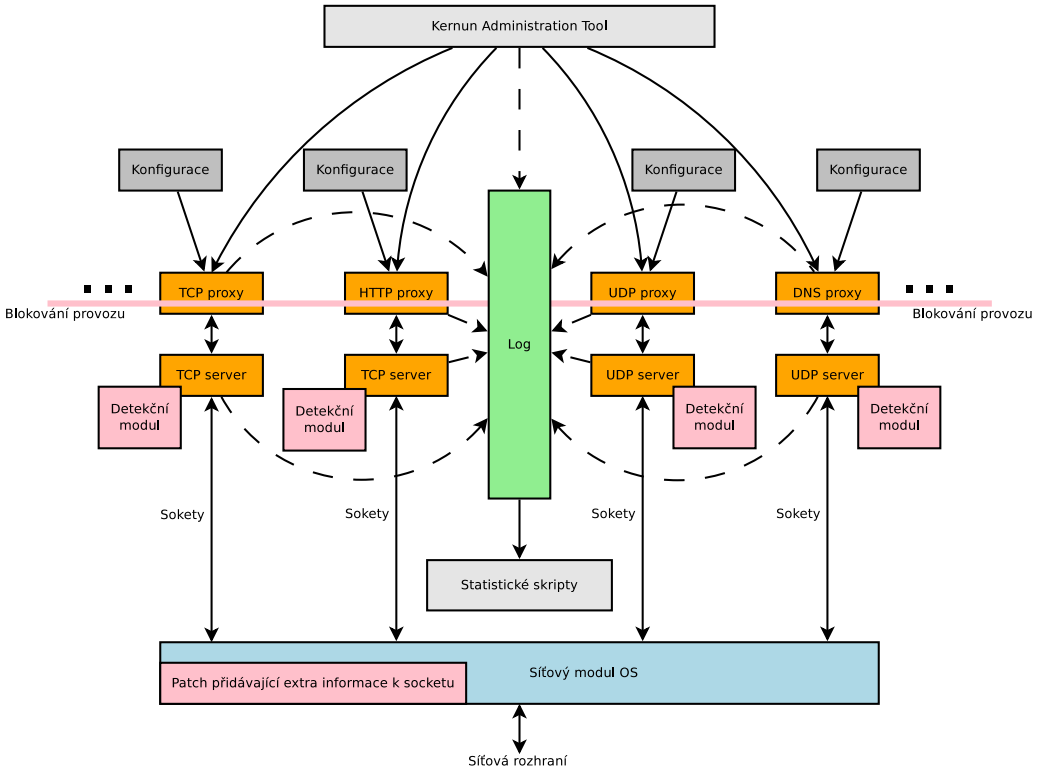
7.2 Rozšíření o blokování vybraných aplikací

Kromě samotné detekce aplikací bylo již od počátku záměrem provést návrh řešení takovým způsobem, aby bylo možné rozšířit výsledné řešení do budoucna o blokování vybraných aplikací, na základě této detekce. Díky tomuto vybranému řešení je možné tohoto cíle dosáhnout několika dalšími úpravami do společných kódů proxy, kdy blokování bude probíhat na základě výsledků detekce, která probíhá na úrovi níže, před zpracováním paketu samotnou specifickou funkcí proxy. Dále je třeba rozšířit konfiguraci o možnost specifikovat politiku, která příslušným účastníkům definuje různé úrovně a povolení používaných aplikací. Konfigurace by tedy mohla být navržena tak, že do současné konfigurace proxy přibyde další sekce, která takovou politiku určuje. Dále může vzniknout globální sekce, která by určovala výchozí politiku, společnou pro všechny instance různých proxy a která by byla platná v případě, že by pro konkrétní aplikaci nebylo nalezeno pravidlo v lokální části konfigurace proxy.

Spolehlivost blokování samozřejmě závisí na rychlosti a spolehlivosti samotné detekce. Především v případě námi vybrané metody, založené na statistických informacích komunikujících síťových toků, trvá nějakou chvíli, než je příslušná aplikace detekována, tudíž může dojít k tomu, že určité množství počáteční komunikace v pořádku projde až ke klientovi, přestože příslušná aplikace není v seznamu povolených aplikací. Je možné dále zlepšovat implementovaný algoritmus, ale principielně v tomto případě není možné dosáhnout úplné a stoprocentní blokace v nejzasším začátku komunikace přísušného síťového toku. Tuto nevýhodu poměrně zásadně odstraňují algoritmy založené na detekci podle samotného obsahu přenášených dat, tedy využívající technik DPI. Vzhledem k modulárnímu návrhu mého řešení však může být do budoucna zváženo zahrnutí takového/takových algoritmů a jejich testování v reálném provozu a případně nahrazení současného. Zároveň však může být využito kombinování těchto algoritmů dohromady, kdy například v úvodní části komunikace bude brán zřetel především na algoritmy zkoumající obsah přenášených dat a teprve později na algoritmy, založené na statistických datech síťového toku.

Samozřejmou součástí návrhu blokování musí být logovací výstupy, ze kterých je možné následně generovat různé grafy, užitečné jak pro administrátora zařízení, tak například i pro vedení příslušné organizace. Toto by se řešilo stejným způsobem, jako se již nyní děje u proxy a dalších komponent v zařízení. Tedy pomocí logování hlášek na konkrétní místo, odkud pak odpovídající skripty pravidelně extrahují informace, ukládají do databáze a následně generují grafické výstupy.

Celkový pohled na navrhované řešení popisuje obrázek 7.1. Ten znázorňuje jak vybranou architekturu detekce provozu, tak i naznačení úrovně, na které by mělo docházet k rozhodování o povolování procházejícího provozu. Důvod, proč by toto rozhodování nemělo být umístěno níže — například ve společném kódu proxy „tcpserver“ a „udpserver“ — je takový, že proxy udržuje stav spojení a především v TCP komunikaci je třeba v případě zablokování provozu příslušně upravit kontext a korektně ukončit proces zpracovávající toto sezení.



Obrázek 7.1: Návrh implementace blokování, založeném na implementovaném řešení. Obrázek vyznačuje úroveň, na které by docházelo k blokování provozu.

7.3 Potenciál pro další rozšíření a zlepšení

Implementovaný prototyp detekčního modulu, který jsem v rámci této práce vyrobil, posloužil jako příprava před přistoupením k implementaci do cílového produktu. Přesto však ještě existuje mnoho možností, jak jej vylepšit, zefektivnit a zrychlit.

V první řadě je problémem rychlost. Přestože se jedná o prototyp, napsaný v interpretovaným jazyce Perl, dá se částečně jeho pomalost pochopit. Implementace v jazyce C/C++ poskytne podstatně větší výkon. Avšak ruku v ruce s tím je třeba i vylepšit výpočet hodnotící funkce, aby se zvedla úspěšnost detekce, která nyní v některých aplikacích zdaleka nedosahuje požadovaných výsledků. Dá se říci, že oba zmíněné nedostatky tkví hlavně právě v implementaci samotné hodnotící funkce. Jelikož se jedná o stěžejní část algoritmu, která je v rámci výpočtů volána velice často, je její funkčnost a rychlost provádění pro celkový výkon algoritmu kritická. Možnosti ve vylepšení hodnotící funkce vidím ve dvou věcech:

Optimalizace výpočtů prováděných v rámci hodnotící funkce. Zde má implementace ještě jistě spoustu volného místa. V případě přepisu do cílového jazyka C/C++ lze využít i bitové operace.

Zvýšení úspěšnosti výsledků hodnotící funkce. Toho lze dosáhnout například přidáním sledovaných ukazatelů síťových toků. Nyní slouží k určovávní náležitosti toků hodnoty z celkem 15 ukazatelů u každého toku. Přidáním dalšího ukazatele by se mohla zvýšit

úspěšnost, ale zároveň se zvýší složitost výpočtu. Proto je třeba takovéto změny provádět opatrně. Zároveň ne všechny ukazatele jsou vhodné pro sledování. Příkladem pro možnou úpravu je postup aktuálního sledování shluků — pakety, mezi kterými není mezera větší než 1 vteřina. Stálo by za vyzkoušení namísto konstantní 1 vteřiny počítat vhodně průměr mezipaketových mezer pro každý tok a na základě této hodnoty měřit shluky.

To byly návrhy, týkající se optimalizace hodnotící funkce. Pro zvýšení výkonnosti se však nabízí i podpora práce ve více vláknech, případně ve více procesech. Toto řešení nedává příliš smysl, pokud je využíván pouze jeden detekční algoritmus. Vzhledem k podobě návrhu, kdy detekční modul má být instanciován individuálně pro každou proxy zvlášť se nezdá, že by toto řešení mohlo přinést další užitek. Je třeba brát v potaz i přidanou režii, která nutně vyvstane pro řízení různých vláken/procesů. Proto bych tuto možnost bral až jako krajní, při opravdové nedostatečné rychlosti výsledného řešení. Ovšem v případě využívání více detekčních algoritmů, by každý z nich mohl běžet nezávisle ve svém vlákně, potažmo procesu. To by mohlo efektivně zparalelizovat příslušné výpočty a režie přidaná v tomto případě pro řízení těchto vláken/procesů by nemusela hrát tak velkou roli.

Dalším bodem, který by mohl zvýšit rychlost detekce je přidání různých heuristik. Například v případě hledání nejbližšího možného odpovídajícího toku by se dala udržovat nějaká průměrná hodnota dosažených vzdáleností a na základě ní ve vhodnou chvíli ukončit prohledávání a jako nejbližší označit právě ten tok, který by prolomil tuto hranici. Tato optimalizace může být dvojsečná, jelikož záleží na způsobu výpočtu a aktualizace hodnoty této hranice. Na druhou stranu ale může zvýšit rychlost detekce, jelikož nebude nutné procházet celou sadu dat. Nutno podotknout, že tento problém by měl z velké části odstraňovat právě použitý hybridní algoritmus, především část, která využívá K-Means.

Cílového řešení se týká také podpora hash-tabulek v jádře operačního systému FreeBSD. Ta v současné chvíli není ideální, jelikož při inicializaci hash-tabulky je nutné znát její celkovou velikost. To značně komplikuje původní návrh, kdy v rámci úprav jádra by se v jaderném modulu měla udržovat instance tabulky toků pro každé síťové ethernetové rozhraní, kde síťové toky různě dynamicky vznikají a zanikají. Nabízí se implementace vlastní jednoduché knihovny do jádra systému, která by implementovala operace s hash-tabulkou lépe podle našich představ.

Úspěšnost detekce na základě těchto statistických dat může poměrně lehce ovlivnit výkyvy v kvalitě připojení a komunikace na síťové lince. Je to z toho důvodu, že v takovém případě jsou ovlivněny časy přenosu, časy přijímání paketů, některé pakety jsou několikrát přeposílány a podobně. To má za následek ovlivnění hodnot sledovaných statistických ukazatelů a nutně degraduje kvalitu takové detekce. Nabízí se otázka, zda-li by nebylo možné této nevýhody využít ve vlastní prospěch. Budeme-li předpokládat, že si můžeme dovolit provádět detekci aplikací mírně invazivním způsobem, mohli bychom v rámci průběhu detekce zkoušet ovlivňovat uměle a podle předem určených pravidel kvalitu linky, případně její propustnost a sledovat, jak se komunikující aplikace chová. Je otázka, zda-li by se daly najít nějaké vzory různých chování pro různé aplikace. Může to být však námět pro další testování do budoucna. Samozřejmě se jedná o invazivní způsob detekce, jelikož aktivně zasahujeme do spojení, avšak je jej třeba provádět tak, aby uživatel nebyl zbytečně frustrován nefunkčností síťového připojení. Zároveň tento návrh má velkou nevýhodu, že takovýto způsob detekce by byl časově příliš náročný. Alespoň ve smyslu potřebného počtu přenesených paketů a provedené komunikace, než by bylo možné dojít k nějakým rozumným výsledkům.

Kapitola 8

Závěr

V rámci této práce jsem provedl nastudování současného stavu detekce síťových aplikací a klasifikace síťového provozu. Seznámil jsem se s několika algoritmy detekce aplikací a se souvisejícími trendy budoucího vývoje v této oblasti. U jednotlivých druhů detekce jsem popsal jejich výhody a možné nevýhody. Dále jsem v práci uvedl několik nástrojů, které lze s výhodou využít při implementaci detekčního algoritmu. Rovněž jsem zmínil i některá komerční řešení, která jsou dostupná na trhu a která se touto problematikou také zabývají.

V další části práce jsem vyjmenoval třídy aplikací, která jsou z pohledu cílového řešení podstatná na ověření fungování navrhovaného detekčního modulu. S ohledem na to jsem přistupoval k výběru vhodné metody detekce a výběru detekčního algoritmu. Vybrán byl algoritmus, kombinující algoritmy K-Means a K-Nearest Neighbor, který vzájemně eliminuje nevýhody obou algoritmů použitých samostatně.

Následným krokem byla implementace prototypu detekčního modulu, který vybraný algoritmus využívá. Byly implementovány dvě varianty — zjednodušená a cílová. Díky tomu jsem mohl provést srovnání výsledků těchto dvou variant. Pro cílovou variantu navíc byly implementovány dvě hodnotící funkce. Testování obou variant proběhlo na předem zachyceným reálným provozem. Výsledky měření jsem vynesl do tabulek a vzájemně jednotlivé výsledky porovnal. Nejelpších výsledků detekce v tomto případě dosáhla detekce HTTP provozu se spolehlivostí kolem 87 %. Naopak nejnižší úspěšnost detekce jsem zaznamenal u provozu DNS, která se pohybovala okolo 22 %. Z měření je patrné, že cílové řešení má potenciál na další zlepšování a před samotným zakomponováním do produktu bude ještě nutné provést optimalizace, zrychlující jeho chod a zlepšující úspěšnost detekce.

Po ověření vybraného algoritmu nad implementovaným prototypem jsem vytvořil několik možných návrhů na zakomponování funkčnosti detekce aplikací do současného firemního produktu Kernun UTM. U jednotlivých variant jsem bral v potaz jejich výhody a nevýhody a diskutoval je. Následně jsem vybral nejvhodnější variantu a popsal ji blíže. Tato varianta se stala základem pro počátek implementace výsledného řešení do cílového produktu.

Poslední část práce se zabývá zhodnocením úspěšnosti implementovaného prototypu a dosažených cílů. Jsou zde diskutovány možnosti, jak dále pokračovat v rozšiřování funkčnosti, mimo jiné možnost rozšíření detekce i o blokování vybraných aplikací v souladu se zakomponováním do navržené architektury. Dále jsou zmíněny některé nedostatky, na které se přišlo jak testováním prototypu, tak i započetím implementace do cílového produktu a jsou navržena jejich možná řešení.

Osobně jsem díky této práci nabyl spousty nových poznatků z oblasti možností detekce síťových aplikací a klasifikace síťového provozu. Mimo jiné jsem si vyzkoušel i základy vlastních úprav do jádra operačního systému FreeBSD.

Literatura

[1] Anderson, P.: What is Web 2.0? Ideas, technologies and implications for education. JISC Technology and Standards Watch, Feb. 2007.
Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.9995&rep=rep1&type=pdf

[2] Ma, J.; Levchenko, K.; Kreibich, C.; aj.: Unexpected means of protocol inference. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, New York, NY, USA: ACM, 2006, ISBN 1-59593-561-4, s. 313–326, doi:10.1145/1177080.1177123.
Dostupné z: http://doi.acm.org/10.1145/1177080.1177123

[3] Clarke, R.: Deep packet inspection: Its nature and implications [online]. [cit. 2013-09-27]. Office of the Privacy Commissioner of Canada's Deep Packet Inspection Project, March 2009.
Dostupné z: http://www.priv.gc.ca/information/research-recherche/2009/clarke_200903_e.asp

[4] Ramos, A.: *Deep packet inspection technologies. V Harold F. Timpton a Micki Krause (eds.) Information and Security Management Handbook (6. vyd.)*. New York: Auerbach Publications, 2009.

[5] *Information Technology - Open Systems Interconnection: Basic Reference Model: The Basic Model*. Second edition; corrected and reprinted 1996-06-15 vydání, 1994.
Dostupné z: http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip

[6] Callado, A.; Kamienski, C.; Szabo, G.; aj.: A Survey on Internet Traffic Identification. *Communications Surveys Tutorials, IEEE*, ročník 11, č. 3, rd 2009: s. 37–52, ISSN 1553-877X, doi:10.1109/SURV.2009.090304.

[7] Fielding, R.; aj.: RFC 2616. Hypertext Transfer Protocol – HTTP/1.1. Network Working Group. [online]. [cit. 2014-04-27]. June 1999.
Dostupné z: http://tools.ietf.org/html/rfc2616

[8] Ylonen, T.; Lonvick, C.: RFC 4252. The Secure Shell (SSH) Authentication Protocol. Network Working Group. [online]. [cit. 2013-09-27]. January 2006.
Dostupné z: https://tools.ietf.org/html/rfc4252

[9] Postel, J.; Reynolds, J.: RFC 854. TELNET PROTOCOL SPECIFICATION. Network Working Group. [online]. [cit. 2014-04-27]. May 1983.
Dostupné z: http://tools.ietf.org/html/rfc854

[10] Klensin, J.: RFC 2821. Simple Mail Transfer Protocol. Network Working Group. [online]. [cit. 2014-04-27]. April 2001.
Dostupné z: http://tools.ietf.org/html/rfc2821

[11] Siemborski, R.; Menon-Sen, A.: RFC 5034. The Post Office Protocol (POP3). Network Working Group. [online]. [cit. 2014-04-27]. July 2007.
Dostupné z: http://tools.ietf.org/html/rfc5034

[12] Internet Assigned Numbers Authority. [online]. [cit. 2013-09-27].
Dostupné z: www.iana.org

[13] Reed, J.: *The OpenBSD PF Packet Filter Book*. Reed Media Services, 2006, ISBN 9780979034206.
Dostupné z: http://books.google.cz/books?id=tsvvNwAACAAJ

[14] FreeBSD Handbook: Firewalls - IPFW. [online]. [cit. 2014-01-09].
Dostupné z: https://www.freebsd.org/doc/handbook/firewalls-ipfw.html

[15] Karagiannis, T.; Broido, A.; Brownlee, N.; aj.: Is P2P dying or just hiding? In *Global Internet and Next Generation Networks*, Dallas, Texas: Globecom 2004, Dec 2004.

[16] Snort: Free, open source network intrusion detection and prevention system [online]. [cit. 2014-01-09].
Dostupné z: http://www.snort.org

[17] The Bro Network Security Monitor [online]. [cit. 2014-01-09].
Dostupné z: http://www.bro.org/

[18] Check Point Software Technologies Ltd.: Application Control Software Blade [online]. [cit. 2014-01-09].
Dostupné z: http://www.checkpoint.com/products/application-control-software-blade/index.html

[19] Palo Alto Networkds: App-ID: Identifying Any Application on Any Port [online]. [cit. 2014-01-09].
Dostupné z: https://www.paloaltonetworks.com/products/technologies/app-id.html

[20] Procera Networks: Empowering Intelligence: Network Application Visibility Library [online]. [cit. 2014-01-09].
Dostupné z: http://www.proceranetworks.com/embedded-dpi-engine-navl.html

[21] Daly, A.: THE LEGALITY OF DEEP PACKET INSPECTION. *International Journal of Communications Law and Policy*, , č. No 14, 2011.
Dostupné z: http://ijclp.net/ojs/index.php/ijclp/article/view/31

[22] VRT: Snort 2.9 Essentials. The DAQ [online]. [cit. 2014-04-14].
Dostupné z: http://vrt-blog.snort.org/2010/08/snort-29-essentials-daq.html

[23] Cisco Security Introduces Open Source Application Detection and Control. The Network: Cisco's Technology News Site [online]. [cit. 2014-04-16].
Dostupné z: http://newsroom.cisco.com/release/1354502

[24] Cisco Announces OpenAppID – the Next Open Source 'Game Changer' in Cybersecurity. Cisco Blog: Security [online]. [cit. 2014-04-16].
Dostupné z: http://blogs.cisco.com/security/cisco-announces-openappid-the-next-open-source-game-changer-in-cybersecurity/

[25] Alshammari, R.; Zincir-Heywood, A.: Machine learning based encrypted traffic classification: Identifying SSH and Skype. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, 2009, s. 1–8, doi:10.1109/CISDA.2009.5356534.

[26] Bar Yanai, R.; Langberg, M.; Peleg, D.; aj.: Realtime classification for encrypted traffic. In *Proceedings of the 9th international conference on Experimental Algorithms*, SEA'10, Berlin, Heidelberg: Springer-Verlag, 2010, ISBN 3-642-13192-1, 978-3-642-13192-9, s. 373–385, doi:10.1007/978-3-642-13193-6˙32.
Dostupné z: http://dx.doi.org/10.1007/978-3-642-13193-6_32

[27] MacQueen, J.: Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, Berkeley, Calif.: University of California Press, 1967, s. 281–297.
Dostupné z: http://projecteuclid.org/euclid.bsmsp/1200512992

[28] Cover, T.; Hart, P.: Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, ročník 13, č. 1, 1967: s. 21–27, ISSN 0018-9448, doi:10.1109/TIT.1967.1053964.

[29] Canini, M.; Li, W.; Zadnik, M.; aj.: AtoZ: an automatic traffic organizer using NetFPGA. Technická Zpráva UCAM-CL-TR-750, University of Cambridge, Computer Laboratory, Květen 2009.
Dostupné z: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-750.pdf

[30] NetFPGA [online]. [cit. 2014-01-02].
Dostupné z: http://netfpga.org/

[31] Wang, Y.; Xiang, Y.; zheng Yu, S.: Automatic Application Signature Construction from Unknown Traffic. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, 2010, ISSN 1550-445X, s. 1115–1120, doi:10.1109/AINA.2010.120.

[32] Dan Pelleg, A. M.: X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, San Francisco: Morgan Kaufmann, 2000, s. 727–734.

[33] Palo Alto Networkds: Application Research Center [online]. [cit. 2014-01-09].
Dostupné z: https://applipedia.paloaltonetworks.com/

[34] Check Point Software Technologies Ltd.: AppWiki [online]. [cit. 2014-01-09].
Dostupné z: http://appwiki.checkpoint.com/appwikisdb/public.htm

[35] Rescorla, E.: RFC 2818. HTTP Over TLS. Network Working Group. [online]. [cit. 2014-04-27]. May 2000.
Dostupné z: http://tools.ietf.org/html/rfc2818

[36] Mockapetris, P.: RFC 1035. DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION. [online]. [cit. 2014-04-27]. November 1987.
Dostupné z: http://ftp.isi.edu/in-notes/rfc1035.txt

[37] Case, J.; Fedor, M.; Schoffstall, M.; aj.: RFC 1157. A Simple Network Management Protocol (SNMP). [online]. [cit. 2014-04-27]. May 1990.
Dostupné z: ftp://www.ietf.org/rfc/rfc1157.txt

[38] Wireshark: Go Deep [online]. [cit. 2014-01-12].
Dostupné z: https://wireshark.org/

[39] Kernun UTM. [online]. [cit. 2014-01-12].
Dostupné z: http://www.kernun.cz/produkty-case-study/kernun-utm/predstaveni-produktu/

[40] Kernun UTM Handbook. TRUSTED NETWORK SOLUTIONS, a.s. [online]. 2014 [cit. 2014-02-20].
Dostupné z: http://download.kernun.com/doc/handbook.html

[41] Qt Project [online]. [cit. 2014-02-22].
Dostupné z: http://qt-project.org/

[42] Neuman; aj.: *RFC 4120: The Kerberos Network Authentication Service (V5)*. 2005.
Dostupné z: http://www.ietf.org/rfc/rfc4120.txt

[43] Microsoft NTLM. Windows.
Dostupné z: http://msdn.microsoft.com/en-us/library/windows/desktop/aa378749(v=vs.85).aspx

[44] Comparing Windows Kerberos and NTLM Authentication Protocols. Security content from Windows IT Pro [online]. 2007 [cit. 2014-05-05].
Dostupné z: http://windowsitpro.com/security/comparing-windows-kerberos-and-ntlm-authentication-protocols

[45] Man-in-the-middle attack. OWASP [online]. [cit. 2014-04-20].
Dostupné z: https://www.owasp.org/index.php/Man-in-the-middle_attack

[46] Pcap(3). FreeBSD Man Pages [online]. [cit. 2014-05-05].
Dostupné z: http://www.freebsd.org/cgi/man.cgi?query=pcap&apropos=0&sektion=3&manpath=FreeBSD+8.4-stable&arch=default&format=html

[47] Divert. FreeBSD Man Pages [online]. [cit. 2014-05-05].
Dostupné z: http://www.freebsd.org/cgi/man.cgi?query=divert

Dodatek A

Obsah DVD

K práci je přiložen DVD disk, obsahující tento text ve formátu .pdf, zdrojové texty práce v jazyce Tex, včetně všech obrázků i jejich zdrojových souborů. Dále obsahuje zdrojové soubory implementace.

Struktura obsahu DVD:

xstour03.pdf
: text práce ve formátu .pdf

latex/
: složka se zdrojovými texty práce v jazyce latex včetně všech zdrojových obrázků, přeložitelné na stroji merlin.fit.vutbr.cz

src/
: složka se zdrojovými soubory implementace

readme.txt
: textový soubor, obsahující tyto informace