VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

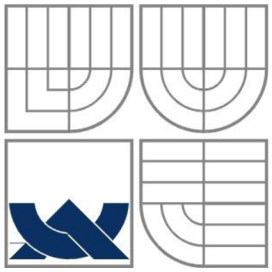
ANALÝZA AUTOMATIZOVANÉHO GENEROVÁNÍ SIGNATUR S VYUŽITÍM HONEYPOTU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

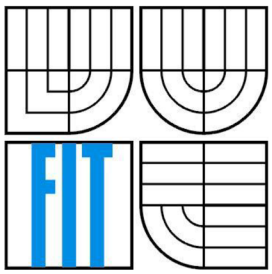
AUTOR PRÁCE
AUTHOR

Bc. LUKÁŠ BLÁHA

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ANALÝZA AUTOMATIZOVANÉHO GENEROVÁNÍ SIGNATUR S VYUŽITÍM HONEYPOTU

ANALYSIS OF AUTOMATED GENERATION OF SIGNATURES USING HONEYPOTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. LUKÁŠ BLÁHA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL DROZD

BRNO 2012

Abstrakt

V této práci je diskutován systém automatického zpracování útoků za pomoci honeypotů. Prvním cílem diplomové práce je seznámení se s problematikou tvorby signatur pro detekci škodlivého kódu na síti, především pak analýza a popis existujících metod automatického generování signatur za pomoci honeypotu. Hlavním cílem práce je využít získaných znalostí k navržení a implementaci nástroje, který bude provádět detekci nově odchyceného škodlivého software na síti či koncové uživatelské stanici.

Abstract

In this paper, system of automatic processing of attacks using honeypots is discussed. The first goal of the thesis is to become familiar with the issue of signatures to detect malware on the network, especially the analysis and description of existing methods for automatic generation of signatures using honeypots. The main goal is to use the acquired knowledge to the design and implementation of tool which will perform the detection of new malicious software on the network or end user's workstation.

Klíčová slova

Signatura, detekční profil, automatické generování signatur, honeypot, škodlivý software, systém detekce průniku, zero-day útok, buffer overflow útok.

Keywords

Signature, detection profile, automatic generation of signatures, honeypot, malware, intrusion detection system, zero-day attack, buffer overflow attack.

Citace

Bláha Lukáš: Analýza automatizovaného generování signatur s využitím Honeypotu, diplomová práce, Brno, FIT VUT v Brně, 2012

Analýza automatizovaného generování signatur s využitím Honeypotu

Prohlášení

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

.....
Lukáš Bláha
20. května 2012

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Michalovi Drozdovi za ochotu a čas strávený při konzultacích této práce. Dále bych rád poděkoval všem členům řešitelskému týmu projektu AIPS vedeného na FIT VUT v Brně za poskytnutá testovací data a cenné rady. Velké díky patří mé rodině za vytrvalou podporu během celého mého studia.

Obsah

1 Úvod

V dnešní době jsou počítačové systémy nedílnou součástí lidského života. Využíváme je již od raného dětství a setkáváme se s nimi téměř na každém kroku. Při studiu ve škole, při práci v zaměstnání, při odpolední zábavě s přáteli i při vyhledávání informací či zjišťování aktuálních zpráv na internetu. Většina z nás, ač si to možná neuvědomujeme, je v kontaktu s počítačovým systémem téměř celý den. Tímto systémem je mobilní zařízení, které denně využíváme k nejrůznějším úkonům jako je například komunikace s přáteli na sociálních sítích, vyřizování pracovních emailů nebo provádění bankovních transakcí.

Fakt, že jsme v téměř nepřetržitém kontaktu s počítačovým systémem, vede k tomu, že všechna citlivá data, ať už firemní či osobní, jsou dnes uložena výhradně v elektronické podobě. Tato data jsou velmi lákavá pro kybernetické zločince, kteří za účelem jejich získání dokáží vynaložit nemalé finanční prostředky. V dřívější době byly škodlivé kódy vytvářeny především za účelem působení škod či pro získání nehynoucí slávy jejich tvůrců. Škodlivý software dnešní doby má úplně jinou podobu. Z jeho tvorby se stal organizovaný zločin, který slouží k cílené a sofistikované krádeži citlivých dat. Za účelem odcizení důvěrných údajů jsou najímány celé týmy počítačových profesionálů, kteří dokáží denně vyprodukovat velké množství nového škodlivého software, který je čím dál více sofistikovanější. Šíření tohoto kódu je díky síti internet jednoduché a také rychlé.

Dosavadní bezpečnostní technologie pracují se zastaralými technikami, které jsou v boji proti sofistikovanému malware málo efektivní. Jedná se převážně o detekci na základě databáze signatur[1], která obsahuje vzorky pouze existujících, již zanalyzovaných útoků. Postup aktualizace těchto systémů je postaven na manuální tvorbě signatur pro každý nově nalezený škodlivý kód. Tento proces je při tak velkém množství nově vygenerovaného malware[2] velmi náročný a pomalý. Škodlivý software v dnešní době dokáže v časovém intervalu od jeho objevení po vydání aktualizace pro bezpečnostní technologie kompromitovat tisíce až milióny systémů po celém světě a způsobit tak velké škody.

Tato diplomová práce se věnuje systému, který se snaží proces tvorby signatur automatizovat. Navazuje tak na semestrální projekt, ve kterém byl systém pro automatické vytváření signatur popsán spolu s analýzou existujících metod automatického generování signatur. Automatický systém by měl v prvním kroku zajistit detekci nově vzniklých a probíhajících útoků. K tomuto účelu jsou nasazeny honeypoty[3]. V dalším kroku systém automaticky vytvoří odpovídající signaturu, která musí být doručena jako aktualizace ke koncovým uživatelům. Proces automatické tvorby signatur snižuje reakční dobu pro detekci nově vzniklých útoků, což by mělo zajistit výrazné snížení počtu infikovaných systémů v krátkém časovém úseku.

1.1 Cíle práce

Prvním cílem diplomové práce je seznámení se s problematikou tvorby signatur pro detekci škodlivého kódu na síti, především pak analýza a popis existujících metod automatického generování signatur za pomoci honeypotu. Hlavním cílem práce je využít získaných znalostí k navržení a implementaci nástroje, který bude provádět detekci nově odchyceného škodlivého software na síti či

[1] Signatura (detekční profil) - podrobněji vysvětleno v kapitole 3.1 12Signatura (detekční vzor).
[2] Malware - zkratka pro malicious software. Tento termín bude vysvětlen v 2. kapitole Malware a jeho detekce.
[3] Honeypot - systém umožňující zachytávat nové vzorky škodlivého software. Podrobněji vysvětlen v kapitole 3.3 Honeypoty.

koncové uživatelské stanici. Podklady využité v práci vychází z řešení projektu AIPS zaměřeného na automatizované zpracování útoků, na kterém se podílí více řešitelů v rámci FIT VUT v Brně.

1.2 Struktura práce

Druhá kapitola této práce popisuje charakteristiky současného škodlivého software spolu s aktuálně používanými technikami pro jeho detekci. Je uveden současný proces tvorby signatur a jsou diskutovány jeho hlavní nedostatky a jejich možná řešení.

Ve třetí kapitole jsou uvedeny komponenty, ze kterých je systém automatického generování signatur složen. Každá komponenta je popsána a zároveň je vysvětlena její funkce v systému. Na závěr kapitoly je zobrazena obecná architektura systému pro automatické generování signatur.

Čtvrtá kapitola obsahuje analýzu a popis existujících metod pro automatickou tvorbu signatur. Pro každou analyzovanou metodu je nejdříve uveden její popis, poté její vstupní a výstupní hodnoty a nakonec je popsán mechanizmus použitý pro generování výsledné signatury. Pokud byla nalezena ukázková výstupní signatura, je uvedena na konci popisu dané metody.

V páté kapitole je uveden popis celkové architektury systému pro automatické zpracování síťových útoků vyvíjeného na FIT VUT v Brně. Nejdříve je popsána architektura již existující serverové části, která slouží ke generování metrik identifikujících zachycené útoky. V druhé části kapitoly je popsána mnou navržená architektura detekční části, jejímž úkolem je monitorovat probíhající síťový provoz a detekovat v něm vzorky útoků.

Šestá kapitola obsahuje popis použitého vývojového prostředí spolu s implementačními detaily jednotlivých modulů navrženého detekčního nástroje Attack Detector. U každého modulu jsou popsány i použité podpůrné nástroje a jsou uvedeny důvody, které mě vedly právě k jejich výběru.

V sedmé kapitole je popsán proces testování vytvořeného nástroje. Nejdříve jsou uvedeny modely navržených prostředí, na kterých bylo testování prováděno a dále je popsán testovací scénář, který sloužil k provedení testů a odladění detekčních schopností nástroje Attack Detector.

Osmá kapitola obsahuje výsledky provedených testů. Je uveden počet korektně detekovaných útoků spolu s počtem vygenerovaných falešných poplachů. Na závěr kapitoly jsou uvedeny výsledky testů rychlosti nástroje.

Závěrečná kapitola je věnována zhodnocení celé práce. Je zde diskutováno splnění cílů vytyčených v úvodu spolu s možnostmi rozšíření a vylepšení dosavadní práce.

2 Malware a jeho detekce

Hlavním cílem této práce je vytvoření nástroje, který dokáže detekovat škodlivý software v síťovém provozu, které nedokážou detekovat běžné technologie jako AntiVir, Intrusion Detection Systémy a Firewally. V úvodu této kapitoly je vysvětlen a vymezen pojem malware (škodlivý software), který bude používán v průběhu celé práce a jehož vymezení je důležité pro pochopení funkce vytvářeného nástroje. Následuje popis současných trendů v oblasti tvorby škodlivých kódů spolu s aktuálním stavem výskytu malware v síti internet. Další částí této kapitoly je popis dosavadních technik, které jsou používány pro detekci případně prevenci proti malware. Na závěr kapitoly jsou diskutovány nedostatky současného přístupu k boji proti malware a je uvedeno jedno z jejich možných řešení, systém pro automatizované zpracovávání útoků, který je základem této diplomové práce.

2.1 Rozdělení malware

Pojem malware je zkratka pro malicious software, anglický překlad českého výrazu škodlivý software. Jedná se o nejobecnější pojem používaný v počítačové terminologii pro označení všech forem nežádoucího, nebezpečného a škodlivého kódu, který slouží pro neoprávněné získání přístupu, krádež či poškození dat. Malware můžeme dělit do velkého počtu kategorií, přičemž mezi jednotlivými skupinami neexistují přesně vymezené hranice. V této kapitole budou stručně zmíněny pouze některé hlavní kategorie malware, které souvisí s tématem práce.

Pojem malware je v této práci používán výhradně pro škodlivý software, který využívá zranitelnosti přetečení zásobníku (buffer overflow). Tato zranitelnost vede nejčastěji k vzdálenému či lokálnímu spuštění libovolného kódu, s jehož pomocí dokáže útočník kompromitovat celý systém. Dalším častým útokem zneužívající tuto zranitelnost je tzv. DoS (Denial of Service), který způsobí nedostupnost cílové služby.

2.1.1 Počítačové viry

Jako virus se v oblasti počítačové bezpečnosti označuje škodlivý program, který se dokáže sám šířit bez vědomí uživatele. Za účelem šíření vkládá svůj zdrojový kód do jiných programů, většinou spustitelných souborů či dokumentů.

2.1.2 Počítačoví červi

Počítačový červ je program s vlastní schopností replikace. Hlavním rozdílem oproti virům je fakt, že ke svému šíření nepotřebuje žádné další soubory. Ve většině případů ke svému spuštění nepotřebuje ani prvotní uživatelskou akci. Červi využívají zranitelností v operačním systému či službách a šíří se výhradně počítačovou sítí, pomocí které zasílají své kopie do ostatních nechráněných zařízení. K šíření používají různé komunikační kanály jako je například email, IRC, ICQ a další.

2.1.3 Trojské koně

Pojmem trojský kůň označujeme v počítačové terminologii program, který za účelem instalace škodlivého kódu do systému předstírá legální funkcionalitu.

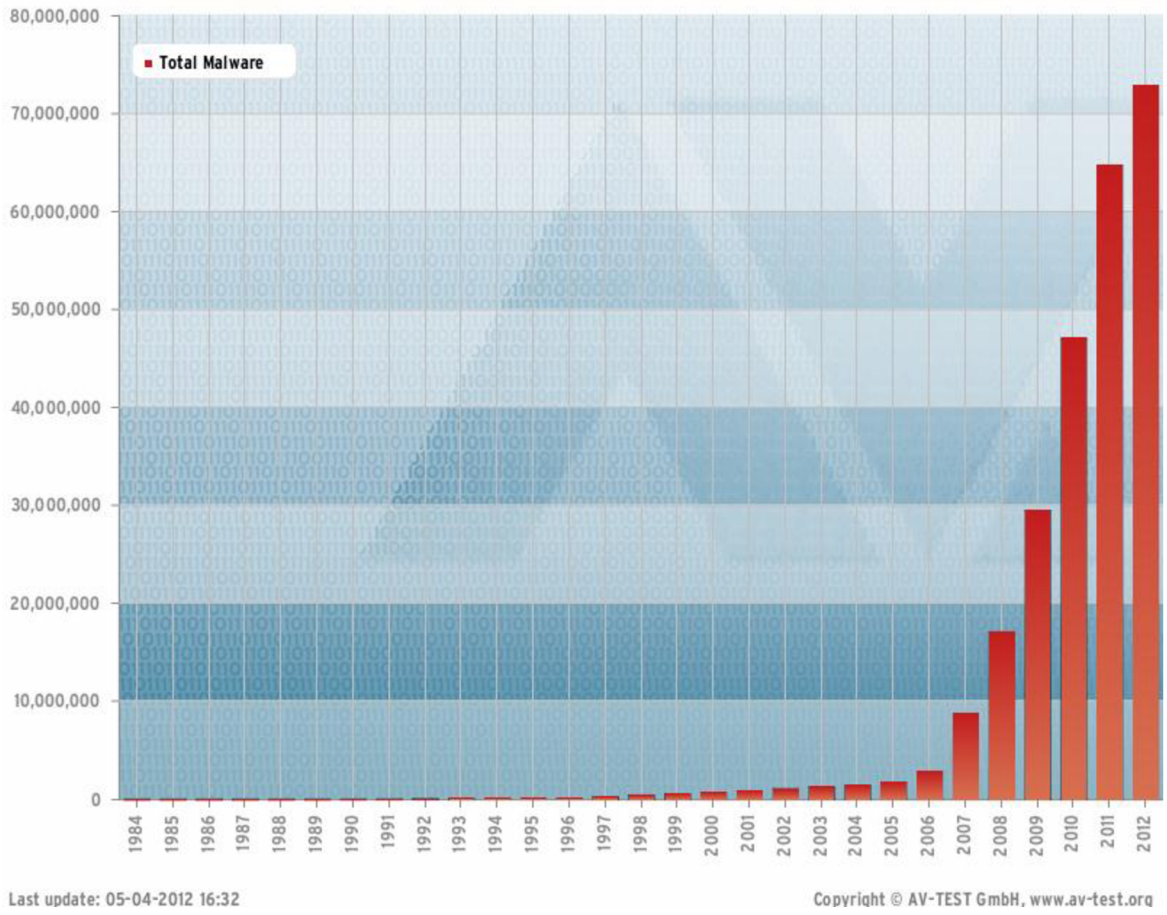
2.1.4 Rootkity

Rootkity používají pokročilé techniky, které spočívají v modifikaci součástí operačního systému. Hlavním cílem je snaha o maximální ukrytí škodlivého kódu a omezení možné detekce. Rootkity se před uživatelem maskují a snaží se zakrývat vlastní běžící procesy případně soubory na disku.

2.2 Povaha dnešního malware

V minulosti bylo zaměření a šíření malware značně odlišné od jeho dnešní podoby. Viry byly zaměřeny především na působení škod či za účelem získání slávy a uznání pro jejich tvůrce. Jejich šíření ze stanice na stanici probíhalo výhradně pomocí přenosných médií. S rozvojem širokopásmového internetového připojení se povaha škodlivého software výrazně změnila. Jeho šíření pomocí této sítě se stalo daleko snadnější a rychlejší. V souvislosti s tím se začalo hovořit o rychle se šířících internetových červech. Rozšířením internetu do všech sfér společnosti výrazně vzrostl i počet potencionálních obětí počítačových útoků.

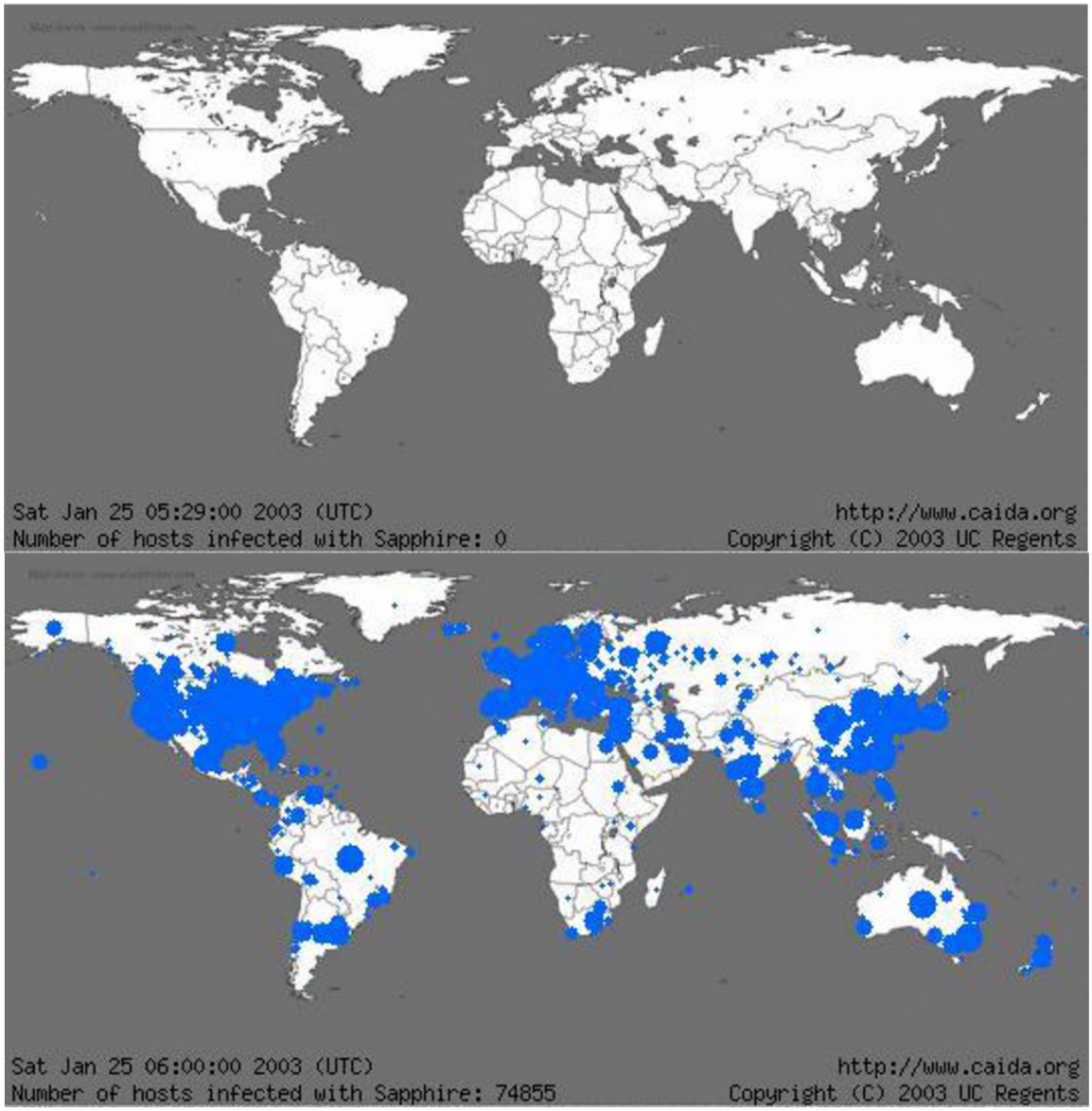
Pro tvorbu nových škodlivých kódů jsou najímány specializované týmy odborníků, kteří vymýšlejí stále sofistikovanější metody, jak pomocí malware získat hodnotná firemní či osobní citlivá data. To vede k velkému počtu nově vzniklého škodlivého kódu. Výrazný růst malware v posledních letech je znázorněn na následujícím obrázku 2.1.



Obrázek 2.1: Počet objeveného malware v jednotlivých letech [1]

Většina nového malware využívá tzv. zero-day zranitelností či chybu v nastavení aplikace. Zero-day zranitelnosti jsou dosud neobjevené bezpečnostní chyby operačních systémů a aplikací. K vyhledávání těchto zranitelností jsou mnohdy najímány týmy reverzních inženýrů, kteří provádí podrobnou analýzu zdrojových kódů software a hledají v nich implementační chyby. Příkladem takového malware může být v poslední době velmi diskutovaný červ Stuxnet, který pro zajištění úspěchu útoku zneužíval hned několik zero-day zranitelností [2]. Obrana proti zero-day útokům je vcelku obtížná. Možné přístupy detekce jsou popsány v následující kapitole.

Další nevýhodou nově vytvořeného malware je rychlost jeho šíření. Díky síti internet se dnešní škodlivé kódy dokáží šířit velkou rychlostí. Během několika minut se dokáží rozšířit do všech geografických částí zeměkoule a stihnou infikovat tisíce zranitelných systémů [3]. Na níže uvedeném obrázku 2.2 je zobrazeno rozšíření internetového červa *Slammer*, který v časovém intervalu 30 minut dokázal infikovat téměř 75 tisíc stanic. Studie pochází již z roku 2003 a byla vybrána proto, že výstižně demonstruje šíření internetových červů. Rychlost šíření současných červů obsahujících zero-day útok je vzhledem k většímu počtu připojených stanic do internetu daleko vyšší. Příkladem může být červ Conficker, který se poprvé objevil na konci roku 2008 a jehož hrozba přetrvává do dnešní doby. Tento červ dokázal za první měsíc svého působení nakazit 15 milionů zařízení [4].



Obrázek 2.2: Rozšíření internetového červa *Slammer* v časovém intervalu třiceti minut [3]

2.3 Techniky detekce malware na síti

Pro detekci škodlivého software na síti jsou v dnešní době využívány především systémy IDS/IPS[4]. Tyto systémy mohou používat jednu nebo více následujících metod:

- porovnávání detekčních vzorů (signatur),
- vyhledávání statistických anomálií,
- vyhledání odchylek od protokolových norem,
- integritní analýza.

2.3.1 Porovnávání detekčních vzorů (signatur)

Tato metoda funguje na principu monitorování aktuálního provozu, ve kterém vyhledává signatury známých útoků. Tento přístup generuje minimální procento falešných poplachů, avšak jeho nevýhodou je závislost IDS systému na aktualizaci databáze známých útoků. Pokud útok změní svou charakteristiku pouze v jediném bajtu, nebude IDS systém založený na porovnávání signatur schopný tento útok detekovat. Zmíněná technika neumožňuje detekci žádného nově vytvořeného malware. Další nevýhodou je proces vytváření nových signatur, který je popsán v následující kapitole 2.4.

2.3.2 Vyhledávání statistických anomálií

Při vyhledávání statistických anomálií nejdříve systém IDS po určitou dobu monitoruje běžný síťový provoz v uzavřeném testovacím prostředí a ukládá si jeho statistické charakteristiky. Poté je systém přenesen do produkčního prostředí, kde kontroluje síťový provoz a vyhledává v něm odchylky od stanovených charakteristik. Pokud jsou v provozu zaznamenány anomálie, je ihned vygenerován příslušný alarm. Tento přístup dokáže detekovat některé nové útoky, má však nevýhodu v podobě vyššího procenta falešných poplachů, protože jako anomálii může detekovat i legitimní provoz, který se nenaučil v testovacím prostředí.

2.3.3 Vyhledávání odchylek od protokolových norem

Další metodou je vyhledávání odchylek od protokolových norem. Každý protokol je důkladně specifikovaný v oficiálním dokumentu (nejčastěji v RFC[5]). Tato metoda funguje na principu detekce odchylek od definovaných norem. Nevýhodou této metody může být vyšší procento generování falešných poplachů. Důvodem je mnohdy vágní definice komunikačních protokolů nebo nedodržování definovaných standardů ze strany programátorů.

2.3.4 Integritní analýza

Poslední metodou je detekce škodlivého provozu na základě integritní analýzy. Tato metoda pomocí kryptografických bezpečnostních funkcí zkoumá, zda daný objekt během komunikace nebyl modifikován [5], [6].

---

[4] Intrusion Detection/Prevention System – systémy pro odhalení či zamezení průniku. Detailněji popsány v kapitole 3.2 IDS / IPS systémy.

[5] Request For Comments – technické a organizační dokumenty o internetu, spravované organizací IETF. <http://www.ietf.org/rfc.html>

2.4 Současný proces tvorby signatur

Dnešní bezpečnostní technologie detekující malware na základě porovnávání signatur mají velkou nevýhodu v tom, že tvorba nových vzorků je prováděna poloautomatizovaným procesem.

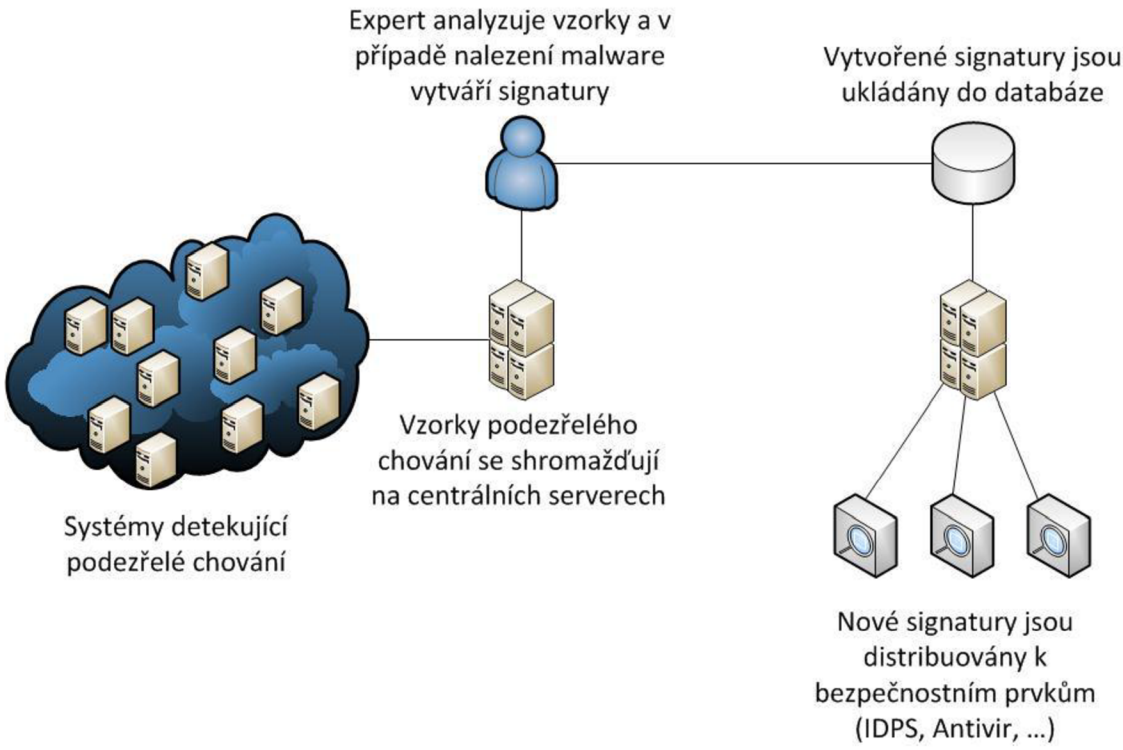
Každá společnost poskytující bezpečnostní řešení (především společnosti vytvářející detekční nástroje založené na signaturách) má pro sběr vzorků podezřelého chování k dispozici velké množství systémů umístěných v síti internet. Většinou se jedná o koncové stanice, které mají nainstalované bezpečnostní řešení od konkrétní firmy (antivir, sandbox, honeypot), který na základě heuristik a jiných přístupů dokáže detekovat anomální chování v systému.

Zaznamenané vzorky podezřelého provozu jsou automaticky zasílány na centrální servery, kde jsou shromažďovány a připraveny pro pozdější analýzu.

V dalším kroku přichází kritická část celého procesu. Sesbírané vzorky jsou ručně analyzovány expertem resp. týmem expertů, a pokud vzorek obsahuje nějaký nový typ malware, je vytvořena odpovídající signatura.

Tato signatura je vložena do centrální databáze vzorků a o vložení jsou informovány servery, obstarávající aktualizace bezpečnostních prvků. Aktualizace databáze vzorků koncových zařízení může být provedena dvěma způsoby. Prvním, častějším způsobem, je tzv. push přístup, kdy jsou aktualizace z centrálního serveru automaticky posílány na koncová zařízení, která ihned aktualizují své databáze vzorků. Druhým způsobem je tzv. pull přístup, kdy centrální server neinformuje koncová zařízení o nové aktualizaci a čeká, až si nové aktualizace klient sám vyžádá [7].

Celý proces současné tvorby signatur je pro lepší pochopení znázorněn na následujícím obrázku 2.3.



Obrázek 2.3: Současný proces aktualizace databáze vzorků malware

2.5 Nedostatky dosavadního přístupu tvorby signatur a jejich možné řešení

Kritickým faktorem v boji proti nově vzniklému malware je reakční doba od objevení zranitelnosti po dobu, kdy je systém aktualizován a je imunní vůči zneužití této zranitelnosti. Hlavním nedostatkem dosavadního přístupu generování signatur je jeho závislost na lidské interakci při tvorbě jednotlivých signatur. Vložením lidského faktoru do tohoto procesu je reakční doba na útok značně prodloužena (mnohdy v rámci dnů až týdnů). Při tak vysoké rychlosti šíření dnešních škodlivých kódů dokáže malware v tomto dlouhém časovém intervalu infikovat tisíce až milióny zařízení. Později vydaná aktualizace tak mnohdy ztrácí na významu.

Řešením tohoto problému je eliminace lidského faktoru v celém procesu. Pomocí automatizace části generování signatur dokážeme výrazně snížit reakční dobu na útok, což zabrání masivnímu rozšíření nově objevených škodlivých kódů.

Systém provádějící automatizaci této části je detailněji popsán v následující kapitole 3 Systém automatického generování signatur a je diskutován v průběhu celé této práce.

3 Systém automatického generování signatur

Na konci předchozí kapitoly bylo zmíněno možné řešení problému současného procesu vytváření detekčních profilů, kterým může být systém automatizovaného generování signatur. V této části práce jsou uvedeny jednotlivé komponenty systému pro automatické generování signatur za pomoci honeypotu a jsou vysvětleny pojmy, jejichž znalost je nezbytná k následné orientaci v textu. Na závěr kapitoly je zobrazeno zjednodušené schéma architektury systému automatického generování signatur.

3.1 Signatura (detekční vzor)

Pojem signatura je v počítačovém světě chápán jako vzor, který identifikuje existující škodlivý kód. V antivirových programech obsahují signatury pouze jednotlivé kusy škodlivého kódu (např. spustitelného souboru), v IDS/IPS systémech jsou signatury rozšířeny o další informace, identifikující síťový provoz (například použitý protokol, cílový port, atd.).

Signatura nebo též detekční vzor je v této práci chápán jako jakákoliv kolekce charakteristik, která klasifikuje síťový provoz jako škodlivý. V podstatě se jedná o množinu pravidel, vůči kterým je porovnáván procházející síťový provoz a pokud jsou všechna pravidla splněna, je daný provoz (většinou paket nebo síťový tok) identifikován jako škodlivý. Dle povahy nástroje je poté buď vygenerován příslušný alarm (IDS) nebo je škodlivý provoz ihned ukončen (IPS) [8].

Příklad signatury pro volně dostupný IDPS nástroj Snort[6] je uveden v následujícím výpisu:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS
                               $HTTP_PORTS
   (msg:"WEB-CGI finger access";
    flow:to_server,established;
    uricontent:"/finger"; nocase;
    reference:arachnids,221;
    reference:cve,1999-0612;
    reference:nessus,10071;
    classtype:attempted-recon;
    sid:839;
    rev:7;)
```

Text 3.1: Signatura IDS nástroje Snort

Detekční vzory jsou v systému pro automatické generování signatur základním stavebním kamenem. Musí být dobře specifikovány a generovány, v opačném případě bude docházet k velkému množství falešných poplachů nebo k nízkému procentu zachycených útoků.

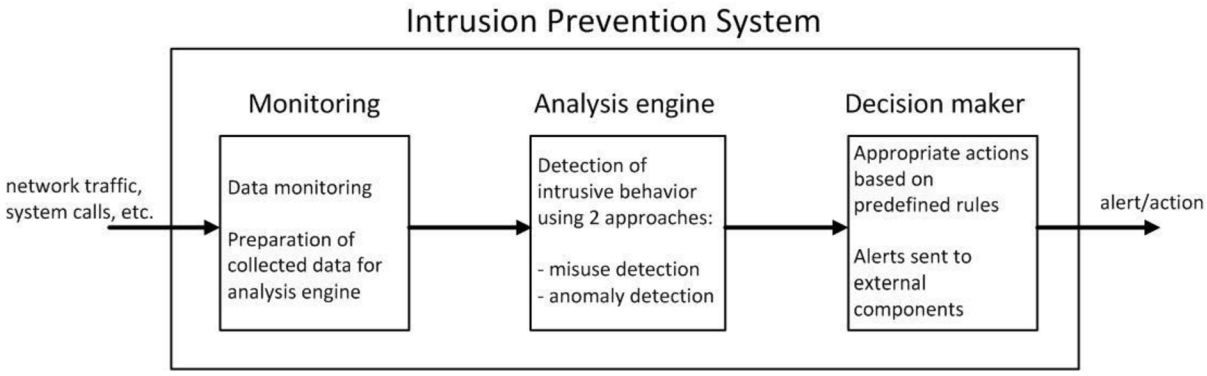
3.2 IDS / IPS systémy

Systém pro odhalení průniku (Intrusion Detection System, IDS) je v informatice chápán jako pasivní systém, který monitoruje události v počítačovém systému nebo síti, porovnává je s nakonfigurovanými pravidly a provádí definovanou akci hned, jak najde nějakou podezřelou aktivitu. IDS systémy dokáží detekovat několik typů škodlivého provozu jako je například útok na síťové služby, útoky na aplikace, neautorizované přístupy a všechny typy malware. Nevýhodou těchto

[6] Snort - analyzátor síťového provozu a open-source IDPS nástroj. <http://www.snort.org>

systémů je vyšší množství vygenerovaných falešných poplachů (tzv. false-positives), kdy je legální provoz označen jako škodlivý. IDS systémy musí být většinou vyladěny tak, aby byl minimalizován počet falešných poplachů, ale zároveň byla maximalizována úroveň detekce opravdových hrozeb. Této optimální úrovně je v reálném prostředí složité dosáhnout.

Systém prevence narušení (Intrusion Prevention System, IPS) je nástroj, který má všechny funkce systému IDS. IPS systém je však na rozdíl od IDS systému aktivní, může se tedy pokusit o eliminaci vzniku možného incidentu. Architektura IPS systému je zobrazena na následujícím obrázku 3.2.



Obrázek 3.2: Architektura IPS systému [9]

IDS a IPS technologie nabízejí mnoho stejných schopností a správci mohou obvykle zakázat preventivní funkce v produktech IPS, což způsobí, že se systém bude chovat pouze jako IDS. Pro lepší orientaci je ve zbytku této práce používán pro obě technologie souhrnný název Intrusion Detection and Prevention System, IDPS. Případné výjimky jsou výslovně uvedeny.

Systémy IDPS se dají dělit podle jejich umístění na HIDPS (Host-based IDPS) a NIDPS (Network-based IDPS). Zatímco HIDPS je softwarový agent, který sleduje systémová volání, činnosti aplikací a úpravy na souborovém systému svého hostitele, NIDPS je většinou nezávislá platforma, která provádí inspekci všech paketů zachycených na síti a snaží se v nich nalézt škodlivý provoz [10].

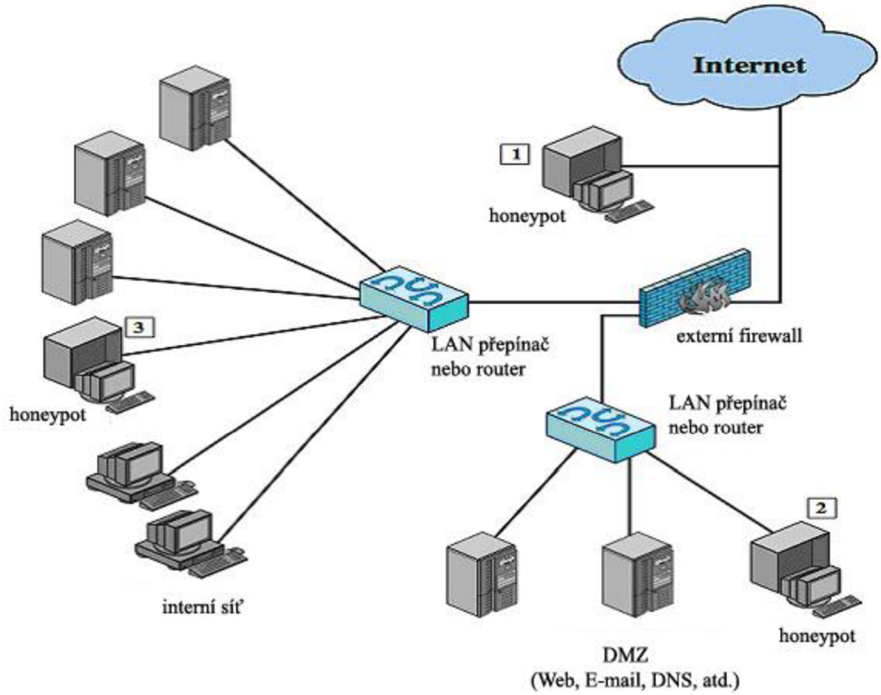
V této práci budeme uvažovat IDPS systémy, které pracují na principu porovnávání signatur. V systému pro automatické generování signatur jsou IDPS systémy koncovým zařízením, pro které jsou signatury generovány.

3.3 Honeypoty

Honeypot je vysoce flexibilní nástroj, který může být nasazen při řešení různých situací v počítačové bezpečnosti. Můžeme ho popsat jako detailně monitorovaný počítačový zdroj, který je sestaven k účelu být prozkoumávaný, napadený nebo kompromitovaný. Honeypot je strategicky umístěn v síti a nakonfigurován tak, aby vypadal jako reálný systém, na který je možné úspěšně provést útok. Díky tomuto konceptu je jakákoli komunikace s honeypotem podezřelá. Mezi hlavní funkce honeypotu patří:

- odvedení pozornosti útočníka tak, aby hlavní informační zdroje nebyly kompromitovány,
- získání informací o postupu útoků,
- identifikování nových zranitelností různých operačních systémů, prostředí a programů, které nebyly doposud odhaleny,

- odchycení počítačových virů a červů pro budoucí studii.



Obrázek 3.3: Možné umístění honeypotů v síťové infrastruktuře [11]

Nejčastěji jsou honeypoty rozdělovány podle jejich úrovně interakce. Honeypoty s vysokou úrovní interakce (high-interaction) poskytují útočníkovi reálný systém, zatímco honeypoty s nízkou úrovní interakce (low-interaction) simulují pouze části systému, například určité služby.

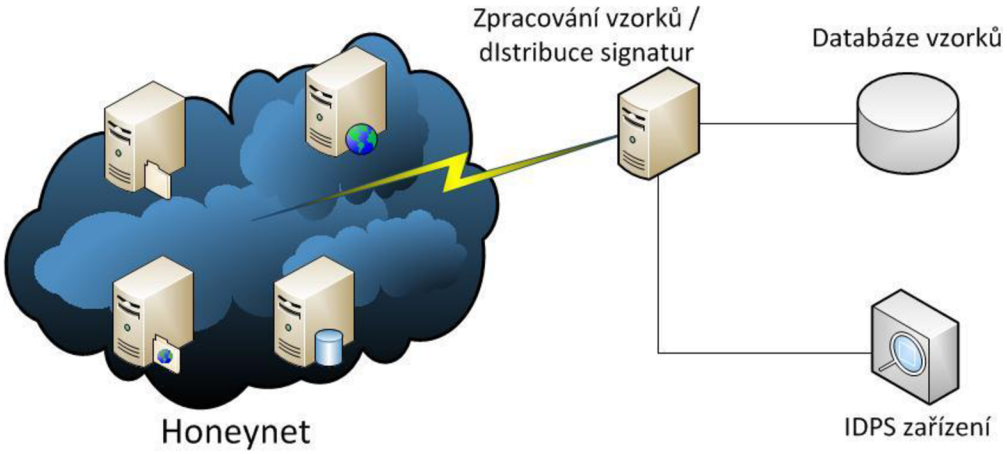
High-interaction honeypoty jsou klasické počítačové systémy. Představují komplexní řešení, protože zahrnují použití operačních systémů a reálných aplikací. Přináší možnost zachycení velkého množství informací, mezi které může patřit odhalení nových rootkitů, zero-day útoků nebo komunikace mezi útočníky.

Low-interaction honeypoty většinou napodobují (emulují) dané služby reálného systému, čímž dovolují útočníkovi pouze omezenou úroveň interakce. Jsou využívány především k shromažďování statistických dat a sběru informací o útocích. Výhodou těchto honeypotů je jejich jednoduchost a snadná údržba. Ve většině případů stačí pouze nainstalovat a nakonfigurovat určitý nástroj [12].

Sdružení většího počtu honeypotů do jedné sítě se nazývá honeynet. Honeypoty resp. honeynety hrají v systému pro automatické generování signatur důležitou roli. Jejich primární funkcí je detekce a sběr informací o nových útocích. Na úrovni detailu zaznamenaných informací je poté závislá přesnost generovaných detekčních vzorů.

3.4 Zjednodušené schéma systému

Pro lepší pochopení celé architektury a funkce systému pro automatické generování signatur jsem vytvořil zjednodušené schéma, které se nachází na obrázku 3.4: Zjednodušené schéma systému pro automatické generování signatur Základními komponentami jsou honeypoty umístěné v honeynetu, které provádí sběr informací o útocích. Centrální server generuje nebo případně i přeposílá nové signatury do centrální databáze a do klientských IDPS zařízení.



Obrázek 3.4: Zjednodušené schéma systému pro automatické generování signatur

4 Existující metody automatického vytváření detekčních profilů

Proces detekce malware v síťovém provozu založený na signaturách je náročný a jeho efektivita závisí na kvalitě detekčních profilů. Vytvoření kvalitního profilu zabírá dlouhý časový interval, který prodlužuje reakční dobu na odhalení nově vzniklých útoků. Z tohoto důvodu je již dlouhodobou snahou vytvořit mechanismus, který by detekční profily automatizovaně vytvářel.

Existující metody automatického generování signatur používají pro vytvoření detekčních profilů rozdílné přístupy, podle kterých se dají dělit do tří skupin popsaných v této kapitole.

U každé analyzované metody je uveden její stručný popis, její vstup a výstup a detailněji popsaný mechanizmus použitý pro generování signatury. Pokud byla nalezena ukázková signatura, je uvedena na konci popisu metody. V této kapitole nejsou diskutovány metody, které nejsou plně automatizované a k tvorbě signatur vyžadují zásah uživatele (např. zajímavá metoda HoneyAnalyzer [13]). Informace uvedené v této kapitole byly čerpány z dostupné literatury. Pro jejich ověření funkčnosti vybraných metod bylo provedeno jejich reálné nasazení.

4.1 Content-based signatury

Content-based signatura je založena na obsahu škodlivých paketů. Hlavním úkolem při vytváření signatury je identifikace charakteristického řetězce, který se nachází v těle škodlivých síťových paketů. IDPS systémy poté porovnají každý procházející paket s řetězcem uvedeným v signatuře, a pokud je nalezena shoda, je paket označen jako škodlivý a je provedena odpovídající akce. Tento způsob je často používán díky své jednoduché implementaci a rychlosti. Hlavním problémem těchto metod je jejich statická povaha. Pokud bude v těle paketu pozměněn nějaký bajt, IDPS systémy nebudou schopny tento paket nadále vyhodnotit jako škodlivý.

Content-based signatury jsou široce používány systémy pro automatické generování signatur. Příklad jejích častého formátu je uveden v následujícím výpisu.

```
(číslo IP protokolu, cílový port, sekvence bajtů)
```

Text 4.1: Příklad formátu content-based signatury

4.1.1 Honeycomb

Honeycomb je postaven jako rozšíření pro low-interaction honeypot honeyd[7]. Síťový provoz odchycený honeypotem je použit jako vstup pro algoritmus na generování signatur. Díky povaze honeypotu je veškerý na něj směrovaný síťový provoz považován za škodlivý. Výstup Honeycombu může být konvertován na formát signatur pro IDS systémy Snort a Bro[8].

Vstup:

Síťový provoz z honeypotu.

Výstup:

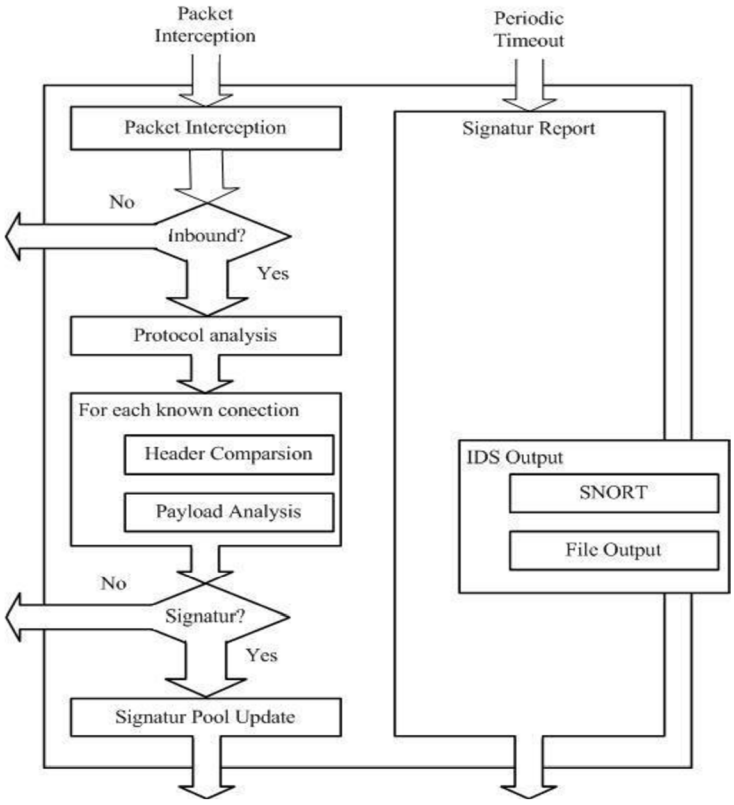
Signatury použitelné v IDS Snort a IDS Bro.

[7] HoneyD - low-interaction honeypot, který v síti dokáže vytvořit virtuální hosty <http://www.honeyd.org/>.
[8] Bro - unixový open-source NIDS <http://bro-ids.org>.

Generující mechanizmus:

Signatura je generována ve dvou krocích. V prvním kroku jsou pakety sestaveny do síťových toků. Pro protokol TCP jsou toky ukládány jako zprávy seřazené podle jejich směru, jelikož TCP spojení jsou obousměrná. V případě protokolu UDP jsou zprávy ukládány pouze jednosměrně. Poté je vytvořena tzv. analyzační signatura, která je založena na anomáliích nalezených v síťové a transportní vrstvě. Není využita žádná znalost aplikačních protokolů.

V druhém kroku jsou uložené toky porovnány mezi sebou pomocí LCS[9] algoritmu ve dvou směrech (horizontálním a vertikálním) a nalezené vzory bajtů jsou přidány do signatury. Porovnání v horizontálním směru znamená, že zpráva je srovnána s každou zprávou z ostatních spojení, která se nachází ve stejném stavu. Vertikální detekce provádí porovnání zpráv z jednoho spojení s odpovídajícími zprávami jiných spojení. To zajišťuje detekci vzorů v interaktivních relacích na rozdíl od horizontální detekce [14]. Proces generování signatur Honeycombem je znázorněn na následujícím obrázku 4.1.



Obrázek 4.1: Proces generování signatury Honeycombu [14]

Příklad signatury:

```
alert udp any any -> 192.168.169.2/32 1434 (msg: "Honeycomb Fri Jul 18
11h46m33 2003 "; content: "|04 01 01 01 01 01 01 01 01 |01|p|AE|B|90 90 90
90 90 90 90 90|h |DC C9 B0|B|B8 01 01 01 01|1|C9 B1 18|P|E2 FD|5 |01 01 01
05|P|89
E5|Qh.dllhel32hkernQhounthickChGetTf|B9|llQh32.dhws2_f|B9|etQhsockf|B9|toQ
hsend|BE 18 10 AE|B|8D|E|D4|P|FF 16|P|8D|E|E0|P|8D|E|F0|P|FF 16|P|BE 10 10
03 |Q|8D|E|CC|P|8B|E|C0|P|FF 16|j|11|j|02|j|00|8D|E|C4|P|8B|E|C0|P|FF 16
```

[9] Longest Common String – algoritmus pro nalezení nejdelšího společného řetězce (podřetězce) v množině řetězců.

```
89 C6 09 DB 81 F3|<a|D9 FF 8B|E|B4 8D 0C|@|8D 14 88 C1 E2 04 01 C2 C1 E2
F1|x|01|Q|8D|E|03|P|8B|E|AC|P|FF D6 EB|"; )
```

Text 4.2: Vytvořená signatura metody Honeycomb

Zobrazená signatura nejdříve obsahuje akci, která má být při úspěšné detekci malware vykonána. V tomto případě bude vygenerován alarm. Dále je uveden název protokolu, na kterém probíhal škodlivý provoz. Následuje pravidlo, které určuje pro jakou zdrojovou adresu a port (v uvedeném případě oboje libovolné) a cílovou adresu a port má být porovnání provedeno. Za tímto pravidlem je uvedena zpráva, která má být v případě úspěšné detekce zobrazena. Posledním parametrem je charakteristický řetězec, který identifikuje škodlivý kód.

4.1.2 Polygraph

Polygraph je systém specificky zaměřený na generování signatur pro polymorfní červy[10]. Klasifikátor monitoruje všechny toky procházející přes jeho senzory a třídí je podle portů. Funkce klasifikátoru toků není blíže specifikována. Vstupem metody je síťový provoz a výstupem můžou být různé typy signatur, které jsou popsány dále.

Vstup:

Síťový provoz.

Výstup:

Konjunktní signatura: neuspořádaná množina tokenů.
Token-subsequence signatura: uspořádaná množina tokenů.
Bayesova signatura: množina tokenů s přiřazenými hodnoceními.

Generující mechanizmus:

Polygraph vytváří tři různé typy signatur, které jsou speciálně navržené pro detekci polymorfních červů. Všechny tři signatury jsou vytvořeny z podřetězců, nazývaných v tomto algoritmu jako tokeny. Tokeny jsou ve své podstatě bajtové sekvence delší než definované minimum. Každý tok je reprezentován množinou tokenů, které se v něm nejčastěji vyskytují.

Konjunktní signatura je množina neuspořádaných tokenů. Pokud signatura a monitorovaný tok obsahují stejné tokeny v různém pořadí, jsou vyhodnoceny jako ekvivalentní. Konjunktní signatura je vytvořena jednoduchou extrakcí tokenů, které se vyskytují ve všech tocích.

Token-subsequence signatura je množina uspořádaných tokenů. Tok je shodný se signaturou, pokud obsahuje stejnou množinu tokenů ve stejném pořadí. Obecná token-subsequence signatura pro množinu toků je nalezena aplikací LCSeq[11] algoritmu na všechny signatury toků, které se nacházejí v této množině.

Bayesova signatura je množina tokenů s přiřazeným skóre, které je odvozeno od Bayesova teorému[12]. Tok je porovnán se signaturou na základě součtu skóre tokenů, které se v toku vyskytují. Pokud je součet větší než nastavený práh, je tok označen jako škodlivý. Tvorba signatury je založena na pravděpodobnosti, že se v červu vyskytuje určitý token [15].

[10] Polymorfní červi se snaží maskovat svou detekci tím, že periodicky mění svůj binární kód.
[11] Longest Common Subsequence – algoritmus pro nalezení nejdelší společné subsekvence v množině sekvencí.
[12] Jedná se o matematickou funkci, založenou na vztahu pravděpodobnosti jevů A a B a jejich podmíněných pravděpodobnostech <http://en.wikipedia.org/wiki/Bayes'_theorem>.

Příklady signatur:

Název signatury	Příklad
Konjunktní signatrura	‘GET ’, ‘ HTTP/1.1\r\n’, ‘: ’, ‘\r\nHost: ’, ‘\r\n’, ‘: ’, ‘\r\nHost: ’, ‘\xFF\xBF’, ‘\r\n’
Token-subsequence signatura	GET .* HTTP/1.1\r\n.*: .* \r\nHost: .* \r\n.*: .*\r\nHost: .*\xFF\xBF.*\r\n
Bayesova signatura	‘\r\n’: 0.0000, ‘: ’: 0.0000, ‘\r\nHost: ’: 0.0022, ‘GET ’: 0.0035, ‘ HTTP/1.1\r\n’: 0.1108, ‘\xFF\xBF’: 3.1517. Threshold: 1.9934

Tabulka 4.1: Vytvořené signatury metody Polygraph

Všechny signatury vytvořené metodou Polygraph obsahují tokeny, které jednoznačně identifikují škodlivý provoz. V případě Bayesovi signatury je těmto tokenům přiřazeno skóre a je uveden práh, při jehož překročení má být generován alarm.

4.1.3 Earlybird

Earlybird nabízí automatický přístup pro rychlou detekci dosud neobjevených červů a virů. Hlavní myšlenkou tohoto přístupu je výpočet otisku pro všechny možné podřetězce příchozího síťového paketu. Každý otisk slouží jako vstup hashovací funkci spolu s protokolem a cílovým portem. Tento hash slouží jako index do takzvané content prevalence tabulky, která obsahuje počet výskytů pro určitou hash hodnotu. Druhá tabulka zvaná address dispersion ukládá počet stejných zdrojových a cílových IP adres pro každou hash hodnotu. Pokud seřadíme content prevalence tabulku podle počtu výskytů a vezmeme v potaz odpovídající záznam v address dispersion tabulce, dostaneme pravděpodobnou množinu červem generovaného provozu. Tento přístup je nazýván content sifting.

Vstup:

Síťový provoz.

Výstup:

Signatura ve formátu pro IDS Snort.

Generující mechanizmus:

Systém generuje signatury formátované pro IDS Snort, podobně jako Honeycomb. Tyto signatury obsahují informace o transportním protokolu a portu. Jelikož algoritmus content sifting neudržuje stav o daném toku, vygenerované signatury popisují pouze informace o obsahu jednotlivých paketů [16].

Příklad signatury:

```
drop tcp $HOME_NET any -> $EXTERNAL_NET 5000 (msg:"2712067784 Fri May 14
03:51:00 2004"; rev:1; content:"|90 90 90 90 4d 3f e3 77 90 90 90 90 ff 63
64 90 90 90 90 90|";)
```

Text 4.3: Vytvořená signatura metody Earlybird

Popis formátu signatury IDS Snort byl uveden v kapitole 4.1.1.

4.1.4 Autograph

Autograph je systém pro automatické generování signatur škodlivých červů. Systém si udržuje seznam podezřelých TCP toků. Pokud počet toků pro dané cílové porty překročí nastavenou mez, je spuštěn proces generování signatury. Autograph měří frekvenci nepřekrývajících se podřetězců v těle paketů všech podezřelých toků. Nejčastěji se vyskytující podřetězce navrhuje jako kandidáty pro zanesení do signatury. Tento algoritmus je nazýván jako content-based payload partitioning (COPP).

Vstup:

Síťový provoz směřující do sítě.

Výstup:

Signatury ve formátu pro IDS Bro.

Generující mechanizmus:

Výše zmínění kandidáti pro signaturu jsou filtrováni a je v nich vyhledáván možný škodlivý obsah. Nejčastěji se opakující část obsahu je poté zvolena jako obsah signatury. Tento proces se opakuje pro zbylé toky, dokud není nalezena část obsahu ze všech toků obsažených v seznamu podezřelých. Nakonec je množina vybraných vzorků uložena do formátu signatury IDS Bro [17].

Příklad signatury:

```
signature my-first-sig {
      ip-proto == tcp
      dst-port == 80
      payload /.*root/
      event "Found root!"
}
```

Text 4.4: Vytvořená signatura metody Autograph

Signatura nejdříve obsahuje její unikátní název a poté jsou uvedeny parametry, které musí být splněny pro provedení dané akce. V tomto případě se jedná o útok vedený přes protokol TCP na službu HTTP (port 80), který obsahuje charakteristický řetězec /.*root/. Poslední uvedený parametr event udává akci, která má být provedena při splnění předchozích podmínek.

4.1.5 TaintCheck

TaintCheck používá dynamickou analýzu označených dat k ochraně určených aplikací. Jakmile jsou označená data použita k nepovolenému účelu porušujícímu nastavenou ochranu (většinou buffer overflow), je spuštěn proces generování signatury.

Vstup:

Síťový provoz. Může být nakonfigurován, aby přijímal data ze standardního vstupu nebo ze zvolených souborů.

Výstup:

Tříbajtový signaturní řetězec.

Generující mechanizmus:

Výsledný tříbajtový řetězec je odvozen od nejvíce signifikantních bajtů, které byly použity k přepsání návratové hodnoty funkce nebo ukazatele na funkci originálního obsahu, například

síťová data jsou porovnána s obsahem paměti. Pokud původní obsah a dané tři bajty nesouhlasí, byla pravděpodobně v intervalu mezi zadáním vstupních dat a detekcí útoku provedena nějaká modifikace dat. V tomto případě je pro tvorbu signatury použit původní obsah. Nevýhodou této metody je velikost vytvořené signatury. Tříbajtový výstupní řetězec je velmi krátký. Pokud budeme uvažovat uniformní rozložení bajtů, bude docházet k falešné detekci jednou za 16MB zpracovaných dat [18].

4.2 Flexibilnější content-based signatury

Přístupy popsané v této kapitole pracují s řetězci bajtů podobně jako metody popsané v předchozí kapitole. Jsou však flexibilnější, jelikož se nepokouší pouze porovnat řetězce nebo podřetězce v příchozích paketech. Signatury generované těmito technikami popisují vzory, které berou v úvahu uspořádání škodlivých bajtů v paketu. Některé techniky pro tento účel používají například regulární výrazy.

4.2.1 PAYL

Metoda PAYL monitoruje anomálie v příchozím provozu a pokud nalezne podezřelý provoz na určitém portu, kontroluje PAYL i odchozí provoz na ten samý cílový port. Výsledné signatury jsou poté vytvářeny pomocí korelace získaných dat.

Vstup:

Vstupní a výstupní provoz na stanici.

Výstup:

Jeden nebo více řetězců.

Generující mechanizmus:

PAYL nejdříve během učící se fáze nastaví tzv. normální profil pomocí n-gramů. N-gram se skládá ze sekvence n po sobě jdoucích bajtů v těle paketu. Jakmile přijde nový paket, jsou vypočítány všechny možné n-gramy a jejich frekvence výskytu. Poté je pomocí vzorce vypočítána vzdálenost mezi příchozími pakety a rozložením n-gramů, které byly vypočítány během učící se fáze. Pokud je tato vzdálenost větší než nastavená mez, jsou tyto pakety umístěny do zásobníku podezřelých paketů pro daný příchozí port. Jakýkoliv odchozí provoz na stejný port je porovnán s pakety uloženými v zásobníku. Pro porovnané řetězce je vypočítáno skóre podobnosti podle určeného vzorce, který zahrnuje výpočet LCS a LCSeq. Pokud je výsledné skóre vyšší než nastavený práh, je odchozí provoz vyhodnocen jako škodlivý a je zablokován [19].

Příklad signatury:

```
|d0|$@|0 ff|5|d0|$@|0|h|d0| @|0|j|1|j|0|U|ff|5|d8|$@|0 e8 19 0 0 0 c3
ff|%`0@|0 ff|%d0@|0 ff|%h0@|0 ff|%p0@|0 ff|%t0@|0 ff|%x0@|0 ff|%| 0@|fc fc
fc fc fc fc fc fc fc fc fc fc fc fc
fc fc fc fc fc 0 0 0 0 0 0 0 0 0 0 0 0|\EXPLORER.EXE|0 0
0|SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon|0 0 0|SFCDisable|0
0  9d ff ff ff|SYSTEM\CurrentControlSet\Service
s\W3SVC\Parameters\Virtual Roots|0 0 0 0|/Scr ipts|0 0 0 0|/MSADC|0 0|/C|0
0|/D|0 0|c:\,,21 7|0 0 0 0|d:\,,217|fc fc fc fc fc fc fc fc fc  fc fc fc
fc fc fc fc fc fc fc fc fc fc
```

Text 4.5: Vytvořená signatura metody PAYL

Vytvořená signatura obsahuje získaný LCS a LCSeq, tedy nejdelší získané shodné řetězce. V tomto případě se jedná o útok na službu explorer.exe, ke kterému jsou využívány hodnoty v registru systému.

4.2.2 PADS

Metoda PADS (Position-Aware Distribution Signatures) používá k vystopování škodlivých aktivit v lokální síti systému dvou typů honeypotů, přičemž high-interaction honeypot přesměrovává spojení na low-interaction honeypoty. Pokud bude high-interaction honeypot kompromitován, low-interaction honeypoty budou schopny odchytit varianty škodlivých červů. Samotné signatury jsou pak generovány offline způsobem.

Vstup:

Síťový provoz zachycený pomocí systému high-interaction a low-interaction honeypotů.

Výstup:

Anomální signatura: BFD (byte frequency distribution) na každé pozici v signatuře.
Normální signatura: BFD legitimního provozu.

Generující mechanizmus:

Signatura PADS se skládá ze signatur pro normální a anomální provoz. Obě signatury obsahují rozdělení četnosti bajtů (BFD) pro každou pozici v řetězci signatury. Proces generování signatury a identifikace červa v těle paketu jsou si velmi blízké. K pochopení procesu generace signatury je potřeba nejdříve pochopit proces identifikace červa.

Algoritmus PADS nejprve prochází obsah paketu tak, že vytvoří okno o velikosti signatury a postupně ho posouvá. Pro každý posun vypočítá skóre shody π. Toto skóre je vypočítáno pomocí vzorce, který zahrnuje výpočet skóre M pro normální i anomální signaturu. Pokud je výsledné skóre π vyšší než nastavený práh (který je normálně nulový) je předpokládáno, že sekvence nese škodlivý kód.

Pozice okna, která má nejvyšší skóre shody je nazvána signifikantní region. Pro nalezení signatury červa je tento region potřeba, protože výsledná signatura je založena na rozdělení četnosti bajtů (BFD) signifikantních regionů všech variant červa. Pokud jsou známy signifikantní regiony všech variant červa, může být BFD lehce vypočítáno. Pro znalost těchto regionů bychom potřebovali všechny signatury, které však nemáme. Pro vyřešení tohoto problému je použit algoritmus EM[13] (Expectation-Maximization), který aproximuje signifikantní regiony pro všechny červy [20].

4.3 Metody používající kontext a sémantiku aplikace

Metody uvedené doposud vytvářely signatury na základě analýzy řetězce bajtů. Techniky popsané v této části využívají sofistikovanější postupy. K tvorbě signatur používají znalosti aplikačních protokolů a dokáží tak rozhodnout, v jakém stavu se musí aplikace nacházet, aby byl škodlivý kód správně detekován. Tato znalost kontextu přispívá k lepším detekčním schopnostem škodlivého software.

[13] Vysvětlení algoritmu je k nalezení na URL: <http://courses.engr.illinois.edu/ece561/spring08/EM.pdf>.

4.3.1 Nemean

Metoda Nemean automaticky generuje signatury na základě záznamu paketů z honeypotu. Systém se skládá ze dvou hlavních komponent, data abstraction component (DAC) a signature generation component (SGC). DAC normalizuje pakety a provádí agregaci toků seřazením paketů do spojení (více paketů mezi hosty) a relací (více spojení mezi hosty). Nakonec jsou agregované relace normalizovány sadou předdefinovaných specifikací pro odpovídající službu (například pro HTTP nebo FTP).

Vstup:

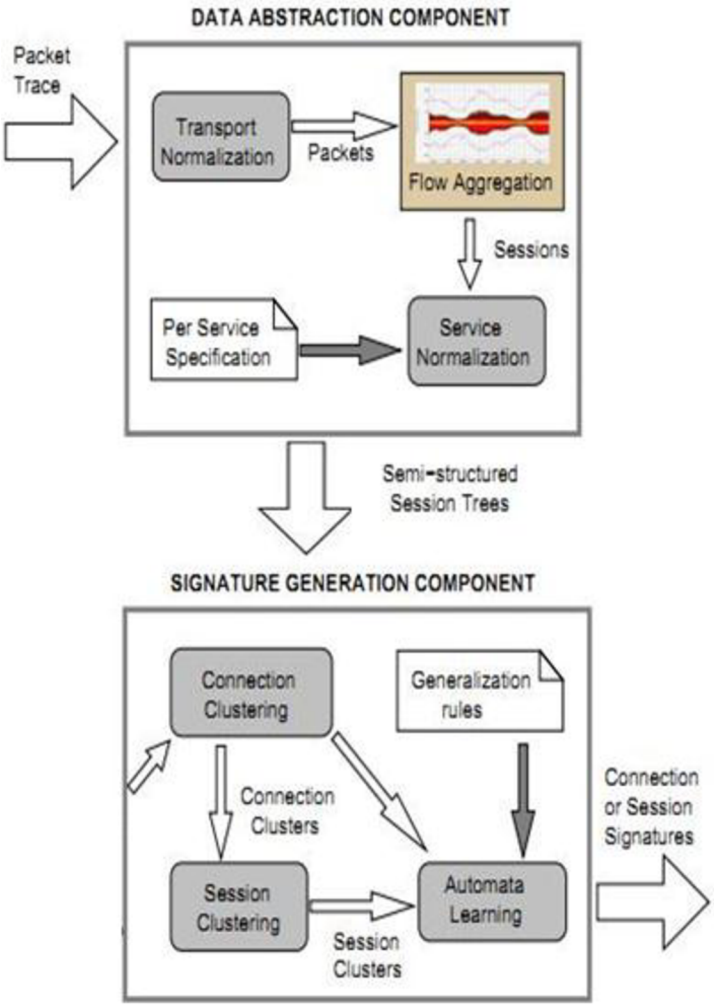
Záznamy paketů z honeynetu.

Výstup:

Regulární výrazy (možnost konverze do IDS Bro signatur).

Generující mechanizmus:

Komponenta SGC shromažďuje relace a spojení na základě metriky podobnosti. Ke generování signatur útoků je použito automatové učení, které získává vstupy ze shluku spojení a relací. Tyto signatury konečných automatů (regulární výrazy) jsou poté transformovány do formátu signatury pro IDS [21]. Pro lepší pochopení mechanizmu je na následujícím obrázku 4.2 zobrazena architektura metody Nemean.



Obrázek 4.2: Architektura systému Nemean [21]

4.3.2 COVERS

COVERS (COntent-based, VulnERability-oriented Signature) systém umožňuje automatické generování signatur pro útoky unesení toku aplikace (control flow hijacking). Systém se skládá ze dvou částí, části pro detekci útoku a části pro generování signatury. Detekce útoku využívá techniku ASLR[14] (address space layout randomization).

Vstup:

Síťový provoz.

Výstup:

Pole přenášející exploit a charakteristiky tohoto pole.

Generující mechanizmus:

První krok identifikuje specifické pakety (nebo tok), které byly součástí útoku a bajty uvnitř těchto paketů, které byly zodpovědné za spuštění varovného hlášení. K identifikaci odpovídajících bajtů je použita forenzní analýza paměti procesu v oblasti poškozeného ukazatele. K tomuto účelu je použit LCS algoritmus, který porovná řetězec posledních vstupních dat a dat uložených v paměti.

V druhém kroku jsou analyzovány pole aplikačního protokolu. K tomuto účelu musí být vytvořeny detailní specifikace protokolu. Poté co jsou jednotlivá pole identifikována podle specifikace, hodnoty těchto polí jsou porovnány s referenčními hodnotami, které jsou průběžně aktualizovány pomocí analýzy neškodného vstupního provozu.

Výsledná signatura obsahuje pole zprávy, které přenáší škodlivý kód a nastavené parametry jeho charakteristik [22].

Příklad signatury:

```
OpenSSL heap overflow:
      { type = 2 , data.size > 420 }

FTP zranitelnost:
      { cmd ="SITE" , params.size > 452 }
```

Text 4.6: Vytvořené signatury metody COVERS

První příklad vygenerované signatury identifikující útok na službu SSL obsahuje číslo pole, přenášející škodlivý kód a parametr udávající podmínku, že přenášená data musí mít délku větší než 420 bajtů. Druhá uvedená signatura představuje zranitelnost služby FTP, konkrétně zneužití příkazu SITE, jehož parametr musí mít pro vygenerování alarmu délku větší než 452 bajtů.

4.4 Ostatní přístupy

Dříve jmenované metody byly klasifikovány jako techniky pro automatizované generování signatur. Techniky uvedené v této části nejsou využiti v žádné automatizované metodě, přesto však přináší zajímavé přístupy a nápady, které by v systému pro automatické generování signatur mohly být v budoucnu použity.

[14] ASLR - bezpečnostní metoda komplikující útočníkovi nalezení adres buněk v paměti.

4.4.1 Paid

Metoda Paid (Program semantics-Aware Intrusion Detection system) chrání aplikace proti útokům zaměřeným na unesení toku aplikace, podobně jako metoda COVERS. Systém za pomoci znalosti zdrojového kódu aplikace analyzuje umístění a pořadí systémových volání spolu s částmi kontrolního toku programu.

Vstup:

Zdrojový kód chráněné aplikace.

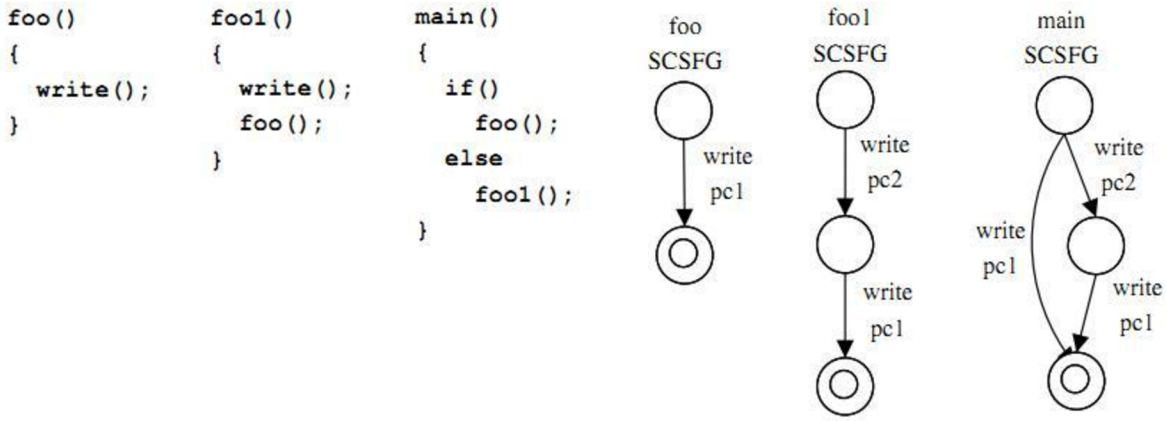
Výstup:

Deterministický konečný automat systémových volání.

Generující mechanizmus:

Proces rekompilace analyzuje použití systémových volání ve zdrojovém kódu aplikace a vytváří graf toku umístění systémových volání (SCSFG – System Call Site Flow Graph), který je obsažen ve výsledné knihovně nebo spustitelném souboru. Tento graf je reprezentován deterministickým konečným automatem, který zobrazuje sekvence systémových volání a jejich umístění v programu [23].

Příklad výstupu:



Obrázek 4.3: Vytvořené grafy toku umístění systémových volání (SCSFG) metody Paid [23]

4.4.2 Vigilante

Vigilante je metoda detekce rychle se šířících červů využívající koncových stanic. Systém představuje koncept tzv. SCA (self-certifying alerts), které obsahují popis útoku. SCA poskytují detailní informace, podle kterých ostatní stanice dokáží ověřit, zda jsou zranitelné. Toto ověření probíhá přehráním SCA zprávy v sandbox[15] prostředí cílové služby.

SCA byly vytvořeny pro 3 časté zranitelnosti. Arbitrary Execution Control (AEC) SCA identifikují zranitelnosti, které dovolují červům přesměrovat tok programu a umožňují tak spuštění libovolných kusů kódu. Arbitrary Code Execution (ACE) SCA popisují zranitelnosti injekce škodlivého kódu do aplikace. Arbitrary Function Argument (AFA) SCA identifikují zranitelnosti injekce dat, které dovolují červům změnit hodnoty argumentů kritických funkcí, jako například systémového volání exec().

[15] Sandbox - prostředí, které je izolované od ostatních procesů. Má vlastní izolované zdroje (paměťový a diskový prostor, přístup k síti, atd.).

Vstup:

Síťový provoz.

Výstup:

Tři typy SCA:

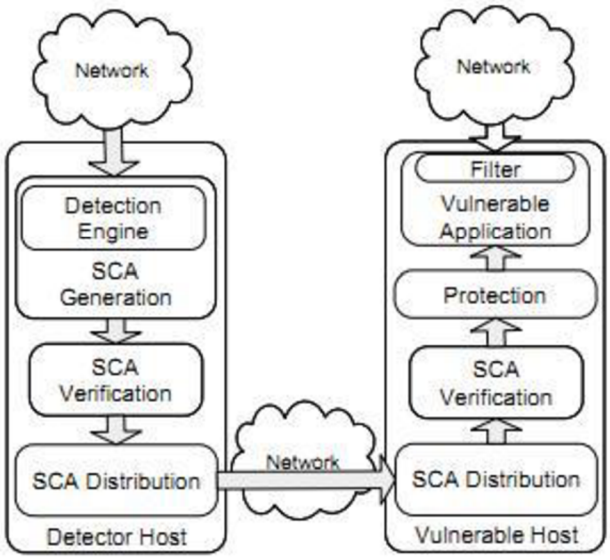
- Arbitrary Execution Control,
- Arbitrary Code Execution,
- Arbitrary Function Argument.

Generující mechanizmus:

Všechny tři typy SCA mají stejný formát a obsahují následující položky:

- identifikaci zranitelné služby,
- typ upozornění,
- ověřující informace,
- sekvenci zpráv pro koncová zařízení, ke kterým musí být zpráva poslána k ověření.

Ověřující informace dovolují koncovému zařízení, kterému jsou zasílány pro potvrzení, vytvořit exploit, jehož úspěch může jednoznačně ověřit sám na sobě (v sandboxu). Ověřující informace jsou pro každý typ upozornění specifické [24].



Obrázek 4.4: Architektura metody Vigilante [24]

Příklad výstupu:

```
Service: Microsoft SQL Server 8.00.194
Alert type: Arbitrary Execution Control
Verification Information: Address offset 97 of message 0
Number of messages: 1
Message: 0 to endpoint UDP:1434
Message data: 04,41,41,41,41,42,42,42,42,43,43,43,43,44,44,44,44,
4A,4A,4B,4B,4B,4B,4C,4C,4C,4C,4D,4D,4D,4D,4E,4E,4E,4E,4F,4F,
50,50,50,50,51,51,51,51,52,52,52,52,53,53,53,53,54,54,54,54,
55,55,56,56,56,56,57,57,57,57,58,58,58,58,0A,10,11,61,EB,0E,…
```

Text 4.7: Výstup metody Vigilante

Výstup metody Vigilante nejdříve obsahuje informaci o názvu zranitelné služby. Poté je uveden jeden ze tří typů alarmu (AEC, ACE, AFA), který způsobil daný útok následovaný ověřovací informací, na které adrese v paměti došlo k útoku a ve které zprávě se tato adresa nalézá. Dále je uveden celkový počet zaslaných zpráv a jsou vypsány jednotlivé zprávy. Každá zpráva obsahuje informaci o tom, jestli byla příchozí nebo odchozí, jaký protokol byl pro komunikaci využit a jaký byl cílový port. Posledním údajem je obsah dané zprávy identifikující škodlivý kód.

4.4.3 DOME

Metoda DOME (Detection Of Malicious Executables) umožňuje detekovat útoky využívající injekci kódu a útoky pocházející ze spustitelných souborů s modifikovaným kódem. Detekční mechanizmus je založen na faktu, že škodlivé kódy často používají systémová volání. Aby DOME identifikoval pozici systémových volání, provádí statickou analýzu spustitelného kódu a kontroluje, zda se tyto pozice v čase běhu nemění.

Vstup:

Spustitelný soubor.

Výstup:

Seznam systémových volání.

Generující mechanizmus:

Přestože metoda DOME negeneruje žádné signatury popisující škodlivé aktivity, dokáže tyto aktivity rozpoznat pomocí porovnání dané aktivity se signaturou popisující normální nebo schválené chování [25].

Příklad výstupu:

```
0140ACA9 SetCurrentDirectoryA
0140A544 FindFirstFileA
013F7616 FindNextFileA
...
01408550 GetLogicalDriveStringsA
013F7485 GetDriveTypeA
0140ACA9 SetCurrentDirectoryA
0140A544 FindFirstFileA
013F7616 FindNextFileA
```

Text 4.8: Výstup metody DOME

Výstup metody DOME obsahuje seznam postupně volaných metod, obsažených v identifikovaném škodlivém software. Tento příklad zobrazuje část seznamu jednotlivých volání Win32 API (application programming interface), které se snaží vyhledat a spustit škodlivý kód uložený v souborovém systému.

4.4.4 HoneyStat

HoneyStat je systém kombinující metody detekce útoku na síti a na koncové stanici. Dokáže emulovat velké množství různých operačních systémů, ve kterých je schopen detekovat tři rozdílné typy událostí:

- paměťové,
- síťové,
- diskové.

Vstup:

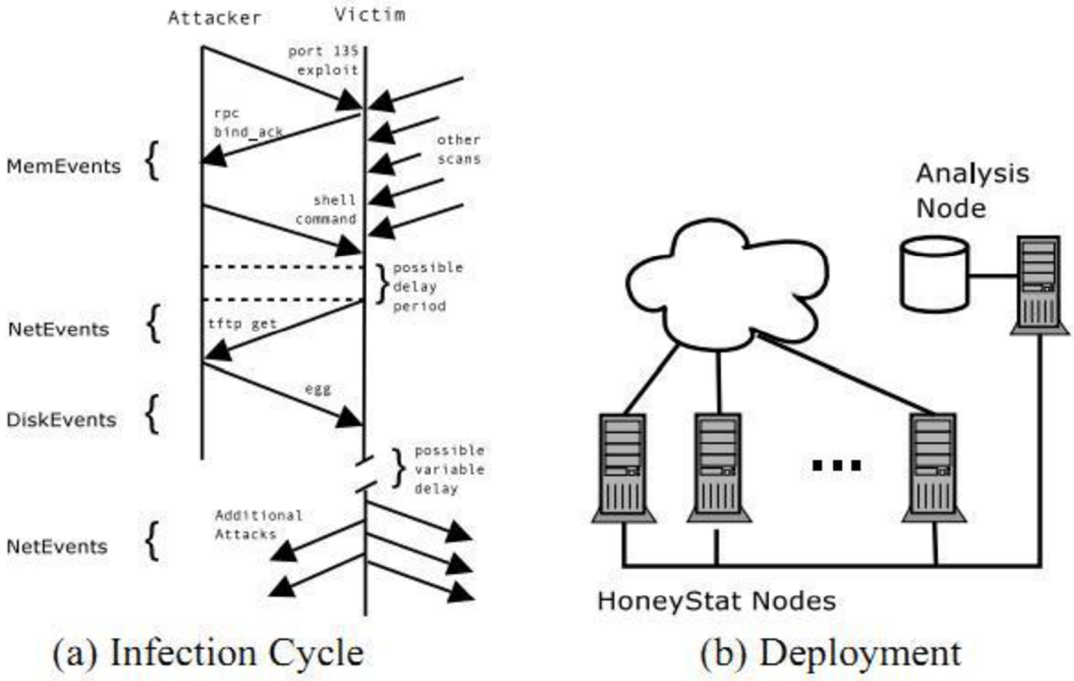
Paměťové, síťové nebo diskové události.

Výstup:

Korelované vstupní události.

Generující mechanizmus:

Systém negeneruje žádné signatury, ale ukládá samotné informace o vzniklých událostech. Sesbírané informace jsou přeposílány do centrálního analyzačního uzlu, který koreluje informace ze všech přijatých událostí [26].



Obrázek 4.5: Cyklus nákazy červem v kontextu HoneyStat a příklad nasazení této metody [26]

5 Architektura systému AIPS

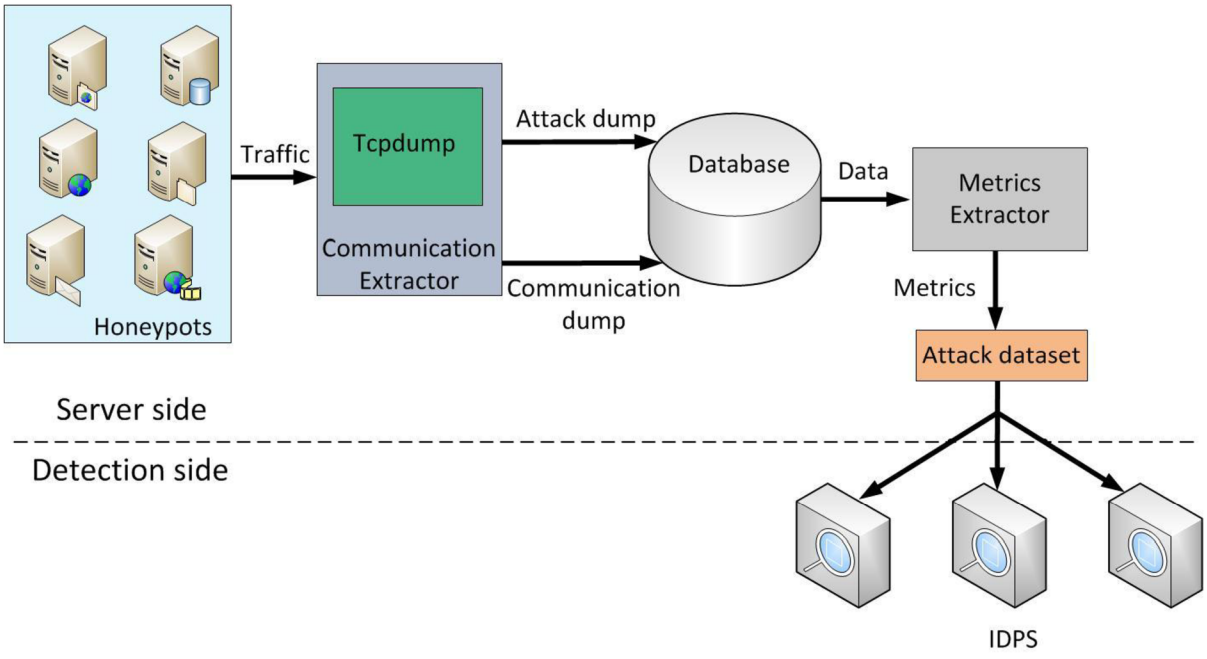
Tato kapitola se týká projektu AIPS (automatic intrusion prevention system), který je vyvíjen v rámci výzkumného projektu na FIT VUT Brno skupinou Buslab, zaměřenou na bezpečnost. Projekt se zabývá automatizovaným rozpoznáváním a zpracováváním počítačových útoků na základě jejich chování. Pro sběr vzorků odchyceného malware je využíván shadow honeypot Argos, popsaný dále v této kapitole, který dokáže zaznamenat útoky zneužívající bezpečnostní chybu buffer overflow. Z odchycených vzorků jsou automaticky vytvářeny signatury, které obsahují soubor metrik identifikující zachycený síťový provoz. V současné době jsou vytvářeny metriky pouze pro provoz používající spojovaný protokol TCP.

Hlavním cílem diplomové práce je vytvoření nástroje, který bude porovnávat metriky získané ze serverové části systému AIPS s metrikami probíhajícího síťového provozu na koncové stanici nebo v síťové infrastruktuře. Metriky lze zpracovávat dvěma způsoby. Prvním způsobem je tzv. automatizované učení, které na základě znalosti dřívějších útoků dokáže rozpoznat nový útok bez znalosti jeho signatury. Druhou variantou je mnou použitá detekce nových útoků na základě znalosti jejich chování, tzv. behaviorální analýza.

Tato kapitola obsahuje popis existující serverové části systému AIPS spolu s návrhem vytvářené detekční části. Vytvářený nástroj jsem vytvořil ve dvou verzích, které odpovídají možným případům reálného nasazení. Návrh těchto verzí je popsán v kapitole 5.3 Návrh systému pro využití v síťových sondách resp. 5.4 Návrh nástroje pro využití na koncových stanicích.

5.1 Serverová část systému

Základní funkčností serverové části systému AIPS je odchytávání nově vzniklých útoků (zero-day) pomocí honeypotů a následné automatické vygenerování odpovídající signatury. Vytvořená signatura je distribuována do klientských IDPS zařízení, které mají za účel její porovnání s probíhajícím síťovým provozem. Architektura celého systému AIPS je zobrazena na následujícím obrázku 5.1.



Obrázek 5.1: Architektura systému AIPS

5.1.1 Honeypoty

Honeypoty jsou důležitou součástí celé architektury systému pro automatické generování signatur resp. systému automatického zpracovávání útoků. Jejich hlavním účelem je emulace různých operačních systémů poskytujících zranitelné služby, na které může útočník nebo automatizovaný malware provést útok.

V systému AIPS je využita architektura tzv. shadow honeypotů. Shadow honeypot je nástroj, který sdílí interní stav s určitou produkční aplikací. Pokud je na tuto službu veden útok, je pomocí mechanizmů implementovaných v shadow honeypotu zastaven a zaznamenán. Informace o útoku jsou poté předány IDPS systémům v téže síti, které zajistí, aby byl daný útok při dalším výskytu okamžitě zablokován. V případě, že aplikaci dojdou validní data, jsou shadow honeypotem neovlivněna a aplikace je tak může zpracovat [27].

Jako shadow honeypot je v systému AIPS použit nástroj Argos, který byl navržen jako nástroj pro automatizované zachytávání nových útoků (zero-day) a generování jejich signatur. Je založený na virtualizačním software QEMU[16], díky kterému dokáže Argos emulovat kompletní operační systém se všemi jeho službami. Pro včasné rozpoznání útoku obsahuje nástroj vrstvu, která analyzuje síťová data a sleduje jejich použití v operační paměti. Argos se zaměřuje na síťové útoky zneužívající některé ze zranitelností kódu, především přetečení zásobníku (buffer overflow). Výhodou Argosu je, že se nezaměřuje na payload[17], ale pouze na pakety, které vykonaly vlastní exploit. Tato technologie zaručí vygenerování stejné signatury například pro polymorfní červy, kteří pro snížení úspěšnosti jejich detekce často mění svůj payload [28].

V případě, že útočník napadne zranitelný systém a způsobí přetečení vyrovnávací paměti (buffer overflow), je jeho pokus detekován a zaznamenán honeypotem v reálném čase. Síťový provoz uložený nástrojem tcpdump[18] spolu s časovým razítkem útoku a paketem, který způsobil přetečení zásobníku (detekovaný Argosem) jsou zasílány na Communication Extractor, kde jsou analyzovány. Ze všech těchto údajů jsou vybrány pouze relevantní pakety, které jsou vloženy do databáze pro další zpracování [29].

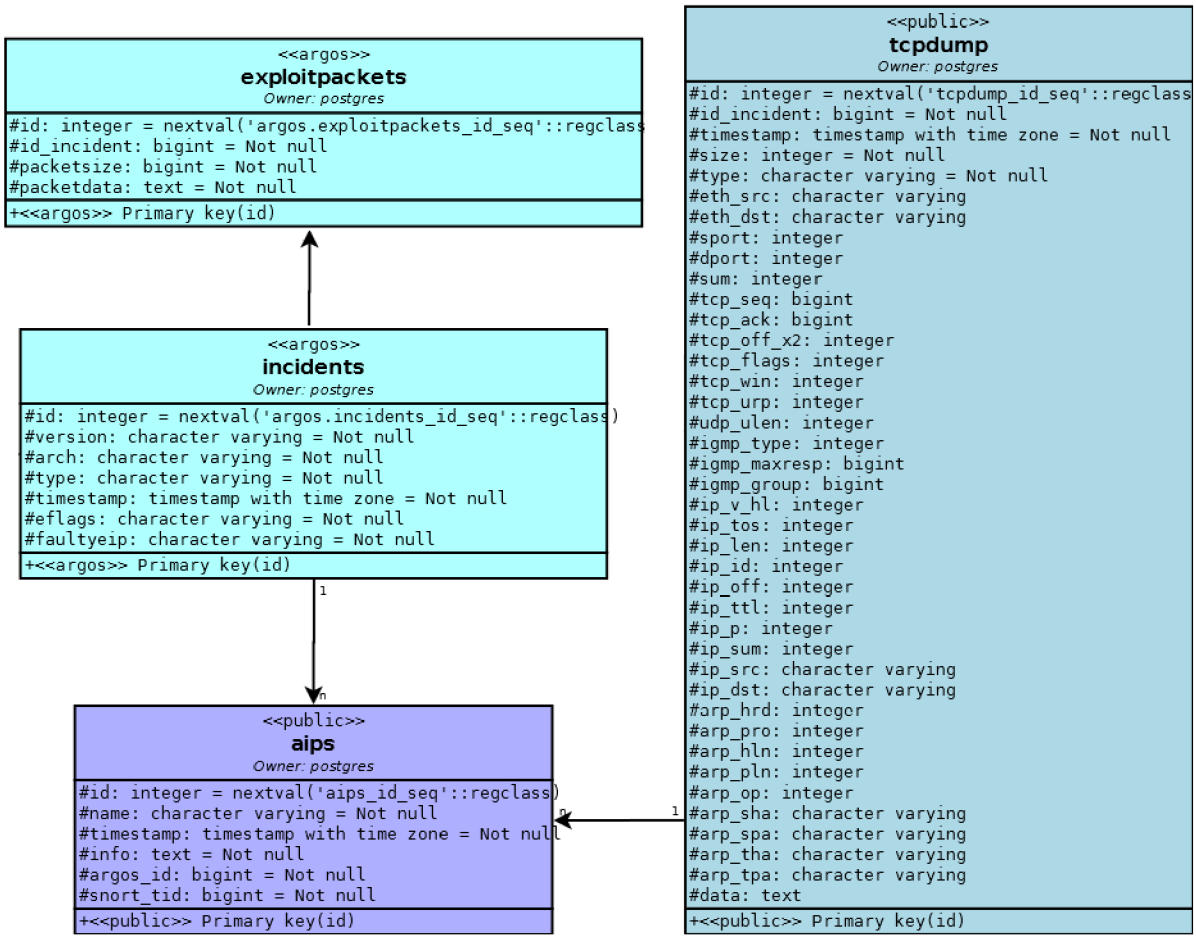
5.1.2 Databáze

Pro účely ukládání získaných dat byl zvolen databázový systém PostgreSQL. Důležité databázové tabulky využívané pro ukládání dat o vzniklých incidentech jsou zobrazeny na obrázku 5.2: Důležitá struktura databáze na serverové části systému AIPS [27].

Zobrazená část databáze se skládá ze čtyř hlavních tříd. První třída *aips* představuje most mezi jednotlivými subsystémy. Spojuje Argos, tcpdump a další detekční systémy jako je například IDS Snort. Data z Argosu jsou uložena ve třídě *incidents* a *exploit packets*.

V případě, že je detekován nový útok, je tento útok zaznamenán jako incident s jedinečným identifikačním číslem, časovou známkou a dalšími důležitými vlastnostmi. Honeypot zaznamená paket, který způsobil buffer overflow a přidá jej k datům incidentu. Subsystém manipulující s tcpdump daty zároveň zaznamená celý TCP provoz spojený s incidentem. Systém AIPS v současné době pracuje pouze s TCP komunikací. V budoucnu je plánována implementace ostatních protokolů třetí a čtvrté vrstvy [29].

[16] QEMU - open-source emulační a virtualizační nástroj <http://wiki.qemu.org/Main_Page>.
[17] Payload - kód vykonaný po úspěšné exploitaci.
[18] Tcpdump - analyzátor síťových paketů. Popsán v kapitole 6.2.1 Tcpdump.



Obrázek 5.2: Důležitá struktura databáze na serverové části systému AIPS [27]

5.1.3 Metrics Extractor

Nástroj Metrics Extractor je v současné době ve stádiu vývoje, o který se starají ostatní členové řešitelského týmu projektu AIPS. Hlavním úkolem tohoto modulu je ze získaných data uložených v databázi vytvořit metriky, které podrobně popisují chování malware spolu s behaviorálními charakteristikami útoku na síťové a transportní vrstvě. V současné době je implementováno 168 metrik, které dohromady tvoří tzv. Attack dataset. Metriky jsou rozděleny podle jejich povahy do pěti kategorií stručně popsaných v následující tabulce 5.1.

Typ metriky	Počet	Popis
behavioral	43	Behaviorální metriky jsou založené na základě popisu vlastností přímo spojených s chováním útoku. Příkladem může být legální či vynucené ukončení spojení, počet toků v definovaných časových intervalech, polynomiální aproximace délky paketů, polynomiální aproximace součtu paketů a podobné informace, které se přímo vztahují k exploitaci zranitelné služby.
dynamic	33	Dynamické metriky představují dynamické chování sítě jako například rychlost, počet bajtů/paketů v odchozím či příchozím provozu atd. měnící se v čase.

localization	8	Lokalizační metriky se používají k určení polohy zdrojů a vyhodnocení stop útoku. Jejich cílem je poskytnout argumenty v rozhodovacím procesu data-mining enginu.
static	50	Statické metriky definují vlastnosti útoku z hlediska statických vlastností, jako je například množství přenesených dat, počet toků, průměrná velikost paketu, počet fragmentovnaných paketů atd.
vector	34	Vektorové metriky jsou definovány jako uspořádané n-tice. Každá hodnota představuje aktuální stav sledované funkce (počet odchozích a příchozích dat, velikost odchozích a příchozích paketů) za jednotku času (vzorkovací frekvence je 1 ms, 5ms, 10ms, 30ms, 50ms a 1s) v měřeném síťovém toku. Počet jednotlivých členů n-tic není stejný a je závislý na vzorkovací frekvenci a době trvání sledovaného toku.

Tabulka 5.1: Typy metrik a jejich stručný popis [27]

Informace o architektuře modulu Metrics Extractor spolu s detailním popisem jednotlivých metrik je k nalezení v [30].

5.1.4 Attack dataset

Vytvořené metriky jsou uloženy do souboru ve formátu CSV (Comma-Separeted Value), ve kterém jsou distribuovány do klientských IDPS zařízení. Attack dataset má v aktuální podobě následující formát (zobrazeno v tabulkovém formátu, ne ve formátu uložení do souboru):

id	class label	distributed	distributed	...
id	label	InPkt1s10i[0]	InPkt1s10i[1]	...
1	False	0	1	...
2	True	1	3	...
...	...	...	...	...

Tabulka 5.2: Testovací formát souboru attack dataset

První dva řádky stejně jako první sloupec obsahují pouze pomocné informace, přičemž první řádek obsahuje typ jednotlivé metriky, druhý řádek její název a první sloupec ID jednotlivých zaznamenaných toků. Druhý sloupec obsahuje hodnotu True nebo False v závislosti na tom, zda se v konkrétním toku vyskytoval paket, který způsobil zneužití zranitelnosti buffer overflow. Ostatní buňky obsahují hodnoty příslušných metrik.

5.1.5 IDPS

Komponenta označená v architektuře systému jako IDPS má sloužit k detekci a případné prevenci proti identifikovaným útokům v síťovém provozu. Tato komponenta není doposud v systému AIPS implementována a její návrh a implementace je hlavním tématem této diplomové práce.

5.2 Detekční část – nástroj Attack Detector

Na základě potřeb projektu AIPS bylo mým úkolem vytvoření doposud neexistující detekční části systému. Vytvořený nástroj bude monitorovat probíhající síťový provoz, ve kterém bude vyhledávat

vzorky odchyceného malware. Nástroj bude zasazen do existující architektury systému AIPS a bude sloužit pouze pro detekci (nikoliv prevenci) zaznamenaných útoků. Za účelem jeho testování budou použita vytvořená data získaná ze serverové části systému.

Nástroj jsem se rozhodl navrhnout ve dvou verzích, které odpovídají reálným případům užití. První verze může být nasazena na síťové sondy, které pracují s velkým objemem dat a jejichž požadavky na rychlost jsou vysoké. Použití druhé navržené verze je plánováno na koncové uživatelské stanici, kde množství síťového provozu není velké.

5.2.1 Požadavky na vytvářený nástroj

Hlavním požadavkem na nástroj bylo vytvoření prvotní implementace, která dokáže zachytávat síťový provoz v určitém časovém intervalu, automaticky z něj generuje metriky a následně vytvořené metriky porovná s testovacími metrikami získanými ze serverové části systému AIPS. Funkční požadavky jsou uvedeny v následujících bodech.

1. Požadavky kladené na funkčnost nástroje plynou především z povahy IDPS systémů. Vytvořený nástroj musí poskytovat vysokou úspěšnost generování tzv. true-positives. Stručně řečeno, nástroj musí korektně detekovat každý známý útok.

2. V souvislosti s předchozím bodem musí být zajištěno generování minimálního počtu falešných poplachů, tzv. false-positives. Zajištění ideálního stavu, tedy zachycení všech útoků při nulovém počtu vygenerovaných false-positives je v reálném prostředí velmi těžké dosáhnout. Každý IDPS systém zpočátku generuje určité procento falešných poplachů. Minimalizace tohoto počtu je ve většině systémů dosaženo tím, že se IDPS systémy takzvaně učí. To vyžaduje zásah operátora, který může vygenerovaný alarm označit jako falešný, čímž se v budoucnu sníží procento vygenerovaných false-positives. Učící se systém nebyl předmětem implementace vytvářeného nástroje.

3. Dalším požadavkem byla funkčnost nástroje na rozdílných platformách. Jelikož se jedná o IDS systém, který běží v klientském prostředí, musí být zajištěna funkčnost především na operačních systémech Unix (při použití jako síťová sonda) a v prostředí platforem Windows (při použití na koncové uživatelské stanici).

4. Jelikož se jedná o první verzi nástroje, musí být v požadavcích myšleno také na jeho budoucí možnosti rozšíření, proto by měl být systém navržen jako modulární.

5. Další vlastností, kterou by měl navržený nástroj disponovat, je rychlé zpracovávání útoků. V případě systému IDS není rychlost kritickou vlastností, přestože by alarm informující o nalezeném útoku měl být vygenerován v co nejkratším časovém intervalu. V budoucnu je však plánováno využití nástroje jako IPS systému, který musí kvůli případnému zásahu proti útoku pracovat v reálném čase.

6. Nástroj musí být postaven na technologiích, které jsou volně dostupné a jejichž použití je bezplatné pro akademické, soukromé i komerční účely.

5.3 Návrh systému pro využití v síťových sondách

Tato kapitola popisuje mnou navrženou architekturu nástroje, která je plánována pro využití v síťových sondách. Síťové sondy mají za úkol zpracovávat velký objem protékajícího provozu, proto byla tato architektura navržena tak, aby nejnáročnější část výpočtu, tedy samotné porovnání signatur, probíhalo na databázovém serveru. Výhodou tohoto přístupu je možnost vysoké paralelizace, kdy do procesu výpočtu může být zahrnuto několik databázových serverů.

Při návrhu architektury a výběru použitých technologií byly brány ohledy především na požadované vlastnosti vyjmenované v předchozí kapitole. Další důležitou roli při návrhu nástroje hrála existence serverové části, která již používá určité technologie a má implementovánu jistou funkcionalitu. Tento fakt vedl například k výběru programovacího jazyka Python.

Výchozím bodem návrhu byla nutnost použití modulu Metrics Extractor, který na serverové straně slouží k vytvoření metrik ze zachyceného síťového provozu a jehož funkcionalita bude až na drobné změny v detekční části stejná.

S nutností využití modulu Metrics Extractor souvisí použití databázového systému, ze kterého bude modul získávat vstupní data a do kterého budou vkládány vytvořené metriky. Databázový systém bude použit i pro porovnání metrik získaných ze síťového provozu s metrikami zaznamenaných útoků.

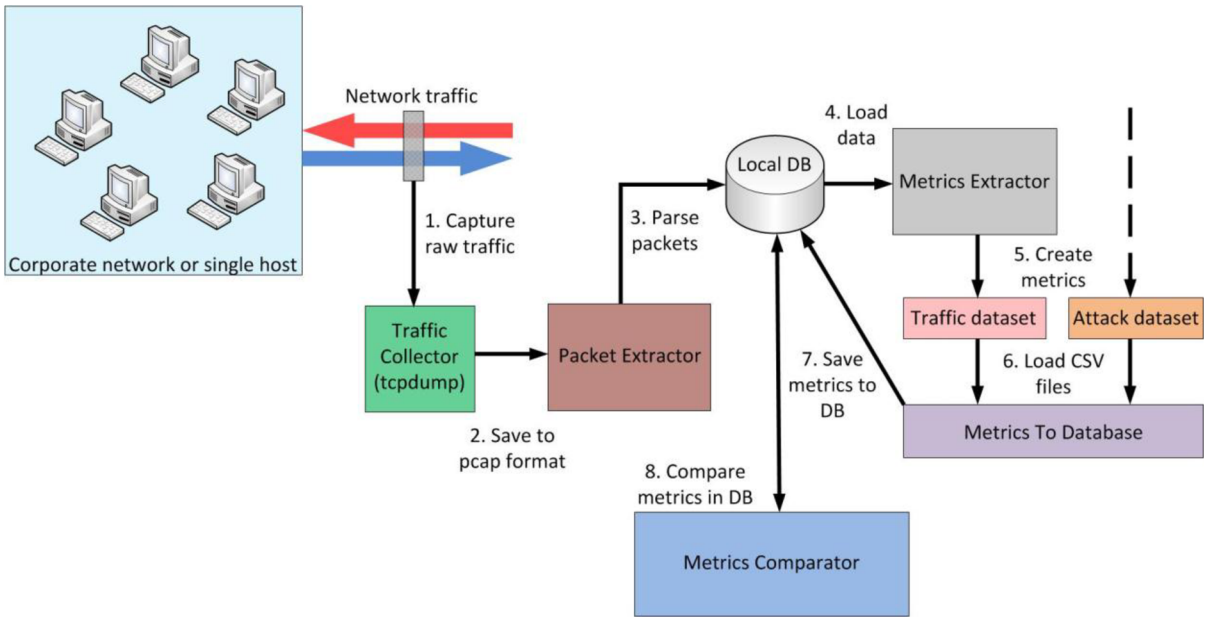
Pro zachycení a ukládání síťového provozu bude vyhledán specializovaný nástroj s požadovanými vlastnostmi. Tento nástroj bude začleněn do modulu Traffic Collector, který bude poskytovat vstupní rozhraní pro spuštění celého nástroje Attack Detector.

Funkcionalita pro rozparsování uložených paketů a následné převedení získaných dat bude samostatně oddělena a bude implementována v modulu Packet Extractor.

Vzhledem k faktu, že modul Metrics Extractor je převzat ze serverové části a není tudíž mým dílem, chtěl jsem do jeho zdrojových kódů zasahovat co nejméně. Pro převod metrik ze souboru do databáze jsem navrhl samostatný modul pojmenovaný Metrics To Database.

Jádro celého nástroje, tedy porovnání vytvořených metrik z procházejícího síťového provozu s metrikami obsahujícími informace o zachycených útocích bude implementováno jako uložená procedura v databázi. Pro spuštění této procedury a zpracování výsledků bude vytvořen modul Metrics Comparator.

Celé navržené schéma architektury nástroje Attack Detector pro využití na síťových sondách je zobrazeno na následujícím obrázku 5.3.



Obrázek 5.3: Architektura nástroje pro použití na síťových sondách

5.4 Návrh nástroje pro využití na koncových stanicích

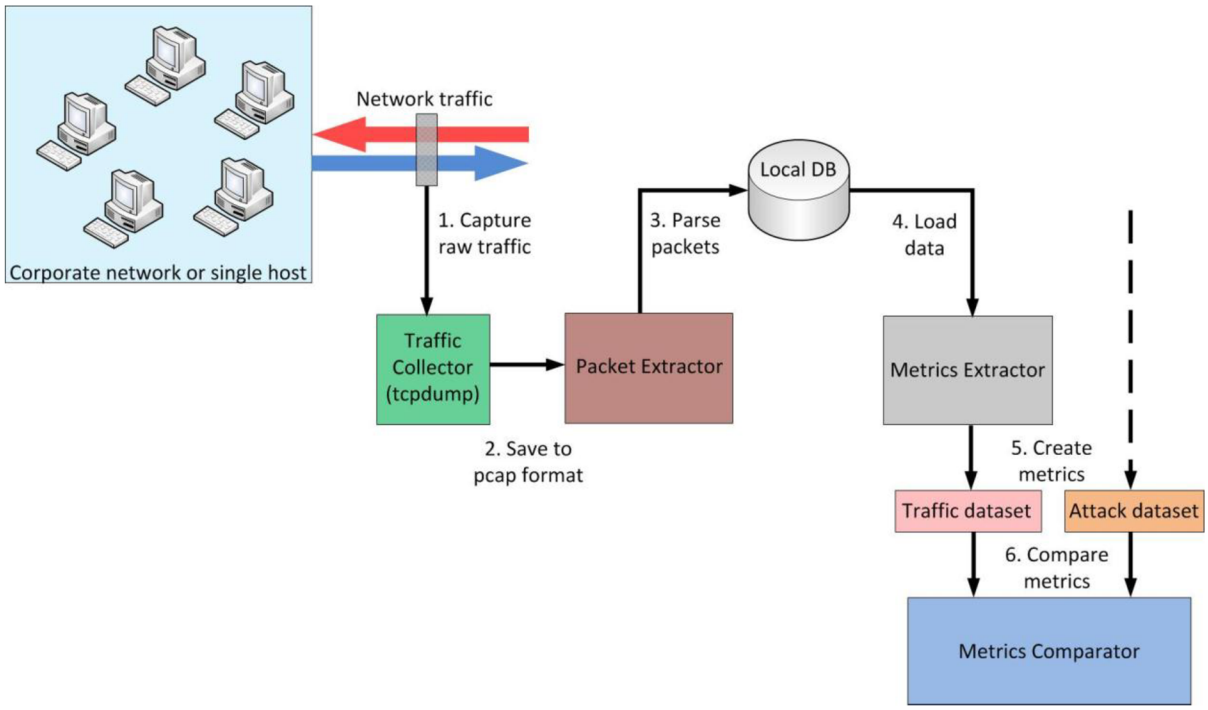
Výhodou architektury popsané v předchozí kapitole je možnost paralelizace výpočtu, který probíhá na straně databázového systému. Na obyčejné koncové uživatelské stanici však nemáme možnost připojení dalšího systému, který by do procesu porovnání metrik mohl být zapojen.

Z tohoto důvodu byl v architektuře pro použití na koncových stanicích odstraněn modul Metrics To Database, čímž byl redukován počet kroků potřebných k získání výsledku z původních osmi na šest. Odstraněním tohoto modulu zároveň odpadá režie (výkonová a časová) nutná k převedení dat ze souboru do databáze.

Hlavním rozdílem je přesunutí algoritmu provádějícího porovnání metrik z databáze do modulu Metrics Comparator, který bude jako vstup přijímat dva soubory obsahující příslušné metriky. Lokálně vytvořený soubor Traffic dataset a soubor Attack dataset získaný ze serverové části.

Lokální databáze bude zachována z důvodů nutnosti jejího využití modulem Metrics Extractor. Databáze bude dále použita pro ukládání logovací informací ze všech implementovaných modulů.

Funkcionalita modulů Traffic Collector a Packet Extractor bude zachována beze změn. Architektura nástroje pro použití na koncových stanicích je zobrazena na následujícím obrázku 5.4.



Obrázek 5.4: Architektura nástroje pro využití na koncových stanicích

5.5 Jednotlivé komponenty navrženého nástroje

Navržený nástroj se skládá z několika navzájem komunikujících modulů. Všechny moduly jsou uvedeny na obrázku 5.3 a jejich navržená funkcionalita je uvedena v následujících kapitolách.

5.5.1 Metrics Extractor

Jak již bylo řečeno dříve, k extrakci metrik pro zachycený síťový provoz bude použit modul Metrics Extractor ze serverové části systému AIPS. Pro jeho korektní běh bude nutné upravit zdrojové kódy tak, aby pracoval se vstupy z lokální databáze, jejíž struktura bude oproti struktuře databáze použité pro ukládání informací o zachycených útocích na serverové části systému zjednodušena.

5.5.2 Lokální databáze

Pro účely ukládání informací o zachyceném síťovém provozu bude na klientském systému nainstalována lokální databáze. Jako databázový systém bude zvolen PostgreSQL a tabulka pro uložení zaznamenaného síťového provozu bude mít stejnou strukturu jako na serverové části. Dále budou vytvořeny dvě tabulky, které budou obsahovat vytvořené metriky převedené ze souborů, a navíc bude databáze obsahovat tabulku pro ukládání všech logovacích informací.

Algoritmus pro detekci útoků v síťovém provozu bude implementován v uložené proceduře CompareMetrics, která bude vytvořena v jazyku Python. Jejím vstupem bude název tabulky obsahující metriky vytvořené ze zachyceného síťového provozu, název tabulky obsahující metriky zachycených útoků a detekční práh, který představuje procentuální hodnotu shody porovnávaných metrik.

5.5.3 Traffic Collector

Modul Traffic Collector bude představovat rozhraní pro spouštění celého nástroje Attack Detector. Jeho vstupem budou všechny parametry potřebné pro běh nástroje jako například název síťového rozhraní, na kterém bude odchytáván probíhající provoz a časový interval, po kterém bude zachycený provoz zpracováván. K odchycení síťového provozu bude v rámci zachování konzistentnosti se serverovou částí použit nástroj tcpdump resp. jeho verze WinDump pro platformu Windows. Výstupem tohoto modulu bude soubor se zaznamenaným provozem, který bude předán modulu Packet Extractor.

5.5.4 Packet Extractor

Po spuštění modul Packet Extractor načte vstupní soubor s uloženým provozem a získá informace o každém v něm obsaženém paketu. K tomuto účelu bude použita knihovna DPKT. Výstupem modulu budou získaná data z předchozího kroku, která budou vložena do lokální databáze. Pro komunikaci s databází bude použita knihovna Psycopg2. Modul následně smaže zpracovaný soubor, aby nedošlo k zaplnění volného místa na disku, a zavolá modul Metrics Extractor, který z uložených dat v databázi vytvoří Traffic dataset. Z důvodů co nejmenšího možného zásahu do zdrojových kódů modulu Metrics Extractor zavolá modul Packet Extractor po úspěšném vytvoření metrik i moduly Metrics To Database a Metrics Comparator.

5.5.5 Metrics To Database

Tento modul bude implementovat převod metrik ze vstupního souboru ve formátu CSV do zadané tabulky v databázi. Vždy před převodem nových dat do příslušné tabulky zajistí odstranění uložených záznamů, starších než definovaný časový interval. To zajistí, že metriky budou vytvářeny vždy pro aktuální provoz a pro provoz zaznamenaný v daném časovém úseku bezprostředně před ním.

5.5.6 Metrics Comparator

Tento modul bude po svém spuštění volat uloženou proceduru v databázi, které předá odpovídající parametry. Vytvořený nástroj bude umožňovat nastavení detekčního prahu, který představuje procento shody porovnávaných metrik. Výstupem modulu budou hlášení o nalezených tocích, které mají procentuální shodu s nějakým škodlivým tokem vyšší než je definovaný detekční práh.

6 Implementace nástroje Attack Detector

Předmětem této kapitoly je popis implementace jednotlivých modulů nástroje Attacke Detector, které byly navrženy a specifikovány v předchozí kapitole. Před začátkem samotné implementace byla provedena analýza, ve které byly vybrány technologie a nástroje, které splňují všechny požadavky uvedené v kapitole 5.2.1 Požadavky na vytvářený nástroj. V úvodu této části práce je popsáno vývojové prostředí a podpůrné prostředky, které byly využity k tvorbě nástroje Attack Detector. Dále jsou uvedeny implementační detaily jednotlivých modulů, ze kterých se nástroj skládá. Pokud modul využívá nějaký nástroj třetí strany, jsou vysvětleny jeho vlastnosti a je uveden důvod, proč byl tento nástroj k implementaci zvolen.

6.1 Vývojové prostředí a podpůrné nástroje

Nástroj Attack Detector byl od začátku vyvíjen ve virtuálním prostředí platformy VMware Workstation 8.0.2[19]. Jako operační systém byla zvolena linuxová distribuce Ubuntu[20] 11.04 (Natty Narwhal) v 64bitové verzi. 64bitová verze operačního systému byla zvolena záměrně a to až poté, co byla vyhledána podpora tohoto systému u všech nástrojů a knihoven, jejichž použití bylo k vývoji nástroje předpokládáno. Tato verze byla zvolena především proto, aby byla zajištěna funkčnost nástroje na systému, který dokáže pracovat s operační pamětí o velké kapacitě (větší, než podporovaných 4GB u 32bitových operačních systémů). Předpokládá se, že nástroj nasazený pro monitorování provozu na velmi rychlých linkách (1Gbit, 10Gbit), bude na hostitelském systému spotřebovávat velké množství prostředků operační paměti.

6.1.1 Python

Python je programovací jazyk, který je používán pro tvorbu široké škály aplikací. Poskytuje jednoduché a rychlé funkce pro zpracovávání textu, programování síťové komunikace, práci s databázovými systémy, webovými technologiemi i grafickými nadstavbami. Jednoduchosti jazyka a jeho použití přispívá existence velkého množství obsažených modulů ve výchozím instalačním balíku. Dalšími přednostmi je snadná práce s pamětí, soubory, regulárními výrazy, spolupráce s ostatními programovacími jazyky a jeho celková jednoduchost syntaxe. Výkon aplikací napsaných v Pythonu je také na velmi dobré úrovni, protože výkonově kritické knihovny jsou implementovány v jazyce C, s kterým Python výborně spolupracuje. Nejen díky těmto vlastnostem má Python velkou uživatelskou základnu, která značnou měrou přispívá k tvorbě volně dostupných knihoven třetích stran [31].

Jazyk Python byl s ohledem na požadavky vytvářeného nástroje zvolen především kvůli následujícím vlastnostem:

- open-source,
- podporované platformy Unix a Windows v 32bitové i 64bitové verzi,

[19] VMware - virtualizační platforma <http://www.vmware.com/products/workstation/>.
[20] Ubuntu - linuxová distribuce <http://www.ubuntu.com/>.

- přijatelná rychlost,
- existence potřebných knihoven třetích stran,
- vysoká rychlost implementace.

Pro implementaci všech modulů nástroje Attack Detector byl zvolen Python verze 2.7.3, přestože již existuje novější verze (aktuálně 3.2.3). Novější verze vyšší než 3.0 nejsou zpětně kompatibilní s verzemi 2.x. Z toho plynou jejich hlavní nevýhody a to stále velmi omezená podpora knihoven třetích stran a fakt, že většina linuxových distribucí používá jako výchozí stále Python verze 2.x.

6.1.2 PostgreSQL

PostgreSQL je objektově-relační databázový systém (object-relational database management system, ORDBMS) vyvinutý na oddělení informatiky University of California v Berkeley. Má za sebou více než 15 let aktivního vývoje, díky kterému získal osvědčenou architekturu. PostgreSQL získal svou dobrou pověst především díky zajištění vysoké spolehlivosti, integrity a korektnosti uložených dat.

Databázový systém má podporu běhu na všech hlavních operačních systémech, včetně Linuxu, Unixu a Windows. Je vyvíjen spolkem PostgreSQL Global Development Group, skládajícího se z hrstky dobrovolníků pracujících pro firmy jako je Red Hat či EnterpriseDB. Licence PostgreSQL, pod kterou je systém vydáván zajišťuje jeho bezplatné použití, úpravu a distribuci a to jak pro soukromé, komerční tak i akademické účely [32].

Jelikož je databázový systém PostrgreSQL použit v serverové části systému AIPS, byl na klientské části zvolen především kvůli zachování konzistentnosti architektury. PostgreSQL splňuje všechny požadavky kladené na výběr lokálního databázového systému:

- rychlý,
- multiplatformní,
- jeho použití je bezplatné,
- umí spolupracovat s Pythonem.

Pro vývoj byla zvolena poslední aktuální verze PostgreSQL 9.1.3. Nástroj by měl být kompatibilní i se staršími verzemi PostgreSQL.

V následujících kapitolách je popsána implementace modulů, které byly uvedeny v obrázku 5.3: Architektura nástroje pro použití na síťových sondách.

6.2 Modul Traffic Collector

Základní funkcí tohoto modulu je ukládání zachyceného síťového provozu na disk, aby mohl být poté extrahován do lokální databáze. K tomuto účelu je použit nástroj tcpdump, který je blíže popsán dále v této kapitole.

Traffic Collector slouží jako vstupní skript ke spuštění celého procesu detekce útoků. Předpokladem pro jeho korektní spuštění je existence nástroje tcpdump v systému. Aby skript mohl nástroj spustit, musí být cesta k spustitelnému souboru tcpdump uvedena v systémové proměnné PATH (ve výchozím nastavení po nainstalování nástroje tomu tak ve většině případů bývá) a uživatel musí mít nastavena oprávnění pro jeho spuštění. Další nutnou podmínkou pro jeho spuštění je existence modulu Packet Extractor ve stejné složce, ze které je modul Traffic Collector spuštěn.

Modul při spuštění požaduje zadání tří povinných přepínačů, jejichž popis a parametry jsou uvedeny v následující tabulce 6.1.

Přepínač	Parametr	Popis
-i	název rozhraní	Název rozhraní, na kterém bude odchytáván síťový provoz.
-t	časový interval	Časový interval, po kterém bude zachycený provoz uložen (v minutách).
-d	složka	Složka, do které se budou ukládat soubory s odchyceným provozem.
-h		Vytiskne nápovědu programu.

Tabulka 6.1: Parametry skriptu Traffic Collector

Skript po kontrole zadaných parametrů nejdříve ověří, zda se ve stejné složce nachází i modul Packet Extractor, potřebný pro běh nástroje Attack Detector. Pokud není modul nalezen, je skript ihned ukončen. V následujícím kroku modul ověří existenci zadané složky pro ukládání zachyceného provozu, a pokud složka neexistuje, automaticky se jí pokusí vytvořit. V dalším kroku je vytvořen název souboru se zachyceným provozem, který má následující formát:

```
hostname_rok_mesic_den_hodina_minuta_sekunda.dump
```

Text 6.1: Název vytvořeného souboru s odchyceným provozem

Uvedený formát názvu souboru je pro funkčnost nástroje irelevantní, sloužil spíše k lepší orientaci při procesu testování. Z uvedeného formátu názvu souboru vyplývá, že skript je naprogramován tak, aby ukládal každý soubor pod jiným názvem, čehož bylo opět hojně využíváno pro kontroly při vývoji nástroje. Tato funkcionalita však v konečném důsledku může vést k vyčerpání veškerého volného místa na disku. Řešení ukládání dat do souboru se stejným názvem není možné, protože by docházelo k přepsání dat ještě dříve, než by byly extrahovány do lokální databáze. Tento problém je řešen až v modulu Packet Extractor, který po úspěšném převedení dat do databáze smaže zpracovaný soubor. Na závěr celého skriptu je spuštěn samotný nástroj tcpdump s následujícími parametry:

```
tcpdump -i iface -U -G mins -w path tcp -z pe
```

Text 6.2: Spuštění nástroje tcpdump v modulu Traffic Collector

V následující tabulce jsou vysvětleny použité parametry nástroje tcpdump.

Parametr	Popis
-G	Ukládá provoz do souboru uvedeného parametrem -w po uplynutí zadaného počtu sekund.
-i	Název síťového rozhraní, na kterém bude tcpdump odchytávat provoz.
tcp	Filtr pro zachytávání pouze TCP provozu.
-U	Zachycený paket je ihned ukládán do souboru a nečeká pro zápis na disk na naplnění bufferu.
-w	Zachycený provoz je uložen do souboru s uvedeným názvem.
-z	Ve spojení s parametrem -G spustí tcpdump ihned po uložení souboru na disk zadaný příkaz.

Tabulka 6.2: Seznam použitých přepínačů, se kterými je nástroj tcpdump spuštěn

6.2.1 Tcpdump

Tcpdump je jeden z nejznámějších paketových analyzátorů (snifferů), který běží v prostředí příkazové řádky. Byl vytvořen již v roce 1987 Van Jacobsonem, Craigem Leresem a Stevem McCannem, kteří v té době pracovali ve výzkumné skupině laboratoří Lawrence Berkley. Nástroj je distribuován pod BSD licencí[21], což z něj dělá bezplatný svobodný software.

Pro odchycení paketů používá tcpdump knihovnu libpcap[22], podobně jako asi nejznámější síťový analyzátor Wireshark[23], který v jádru poskytuje téměř totožné možnosti využití jako tcpdump, ovšem s rozšířením o přehledné grafické uživatelské rozhraní. Tcpdump byl původně vyvinut pro unixové operační systémy (běží bez problémů na Linuxu, Solarisu, BSD, Mac OS X, HP-UX a AIX). Díky své oblíbenosti byl portován i do prostředí Windows, kde se vyskytuje pod názvem WinDump. O nástroji WinDump bude pojednáno v kapitole 6.4 Verze pro platformu Windows.

Hlavním účelem nástroje tcpdump je zachytávání veškerého provozu na určeném síťovém rozhraní. Nástroj dokáže zachytávat síťový provoz v podobě řetězce probíhajících bajtů (tzv. raw traffic), případně ho dokáže transformovat do uživatelsky přívětivější textové formy. Tcpdump obsahuje velké množství přepínačů, pomocí kterých dokážeme síťový provoz například zachytávat v určitém časovém intervalu, třídit podle definovaných filtrů, různě konvertovat jeho výstup a nakonec tento výstup uložit do souboru ve formátu pcap pro pozdější analýzu [33].

Nástroj tcpdump byl zvolen především kvůli vlastnostem uvedeným v následujícím seznamu:

- svobodný software,
- multiplatformní,
- pracuje v příkazové řádce,
- obsahuje široké možnosti pro filtrování a ukládání síťového provozu.

6.3 Modul Packet Extractor

Hlavním úkolem tohoto modulu je převedení zachyceného síťového provozu ve formátu pcap do definované tabulky v databázi. Tento převod je nutný provést kvůli modulu Metrics Extractor, který je vytvořen tak, aby přijímal vstup z databáze.

Skript pro korektní spuštění požaduje zadání jediného povinného parametru, kterým je cesta k souboru s uloženým síťovým provozem. Po kontrole parametru a otevření zadaného souboru se skript pokusí o připojení k databázi, kde vyhledá tabulku obsahující tcpdump data. Název této tabulky je uveden v konfiguračním souboru.

Pro komunikaci s databází je ve všech vytvořených skriptech použita knihovna psycopg ve verzi 2.4.5. Psycopg je PostgreSQL databázový adaptér vytvořený pro programovací jazyk Python. Mezi jeho hlavní rysy patří plná podpora Python DB API 2.0 a bezpečné využití vícevláknového přístupu (vlákna mohou sdílet jedno připojení). Byl navržen pro podporu náročných vícevláknových aplikací pracujících s velkým počtem databázových kurzorů a generujících velké množství souběžných databázových transakcí [34].

Prvním krokem při práci s databází je úprava tabulky udržující extrahované tcpdump data. Struktura této tabulky je k nalezení na obrázku 5.2: Důležitá struktura databáze na serverové části systému AIPS [27]. Důležitou hodnotou je časové razítko zachyceného paketu, podle kterého dochází

[21] BSD licence - licence svobodného software <http://www.linfo.org/bsdlicense.html>.
[22] Libpcap - knihovna pro zachytávání paketů <http:/www.tcpdump.org/pcap3_man.html>.
[23] Wireshark - síťový analyzátor protokolů s grafickým rozhraním <http://www.wireshark.org/>.

k úpravě dat obsažených v tabulce. Skript ponechá v tabulce vždy pouze ty pakety, jejichž časové razítko splňuje podmínku uvedenou v následujícím textu 6.3:

```
packet_timestamp > timestamp.now - purgeInterval
```

Text 6.3: Vzorec pro výpočet časového razítka ponechaných paketů v tabulce

Hodnota timestamp.now představuje aktuální časové razítko. Proměnná purgeInterval je nastavena v konfiguračním souboru a představuje časový interval v minutách, který určuje, jak staré záznamy budou v databázi ponechány. Nesmíme zapomenout, že soubory s odchyceným provozem jsou také ukládány až po definovaném časovém úseku. Pokud tedy budeme odchycený provoz ukládat například po deseti minutách a hodnota purgeInterval bude nastavena na 20 minut, budou v databázi ponechány záznamy o 10 minut starší, než je první nově analyzovaný paket.

Ponechání starších záznamů v databázi je důležité pro generování metrik nástrojem Metrics Extractor. Pokud bychom v databázi ponechali vždy pouze aktuálně odchycený provoz, mohlo by dojít k situaci, kdy nebude správně identifikovaný začátek škodlivého toku, který se vyskytoval v souboru odchyceném o časový interval dříve. K tomuto jevu přesto může dojít při prvním spuštění nástroje, protože nemáme žádný záznam o předchozích paketech.

Tato funkcionalita zároveň zajistí, že nedojde k přeplnění databáze a k případnému vyčerpání volných prostředků na pevném disku.

Pro práci se soubory ve formátu pcap byla zvolena knihovna DPKT verze 1.7, kterou vytvořil Jon Oberheide. Tato knihovna obsahuje rychlé a jednoduché funkce pro tvorbu a parsování síťových paketů. Knihovna nepodporuje velké množství protokolů, pro naše účely si však vystačíme s hlavními podporovanými protokoly: IP, TCP a při případném rozšíření i s protokolem UDP. Nevýhodou této knihovny je její neexistující dokumentace. Na internetu jsou uvedeny většinou pouze příklady jejího použití. Při programování nástroje jsem vycházel z těchto zdrojů [35], [36].

Po vyčištění tabulky se skript pokusí o otevření pcap souboru a v cyklu zpracovává každý uložený paket. Tato část zdrojového kódu je zobrazena v následujícím textu 6.4.

```
# cycle through every packet in pcap file
for ts, buf in pcap:
            eth = dpkt.ethernet.Ethernet(buf)

            # if it is not an IP packet do not process
            if eth.type != dpkt.ethernet.ETH_TYPE_IP:
                  continue

            ip = eth.data

            # if it is not a TCP packet do not process
            if ip.p != dpkt.ip.IP_PROTO_TCP:
                  continue

            # get TCP data
            tcp = ip.data
```

Text 6.4: Část zdrojového kódu zpracovávajícího pcap soubor

Kontrola, zda se jedná o TCP paket je na první pohled zbytečná, jelikož tcpdump, který ukládá odchycený provoz je nastaven k filtrování pouze protokolu TCP. Tato kontrola je do modulu implementována především kvůli možnému budoucímu rozšíření nástroje pro práci s protokolem UDP. Po rozparsování zpracovávaného paketu je vytvořen SQL příkaz, který vloží získané hodnoty do databáze. Vytvořená tabulka obsahuje i hodnoty, které nejsou v této době využívány (například ARP, IGMP a UDP záznamy), ale mohou být využity při případném budoucím rozšíření.

Po úspěšném zpracování všech paketů je soubor smazán z důvodů uvedených v kapitole 5.5.4 Packet Extractor.

Modul Packet Extractor poté zavolá nástroj Metrics Extractor, který z uložených dat vytvoří CSV soubor obsahující výstupní metriky. Po úspěšném vygenerování metrik je v případě verze nástroje pro použití v síťových sondách zavolán modul Metrics To Database pro převedení metrik do databáze nebo v případě verze nástroje pro použití na koncových stanicích přímo modul Metrics Comparator, který vytvořené metriky porovná s metrikami zachycených útoků.

6.4 Verze pro platformu Windows

Před samotným použitím nástroje či knihovny k vývoji aplikace bylo zjištěno, zda je zaručena jejich funkčnost na systémech Windows. Předpokládal jsem tedy hladký průběh instalace a spuštění nástroje Attack Detector i na této platformě. Jedinou očekávanou změnou byla výměna nástroje tcpdump za nástroj WinDump. V tomto okamžiku však nastal problém.

Přestože má být nástroj WinDump plnohodnotným portem nástroje tcpdump do prostředí Windows, neposkytuje jeho kompletní funkcionalitu. Konkrétně požadovaná možnost nastavení ukládání síťového provozu po definovaném časovém intervalu (přepínač -G u nástroje tcpdump) úplně chybí. Tato vlastnost je pro správnou funkčnost nástroje Attack Detector kritická, její náhrada by znamenala změnu celého návrhu resp. složité programové úpravy.

Raději než razantní úpravu kódu nástroje jsem se rozhodl pokusit se vyhledat náhradu za nástroj WinDump. Po otestování několika paketových analyzátorů jsem nalezl nástroj TShark popsaný dále v kapitole, který je konzolovou verzí oblíbeného a často používaného Wiresharku a poskytuje funkcionalitu ukládání provozu v zadaném časovém intervalu. Bohužel však neposkytuje možnost automatického spuštění příkazu po uložení souboru (přepínač -z u nástroje tcpdump). Jelikož jsem nenašel žádný jiný nástroj pro platformu Windows, který by poskytoval všechnu potřebnou funkcionalitu nástroje tcpdump, rozhodl jsem se použít nástroj TShark a přistoupit k úpravě kódu.

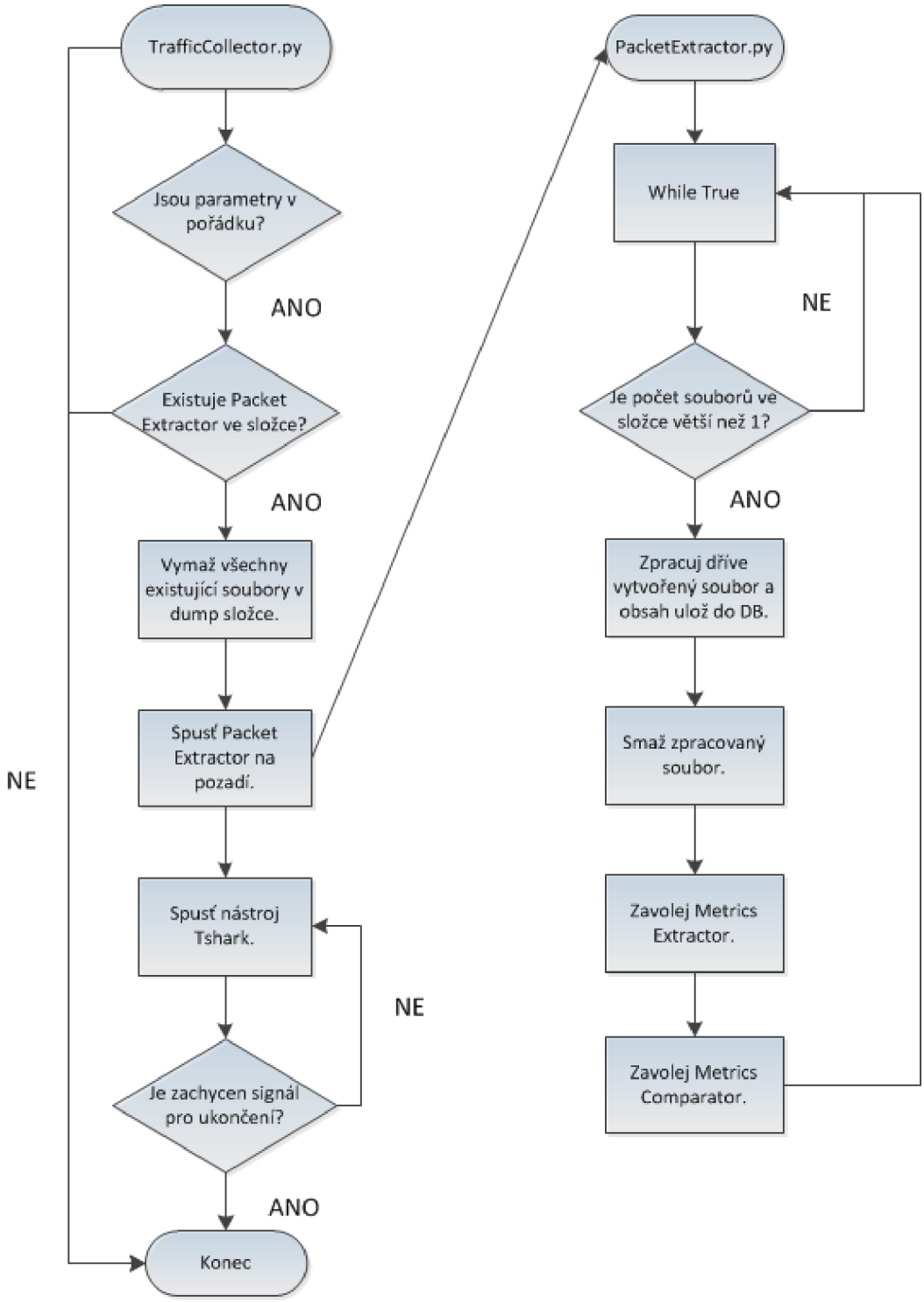
Úprava nástroje pro platformu Windows spočívala v modifikaci modulů Traffic Collector a Packet Extractor. Změny v prvním jmenovaném modulu nebyly příliš razantní. Nástroj tcpdump byl vyměněn za nástroj TShark, byla přidána funkce pro vymazání obsahu složky, kam se mají ukládat soubory se zachyceným provozem a těsně před spuštěním TSharku byl na pozadí zavolán modul Packet Extractor.

```
"tshark.exe -i %s -b duration:%s -w %sout -l -f tcp" % (iface, mins,
dumpdir)
```

Text 6.5: Zdrojový kód spuštění nástroje TShark v modulu Traffic Extractor

Celý návrh modulu Packet Extractor musel být přepracován. Modul nyní v parametru přebírá místo názvu konkrétního souboru název složky, kam jsou ukládány soubory s provozem. Hlavní změnou je běh programu v nekonečné smyčce, ve které je kontrolován obsah zadané složky. Pokud je počet souborů ve složce menší než dva, program nevykonává žádnou akci. Jedná se totiž o případ, kdy ve složce není žádný soubor (ihned po spuštění) nebo se v ní nachází první soubor s provozem, do kterého jsou aktuálně ukládána data, a není s ním tedy možné pracovat. Po uplynutí časového intervalu uzavře nástroj TShark soubor s odchyceným provozem a vytvoří další. Tuto změnu modul Packet Extractor ihned detekuje a pokusí se o zpracování souboru, který je vybírán na základě jeho jména, jelikož nástroj TShark ukládané soubory čísluje vždy ve stoupajícím pořadí. Po úspěšné extrakci dat do databáze je soubor smazán a celý proces se opakuje. Schéma změn je uvedeno na obrázku 6.1: Vývojový cyklus nástroje Attack Detector pro platformu Windows.

Pro korektní spuštění modulu Traffic Collector na platformě Windows musí být složka, ve které se nachází nástroj TShark uvedena v systémové proměnné PATH.



Obrázek 6.1: Vývojový cyklus nástroje Attack Detector pro platformu Windows

6.4.1 TShark

TShark je nástroj obsažený v instalačním balíku open-source paketového analyzátoru Wireshark, který je volně dostupný pro všechny používané platformy jako je Unix a Windows. TShark má

schopnost zachytávat hrubý síťový provoz, číst pakety z dříve zachycených souborů, dekódovat data, využívat filtrů atd. Pro ukládání dat používá formátu libpcap (winpcap) stejně jako tcpdump (WinDump) [37]. K jeho hlavním výhodám ve srovnání s nástrojem WinDump patří daleko širší možnosti nastavení při ukládání síťového provozu. Přesto však nepodporuje veškerou funkcionalitu nástroje tcpdump, konkrétně potřebné spuštění příkazu po uložení síťového provozu do souboru.

6.5 Modul Metrics To Database

Úkolem modulu Metrics To Database je načtení souboru v CSV formátu, který byl vytvořen modulem Metrics Extractor a převedení jeho obsahu do databáze. Modul přijímá dva povinné parametry. Prvním je název souboru obsahujícího metriky a druhým je název tabulky v databázi, kam má být obsah převeden. Po kontrole parametrů se modul pokusí otevřít soubor, který následně načte po řádcích do paměti. Každý řádek je poté rozdělen na jednotlivé hodnoty podle oddělovače (středník). Následně se skript připojí k databázi, vymaže ze zadané tabulky staré záznamy a pokusí se vložit získaná data. Při úspěšném uložení všech dat vypíše modul počet nově vložených záznamů.

6.6 Modul Metrics Comparator

V první části této kapitoly je popsána implementace návrhu nástroje spolupracujícího s databází, jak bylo uvedeno v kapitole 5.3 Návrh systému pro využití v síťových sondách. Dále je vysvětlen algoritmus sloužící k porovnávání jednotlivých metrik.

Druhá část kapitoly obsahuje popis implementace návrhu uvedeného v kapitole 5.4 Návrh nástroje pro využití na koncových stanicích.

6.6.1 Verze pro síťové sondy

Nasazení nástroje využívajícího pro porovnání metrik databázi je předpokládáno na síťových sondách, které pracují s velkým objemem protékajícího síťového provozu. Tyto sondy se mohou skládat z několika propojených databázových serverů schopných tento velký objem dat paralelně zpracovávat.

Implementace samotného modulu Metrics Comparator je poměrně jednoduchá. Skript se po spuštění pokusí připojit k databázi a v případě úspěšného navázání spojení zavolá uloženou proceduru CompareMetrics, které předá tři parametry. Prvním parametrem je název tabulky, ve které se vyskytují metriky odchyceného síťového provozu, druhým parametrem je název tabulky obsahující metriky zachycených útoků a jako třetí parametr je předána hodnota detekčního prahu, která určuje úroveň shody, při které má být síťový provoz označen jako škodlivý. Všechny tyto parametry získá skript z konfiguračního souboru. V případě, že byly nalezeny nějaké útoky, jsou zobrazeny informace ve formátu uvedeném v textu 6.6. V opačném případě nejsou zobrazeny žádné informace.

```
zdrojová_IP_adresa:zdrojový_port -> cílová_ip_adresa:cílový_port
```

Text 6.6: Formát zobrazených informací o škodlivém útoku

Algoritmus porovnání jednotlivých metrik je implementován v databázi v uložené proceduře CompareMetrics. Pro její vložení do databáze jsem vytvořil skript, který může být použit například při instalaci a inicializaci prostředí pro bezproblémový běh celého nástroje. Uložená procedura je implementována v jazyku Python. Pro její spuštění v databázovém systému PostgreSQL je nutné

vytvořit podporu pro jazyk plpython. Toho docílíme spuštěním SQL příkazu uvedeného v textu 6.7 v odpovídající databázi. Tento příkaz je implementován i ve vytvořeném skriptu.

```
CREATE LANGUAGE plpythonu;
```

Text 6.7: Vytvoření podpory jazyka plpython v PostrgeSQL databázi

Algoritmus porovnávací funkce je následující. V prvním kroku jsou z databázové tabulky obsahující metriky zachycených útoků vybrány záznamy, které ve sloupci label obsahují hodnotu True, která identifikuje, že se jedná o útok. Následně jsou vybrány všechny záznamy z tabulky obsahující metriky zachyceného provozu. Každý získaný řádek je po hodnotě porovnán s jednotlivými metrikami zachycených útoků. Porovnání je provedeno na základě zjištění procentuálního rozdílu jednotlivých hodnot.

Pokud se jedná o bool hodnoty je navrácena buď stoprocentní, nebo nulová shoda. V případě porovnání ostatních hodnot je zavolána funkce CompareDouble, která má následující zdrojový kód:

```
def CompareDouble(val1, val2):

      # same values - return 100 percent
      if val1 == val2:
            return 1.0

      # count difference between values
      diff = fabs(val1 - val2)

      # return percentage of difference
      if val1 != 0:
            ret = 1 - fabs(diff/val1)

            # more than 100 percent difference - return 0
            if ret < 0:
                  return 0.0

            # return difference
            return ret
      else:
            ret = 1 - fabs(val2)

            # more than 100 percent difference - return 0
            if ret < 0:
                  return 0.0

            # return difference
            return ret
```

Text 6.7: Vytvoření podpory jazyka plpython v PostrgeSQL databázi

Pokud jsou porovnávané hodnoty stejné, je navrácena stoprocentní shoda. Pokud se hodnoty liší, je vypočítán jejich číselný rozdíl, který je následně převeden na procento shody vstupních hodnot. Pokud je rozdíl hodnot více než stoprocentní, je navrácena nulová shoda.

Hodnoty procentuálního rozdílu jednotlivých hodnot jsou ukládány do vektoru, který je vstupem do funkce ProcessMetricsSimilarity. V této funkci je každá hodnota vektoru vynásobena svojí váhou. Váhy jsou určovány na základě důležitosti jednotlivých metrik. Například zdrojový port má velmi nízkou váhu, jelikož tyto porty jsou náhodně generované a pro zjištění chování útoku má minimální důležitost. Naopak cílový port má vysokou váhu, jelikož ve většině případů určuje službu, na kterou je útok veden (i když v některých případech jsou služby instalovány na nestandardní porty a důležitost cílového portu je při rozhodování o chování útoku v tomto prostředí nižší). Funkce

navrací součet získaných hodnot, který odpovídá procentu shody metriky síťového toku s metrikou zachyceného útoku.

V posledním kroku algoritmu je navrácená hodnota z funkce ProcessMetricsSimilarity porovnána s definovaným detekčním prahem. Pokud je hodnota vyšší, je síťový tok označen jako škodlivý a je generován odpovídající výstup upozorňující na tuto skutečnost. Příklad výstupu je zobrazen v následující kapitole 6.7 Logování událostí.

6.6.2 Verze pro uživatelské stanice

Hlavní výhodou verze uvedené v předchozí kapitole je možnost paralelního zpracování výpočtu pomocí několika databázových serverů. Na koncové uživatelské stanici však taková úroveň paralelizace není možná, proto byl návrh architektury pro koncové stanice pozměněn.

Zásadní implementační změnou je přesunutí algoritmu provádějícího porovnání metrik z databázového systému do modulu Metrics Comparator. Původní algoritmus byl ponechán, byly však provedeny některé úpravy kódu plynoucí ze změny návrhu.

Vstupem upraveného modulu Metrics Comparator jsou nyní dva soubory obsahující příslušné metriky. Lokálně vytvořený soubor Traffic dataset a soubor Attack dataset získaný ze serverové části. Názvy těchto souborů modul načte z konfiguračního souboru. Další změnou je dodaná funkcionalita načtení a rozparsování vstupních souborů.

Formát výstupu zůstává vůči předchozí verzi nezměněn.

6.7 Logování událostí

Pro účely logování všech důležitých událostí jsem vytvořil funkci, která ukládá vzniklé záznamy do lokální databáze. Tento způsob byl zvolen především z důvodů snadnější a flexibilnější práce s logy, v porovnání s manipulací s log záznamy uloženými v souboru. Databázová tabulka, do které jsou zaznamenávány logované události, má strukturu uvedenou v následující tabulce 6.3.

Název sloupce	Typ hodnoty	Popis
id	SERIAL	Identifikační číslo logu.
module	VARCHAR(25)	Název modulu, který log vygeneroval.
timestamp	TIMESTAMP	Čas zaznamenání logované události.
log	TEXT	Obsah logu.

Tabulka 6.3: Struktura tabulky pro logovací záznamy

Každý uložený log záznam obsahuje název modulu odesílajícího vzniklou událost, časové razítko, kdy událost vznikla a samotný obsah vygenerované zprávy. Ve výchozím nastavení se log zprávy zobrazují i do konzole. Tuto funkcionalitu má možnost uživatel vypnout nastavením příslušné hodnoty v konfiguračním souboru.

Nevýhodou ukládání log záznamů do databáze je ztráta informací v případě, že je databáze nedostupná nebo nastane chyba při zápisu. Pro tyto případy je naimplementována funkcionalita, která při chybě zápisu do databáze zajistí uložení všech záznamů do souboru na lokálním disku.

```
TrafficCollector  2012-05-12 04:25:35.661856    Starting tcpdump.
tcpdump: listening on eth1, link-type EN10MB (Ethernet), capture size
65535 bytes
```

```
PacketExtractor   2012-05-12 04:25:56.830452    Deleted data older than
2012-05-12 04:15:56.820306
PacketExtractor   2012-05-12 04:25:58.134403    DB sucessufully filled.
File dumps/ubuntu_2012_05_12_04_25_35.dump. Added 678 rows.
PacketExtractor   2012-05-12 04:25:58.161034    Calling Metrics Extractor.
PacketExtractor   2012-05-12 04:26:07.943234    Traffic metrics sucessfully
created.
PacketExtractor   2012-05-12 04:26:07.986743    Calling Metrics Comparator.
MetricsComparator 2012-05-12 04:26:08.474564    Attack from
192.168.1.100:50892 to 192.168.1.200:21.
MetricsComparator 2012-05-12 04:26:08.528686    Attack from
192.168.1.100:35339 to 192.168.1.200:21.
MetricsComparator 2012-05-12 04:26:08.611676    Attack from
192.168.1.100:39030 to 192.168.1.200:21.
MetricsComparator 2012-05-12 04:26:08.802211    Attack from
192.168.1.100:36580 to 192.168.1.200:80.
MetricsComparator 2012-05-12 04:26:08.836034    Attack from
192.168.1.100:41084 to 192.168.1.200:80.
MetricsComparator 2012-05-12 04:26:09.076730    Attack from
192.168.1.100:60951 to 192.168.1.200:80.
MetricsComparator 2012-05-12 04:26:09.129675    Attack from
192.168.1.100:57875 to 192.168.1.200:8080.
MetricsComparator 2012-05-12 04:26:09.183866    Attack from
192.168.1.100:36580 to 192.168.1.200:80.
MetricsComparator 2012-05-12 04:26:09.394026    Attack from
192.168.1.100:57838 to 192.168.1.200:80.
MetricsComparator 2012-05-12 04:26:09.418707    Attack from
192.168.1.100:41587 to 192.168.1.200:25.
PacketExtractor   2012-05-12 04:26:09.563315    Metrics Comparator
finished.
^C17718 packets captured
17718 packets received by filter
0 packets dropped by kernel
TraficCollector   2012-05-12 04:26:15.658018    Tcpdump exits by
KeyboardInterrupt.
```

Text 6.6: Zobrazení výstupu logovacích zpráv do konzole

Z důvodů velmi detailního a rozsáhlého logování informací modulu Metrics Extractor není jeho výstup zaznamenáván do logovacích informací.

6.8 Ostatní skripty

Během vývoje nástroje Attack Detector bylo vytvořeno několik pomocných skriptů, které usnadňují práci především při nastavování prostředí pro korektní běh aplikace. Konkrétně se jedná o skripty CreateLogTable.py, CreateMetricsTable.py a CreateTcpdumpTable.py, které přijímají jediný parametr a to název databázové tabulky, kterou mají vytvořit. Tyto skripty mohou být v budoucnu využity například při vložení do instalačního souboru, který připraví databázovou strukturu.

6.9 Konfigurační soubor

Nastavení celého nástroje probíhá v konfiguračním souboru Config.py jehož obsah je zobrazen v následujícím textu 6.5: Obsah konfiguračního souboru.

```
# version
pythonVersion = True
```

```
# detection
detTreshold = 0.806

# database connection
dbHost = 'localhost'
dbUser = 'postgres'
dbPass = 'postgres'
dbName = 'aips'
dbPort = '5432'

# metrics
attackMetricsTable = 'attack_metrics'
trafficMetricsTable = 'traffic_metrics'
attackMetricsFile = 'attack_output.csv'
trafficMetricsFile = 'output.csv'

# tcpdump
tcpdumpTable = 'tcpdump'
purgeInterval = '10'

# logging
logTable = 'log'
logFile = 'AttackDetector.log'
logOnConsole = True
```

Text 6.5: Obsah konfiguračního souboru

Podle první konfigurační hodnoty nástroj rozhoduje, jakou verzi spustit. Pokud je proměnná pythonVersion nastavena na True, je spuštěna verze pro koncové uživatelské stanice.

Sekce *detection* obsahuje jedinou konfigurační proměnnou detTreshold, která určuje míru detekčního prahu v procentech. Ideální hodnota tohoto prahu bylo předmětem důkladného testování.

Kategorie *database* slouží k nastavení parametrů pro spojení do lokální databáze. Pro korektní připojení s databází musí být zadány všechny uvedené parametry.

Sekce *metrics* obsahuje názvy tabulek, ve kterých má nástroj hledat uložené metriky odchyceného provozu a známých útoků. Pro verzi nevyužívající databázi jsou zde uloženy názvy odpovídajících vstupních souborů s metrikami.

V oddělení *tcpdump* se vyskytuje hodnota určující název tcpdump tabulky v databázi a proměnná určující délku časového intervalu, po který jsou v databázi udržována odchycená data.

Kategorie *logging* obsahuje proměnné pro nastavení logování událostí. Konkrétně je tu název logovací tabulky v databázi, název logovacího souboru, do kterého se zapisují logy při chybě s databází a možnost vypnutí výpisu logovacích informací do konzole.

7 Testování vytvořeného nástroje Attack Detector

Po implementační části uvedené v předchozí kapitole následovala etapa testování vytvořených modulů. Za tímto účelem byly navrženy dvě testovací prostředí odpovídající reálným případům nasazení nástroje. Tato prostředí byla použita nejen pro účel testování, ale také pro ladění detekčních schopností nástroje Attack Detector. V této kapitole jsou nejprve zobrazena schémata vytvořených prostředí spolu s popisy jejich komponent. Dále je popsán testovací scénář, který byl použit při procesu testování nástroje.

7.1 Testovací prostředí

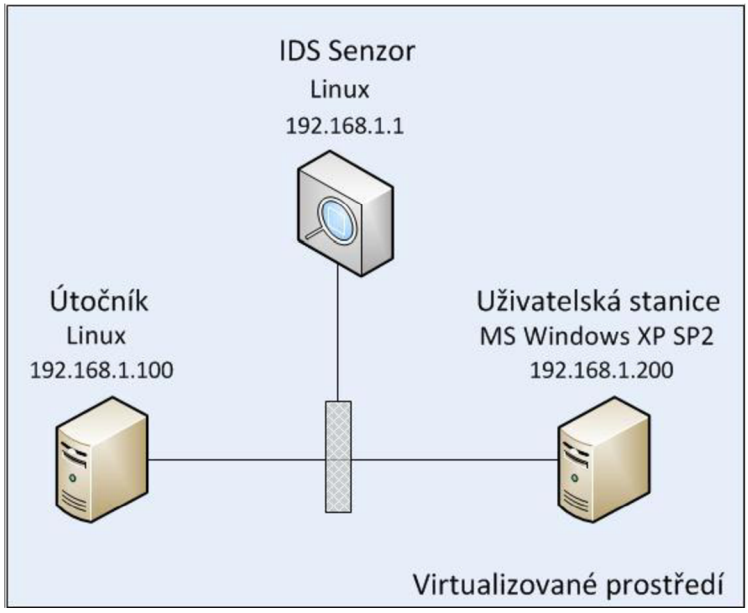
Abych mohl vytvořený nástroj důkladně otestovat, vytvořil jsem si jednoduché virtualizované prostředí, založené na platformě VMware Workstation. Virtualizované prostředí přináší výhody v podobě testování na jedné fyzické stanici. Další velkou výhodou je vytváření tzv. snapshotů[24], díky kterým je velmi ulehčena případná obnova systému.

Při návrhu testovacího prostředí jsem se snažil architekturu co nejvíce zjednodušit a minimalizovat tak počet potřebných komponent. Zároveň jsem se snažil vytvořit prostředí tak, aby odpovídala případům nasazení. Jelikož se jedná o IDS nástroj, možnosti nasazení jsou zpravidla dvě:

- NIDS – nástroj je nasazen pro skenování provozu procházejícího síťovou infrastrukturou
- HIDS – nástroj se nachází na klientské stanici

7.1.1 Prostředí NIDS

Nejdříve jsem se rozhodl otestovat architekturu NIDS, při které jsem využil tří systémů. Schéma tohoto testovacího prostředí je zobrazeno na následujícím obrázku 7.1.



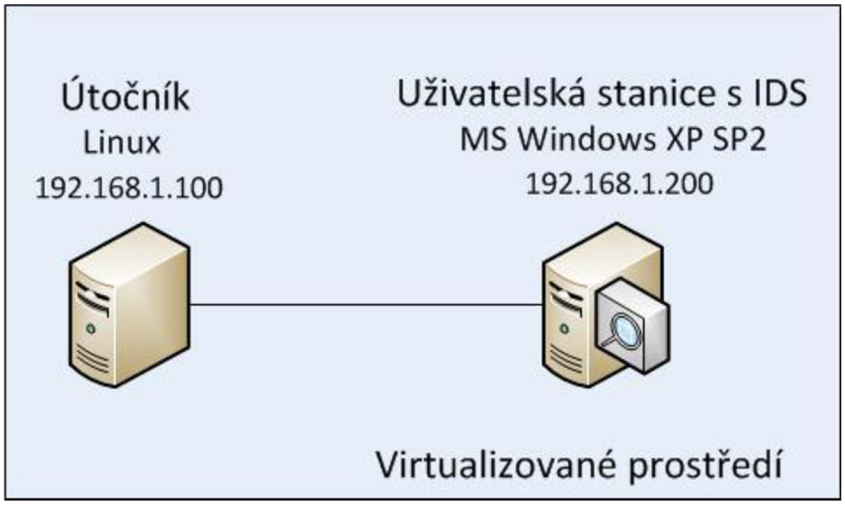
Obrázek 7.1: Schéma navrženého testovacího prostředí (NIDS)

[24] Snapshot - uložený stav systému v určitém okamžiku (paměť, obsah disku, …).

Tento přístup byl pro otestování jednodušší. Jeho hlavní výhodou totiž bylo využití již existujícího systému (na schématu označený jako IDS), na kterém byl nástroj od začátku vyvíjen. Stačilo tedy systémy pouze připojit do jedné virtualizované sítě a nebylo potřeba žádné další konfigurace. Toto testovací prostředí sloužilo především k provedení testů, které souvisely s optimalizací nástroje v rámci určování míry detekčních schopností.

7.1.2 Prostředí HIDS

V tomto prostředí byl vytvořený nástroj nasazen jako IDS řešení na koncové stanici (HIDS). Na testovací uživatelské stanici byl nainstalován operační systém MS Windows. To znamenalo nainstalovat a nastavit na stanici všechny potřebné nástroje a knihovny pro toto prostředí. Vybavení klientské stanice bude blíže popsána v sekci 7.1.5 Uživatelská stanice. Pro platformu Windows bylo dále nutné provést úpravu některých částí zdrojového kódu nástroje Attack Detector, jak bylo uvedeno v kapitole 6.4 Verze pro platformu Windows. Toto testovací prostředí bylo vytvořeno především za účelem ověření funkčnosti nástroje na platformě Windows. Schéma prostředí je uvedeno na následujícím obrázku 7.2.



Obrázek 7.2: Schéma navrženého testovacího prostředí (HIDS)

7.1.3 Stanice útočník

Na stanici označené ve schématu jako útočník byla nainstalována poslední verze linuxové distribuce BackTrack[25], konkrétně BackTrack 5 R2. Tato distribuce je vytvořena pro účely penetračního testování a obsahuje velké množství připravených nástrojů. Mnoho z nich je určeno pro testování kvality IDS systémů.

Hlavním nástrojem, který jsem použil pro provedení útoků na uživatelskou stanici je Metasploit Framework v4.3.0[26]. Tento nástroj představuje platformu pro testování útoků na zranitelné aplikace a operační systémy. Jeho velkou výhodou je jednoduché ovládání a široká databáze exploitů, která aktuálně obsahuje více než 900 vzorků. Nástroj byl zvolen také proto, že byl použit pro provedení útoků na honeypot systémy. Odchycená data byla následně převedena do serverové databáze, ze které byly vygenerovány testovací metriky zachycených útoků.

[25] Backtrack - linuxová distribuce obsahující velké množství nástrojů pro penetrační testování <http://www.backtrack-linux.org/>.
[26] Metasploit Framework - nástroj sloužící pro penetrační testování systémů a aplikací <http://metasploit.com/>.

7.1.4 IDS senzor

Stanice, která sloužila pro vývoj nástroje, byla použita zároveň jako IDS senzor v testovacím prostředí. Popis softwarového vybavení této stanice je k nalezení v kapitole 6.1 Vývojové prostředí a podpůrné nástroje.

7.1.5 Uživatelská stanice

Pro koncovou stanici byla zvolena 32bitová verze operačního systému Microsoft Windows XP s nainstalovaným Service Packem 2. Tento operační systém byl zvolen především proto, že na něm funguje velké množství zranitelných aplikací, na které je možné použití exploitů.

Pro potřeby otestování vytvořeného nástroje Attack Detector bylo na uživatelskou stanici nainstalováno několik aplikací obsahujících bezpečnostní chybu typu buffer overflow. Na tento zranitelný software byly následně vedeny útoky, které má vytvořený nástroj za úkol detekovat. Pro úspěšné provedení útoku byl na uživatelské stanici vypnut osobní firewall. Výběr aplikací byl proveden na základě existence veřejného exploitu obsaženého v nástroji Metasploit Framework.

Instalační soubory zranitelných aplikací se shánějí velmi těžko, jelikož výrobci usilují o to, aby zranitelný software byl co nejdříve stažen z oběhu. Po několikahodinovém usilovném pátrání na internetu byly aplikace získány z [38]. Následuje seznam názvů zranitelného software nainstalovaného na uživatelské stanici spolu s jeho verzí a službou, která byla použita pro exploitaci. Všechny služby používají spojovaný protokol TCP.

Název	Verze	Protokol
Alt-N SecurityGateway	1.0.1	HTTP
BadBlue Enterprise Edition	2.72	HTTP
CesarFTP	0.99g	FTP
Easy Chat Server	2.2	HTTP
EasyFtp Server	1.7.0.11	HTTP
FileCOPA FTP Server	1.01	FTP
Firebird Server	1.5.3.4870	GDS DB
Firebird Server	2.0.0.12748	GDS DB
freeSSHd	1.09	SSH
Mercury for Win32	4.5.1	SMTP
PSOProxy	0.91	HTTP
Sambar Server	6.01	HTTP
Savant Web Server	3.1	HTTP
WAR-FTPD	1.65	FTP
Windows RSH daemon	1.8	RSH
Xitami Web Server	2.5c2	HTTP

YahooPOPs!	0.6	SMTP

Tabulka 7.1: Nainstalovaný zranitelný software na uživatelské stanici

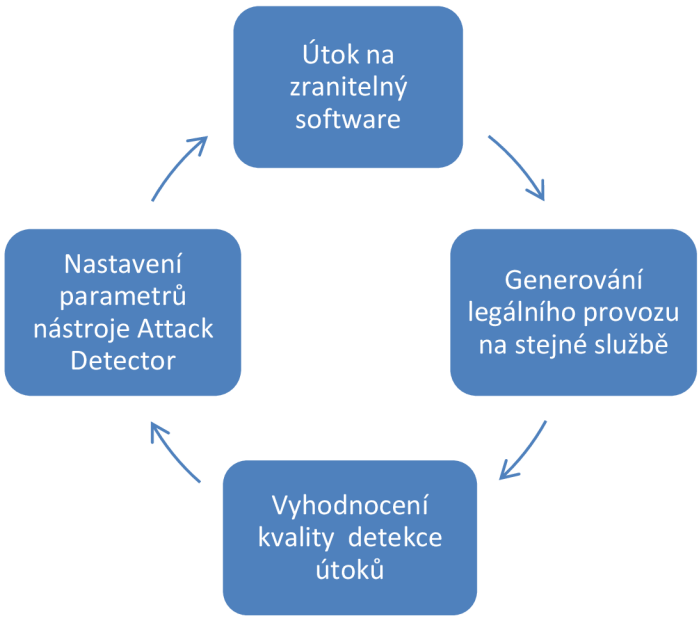
Pro správnou funkčnost nástroje Attack Detector na platformě Windows musel být na stanici nainstalován software zobrazený v následující tabulce 7.2.

Název	Verze
PostgreSQL	9.1.3
Python Knihovna DPKT Knihovna Psycopg2 Knihovna Numpy	2.7.3 1.7 2.4.5 1.6.1
TShark	1.6.2
Winpcap	4.1.2

Tabulka 7.2: Nainstalovaný software potřebný pro běh nástroje na platformě Windows

7.2 Testovací scénář

Za účelem zjištění a případného vylepšení detekčních schopností nástroje Attack Detector jsem připravil testovací scénář, jehož postup je graficky znázorněn na obrázku 7.3. Tento scénář byl vytvořen především pro zjištění úrovně správně detekovaných útoků a s tím spojeným počtem vygenerovaných falešných poplachů.



Obrázek 7.3: Proces testování detekčních schopností nástroje Attack Detector

Hlavním cílem tohoto testovacího procesu bylo vyladění detekčních schopností nástroje tak, aby detekoval co nejvyšší počet známých útoků a generoval minimální procento falešných poplachů.

7.2.1 Útok na zranitelný software

Prvním krokem při procesu testování bylo provedení útoku na konkrétní zranitelnou službu nainstalovanou na uživatelské stanici. Pro tento účel byl využit již zmiňovaný Metasploit Framework. Jako payload byl ve většině případů zvolen windows/meterpreter/reverse_tcp, který v případě úspěšné exploitace naváže spojení zpět na útočníkův server. V následující tabulce jsou uvedeny všechny použité exploity spolu s označením úspěšnosti útoku buffer overflow a případnými poznámkami.

Název exploitu	Úspěšnost	Poznámky
exploit/windows/ftp/cesarftp_mkd	DoS	Windows XP zahlásí chybu. Aplikace běží, nicméně FTP služba není dostupná.
exploit/windows/ftp/easyftp_cwd_fixret	Remote	
exploit/windows/ftp/easyftp_list_fixret	DoS	Služba se zastaví, nespustí se payload. Verze exploitu Windows XP SP3.
exploit/windows/ftp/easyftp_mkd_fixret	Neúspěšný	
exploit/windows/ftp/filecopa_list_overflow	DoS	Aplikace spadne, nespustí se payload. Verze exploitu Windows XP SP2 Italian.
exploit/windows/ftp/warftpd_165_pass	DoS	Aplikace spadne, nespustí se payload. Verze exploitu Windows 2000.
exploit/windows/ftp/warftpd_165_user	Remote	
exploit/windows/http/altn_securitygateway	Remote	
exploit/windows/http/badblue_passthru	Remote	
exploit/windows/http/easyftp_list	DoS	Služba se zastaví. Connection reset by peer. Verze exploitu Windows XP SP3.
exploit/windows/http/efs_easychatserver_username	Remote	
exploit/windows/http/psoproxy91_overflow	Remote	
exploit/windows/http/sambar6_search_results	DoS	Aplikace spadne, nespustí se payload.
exploit/windows/http/savant_31_overflow	Remote	
exploit/windows/http/xitami_if_mod_since	DoS	Aplikace spadne, nespustí se payload.
exploit/windows/misc/fb_isc_attach_database	Remote	Verze exploitu pro Firebird 2.0.0.12748.
exploit/windows/misc/fb_isc_create_database	Remote	Verze exploitu pro Firebird 2.0.0.12748.
exploit/windows/misc/fb_svc_attach	Remote	Verze exploitu pro Firebird 1.5.3.4870.
exploit/windows/misc/windows_rsh	DoS	Aplikace spadne, nespustí se payload.
exploit/windows/smtp/mercury_cram_md5	Remote	
exploit/windows/smtp/ypops_overflow1	DoS	Aplikace spadne, nespustí se payload.
exploit/windows/ssh/freesshd_key_exchange	DoS	Aplikace spadne, nespustí se payload. Verze exploitu Windows XP SP1.

Tabulka 7.3: Použité exploity a jejich úspěšnost

Útoky označené jako Remote vedly k úspěšnému spuštění payloadu, došlo tedy ke kompletní kompromitaci cílové stanice. Při útoku označeném jako DoS (Denial Of Service) došlo k úspěšné exploitaci zranitelnosti buffer overflow, která způsobila selhání aplikace. Nedošlo však ke korektnímu spuštění neseného payloadu. Ve většině případů to bylo způsobeno tím, že exploit byl vytvořen pro jinou verzi operačního systému. Z provedených útoků úspěšně neproběhl pouze jeden.

Aby mohl být proces testování dané služby opakován bez nutnosti opětovného provedení útoku, byl zaznamenaný provoz uložen do souboru. Provoz byl uložen a analyzován i v případě, kdy se útok nepodařilo úspěšně realizovat.

7.2.2 Generování legálního provozu

Pro otestování počtu falešných poplachů byl v druhém kroku generován legální provoz ze stanice útočník na danou testovanou službu spuštěnou na uživatelské stanici. Vždy byl kladen důraz na legální použití příkazů, které vedly k zneužití zranitelnosti buffer overflow. V případě služby HTTP se většinou jednalo o připojení na webový server a provedení co největšího počtu povolených funkcionalit, například uploadu velkých souborů či zadání velkého množství dat do vstupních polí. Na službě FTP byly hojně využívány příkazy, pomocí kterých byla zranitelnost zneužita (USER, PASS, MKD, LIST, CWD, …). Generovaný legální provoz byl opět zachycen a uložen do souboru pro potřeby dalšího testování.

7.2.3 Vyhodnocení úrovně detekce

Ve třetím kroku byly ze získaného provozu vytvořeny metriky, které byly porovnány s testovacím Attack datasetem získaným ze serverové části systému AIPS.

Testovací dataset byl vytvořen ostatními členy řešitelského týmu projektu AIPS stejným způsobem ještě před samotným vývojem nástroje Attack Detector. Na honeypotech byly nainstalovány stejné zranitelné aplikace, na které byly vedeny stejné útoky. Attack dataset aktuálně obsahuje 12 detekovaných útoků.

Vyhodnocení úrovně detekčních schopností nástroje bylo založeno především na úspěšném odhalení provedeného útoku. Důležitým údajem byl také počet vygenerovaných falešných poplachů.

7.2.4 Nastavení parametrů nástroje

V posledním kroku testovacího procesu bylo provedeno většinou experimentální nastavení parametrů nástroje, jejichž změna má dopad na úroveň jeho detekčních schopností.

Hlavním parametrem, který určuje úroveň detekce probíhajících útoků je detekční práh, který představuje procentuální hodnotu shody porovnávaných metrik. Jeho hodnota byla vyladěna tak, aby byly zachyceny všechny testované útoky.

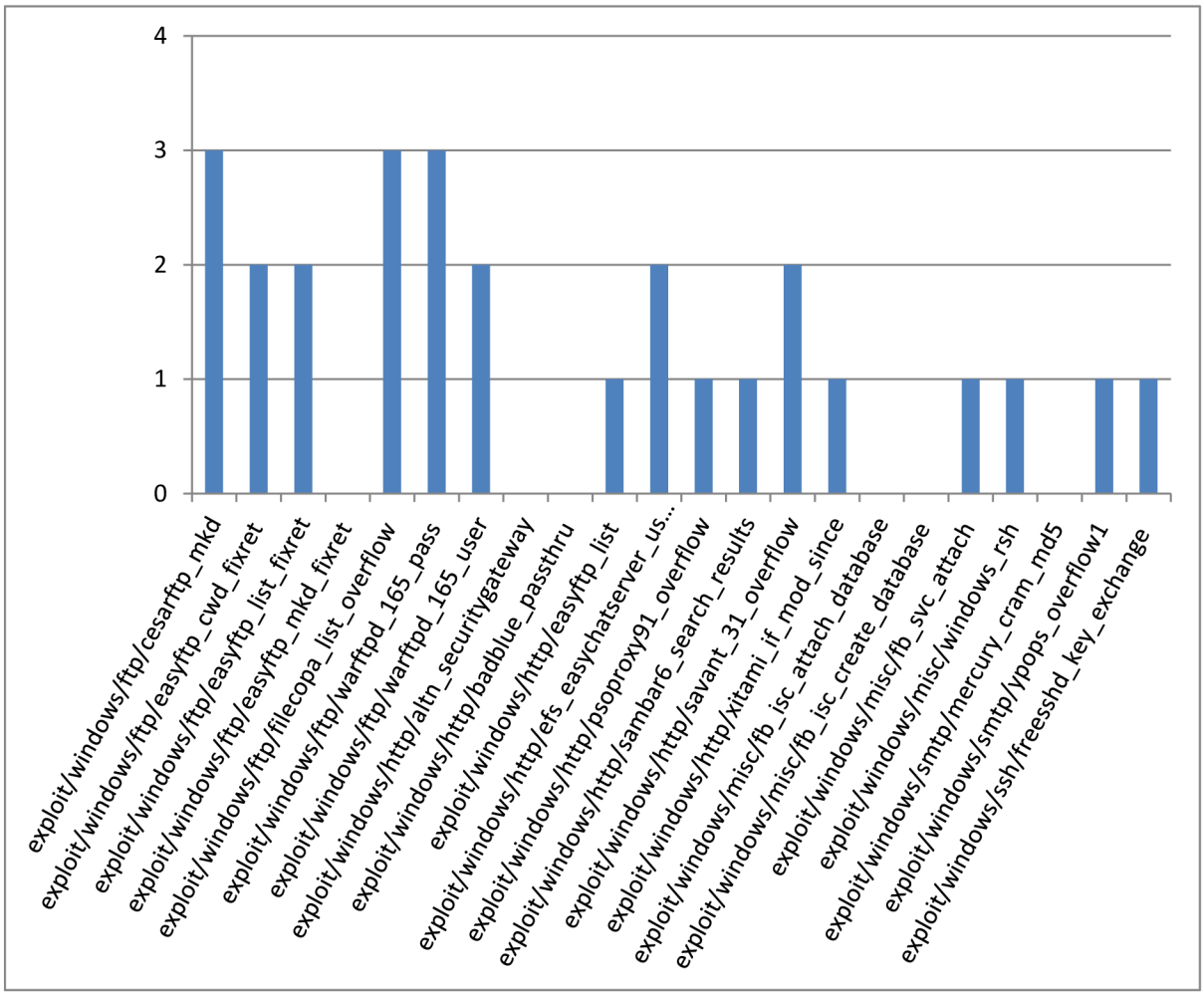
Další parametry ovlivňující rozhodovací schopnosti nástroje, jsou váhy jednotlivých metrik. Tyto váhy určují důležitost dané metriky při procesu vytváření výsledné procentuální shody.

8 Výsledky testů

Po vyladění detekčních schopností nástroje byly provedeny finální testy, jejichž výsledky jsou uvedeny v této kapitole. Hlavním kritériem pro vyhodnocení úspěšnosti detekce nástroje je počet korektně zaznamenaných testovaných útoků. Těmto získaným výsledkům se věnuji v první části kapitoly. Dále je diskutováno další velice důležité kritérium, kterým je počet vygenerovaných falešných poplachů. Výsledky testů ukazují, že nástroj Attack Detector generuje malé procento těchto poplachů. Na závěr celé kapitoly jsou uvedeny výsledky testů rychlosti běhu jednotlivých modulů vytvořeného nástroje.

8.1 Počet detekovaných útoků

Pro otestování úspěšnosti detekce útoků nástrojem Attack Detector bylo využito 22 exploitů, jejichž názvy a úspěšnost jsou uvedeny v tabulce 7.3: Použité exploity a jejich úspěšnost. Počet upozornění, který nástroj vygeneroval pro jednotlivé útoky je zobrazen na následujícím grafu 8.1.



Graf 8.1: Počet vygenerovaných alarmů pro jednotlivé exploity

8.1.1 Nezachycené útoky

Z uvedeného grafu je patrné, že šest útoků nebylo vůbec detekováno. Důvod, proč nástroj Attack Detector nezaznamenal tyto provedené útoky je ten, že metriky identifikující odpovídající škodlivý tok se nevyskytovaly v testovacím Attack datasetu získaného ze serverové části systému AIPS. Přestože byl útok na honeypot proveden, detekční systém ho nedokázal identifikovat a vygenerovaná metrika nebyla v testovacím Attack datasetu označena jako škodlivá.

8.1.2 Vícenásobně detekované útoky

Zbylých 16 útoků bylo úspěšně zachyceno, přestože se v testovacím datasetu vyskytuje pouze 12 označených škodlivých metrik. Nástroj dokonce některé útoky identifikoval vícekrát pomocí různých metrik. Tento fakt je důsledkem toho, že provedené útoky mají velmi podobné až identické chování. Vezmeme-li v úvahu například útoky na službu FTP (prvních sedm vynesených hodnot v grafu) je jejich povaha téměř totožná. Vždy se jedná o připojení na službu a následné zaslání požadavku (paketu) zneužívajícího zranitelnost buffer overflow. Téměř identické chování útoků je patrné z následujících obrázků, které zobrazují komunikaci rozdílných exploitů se službou FTP a na kterých je zároveň vyznačen paket, který způsobil zneužití chyby buffer overflow. Podobné chování útoků vysvětluje i vícenásobnou detekci některých škodlivých kódů cílených na aplikace používající protokol HTTP.

```
TCP      74 50892 > ftp [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=22750886 TSecr=0 WS=16
TCP      78 ftp > 50892 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=1 TSval=0 TSecr=0 SACK_PERM=1
TCP      66 50892 > ftp [ACK] Seq=1 Ack=1 Win=14608 Len=0 TSval=22750886 TSecr=0
FTP     103 Response: 220 CesarFTP 0.99g Server Welcome !
TCP      66 50892 > ftp [ACK] Seq=1 Ack=38 Win=14608 Len=0 TSval=22750887 TSecr=1234
FTP      82 Request: USER anonymous
FTP     107 Response: 331 User login OK, waiting for password
FTP      92 Request: PASS mozilla@example.com
FTP     111 Response: 230 User password OK, CesarFTP server ready
FTP    1040 Request: MKD
TCP      66 50892 > ftp [FIN, ACK] Seq=1017 Ack=124 Win=14608 Len=0 TSval=22750890 TSecr=1234
TCP      66 ftp > 50892 [ACK] Seq=124 Ack=1018 Win=64519 Len=0 TSval=1234 TSecr=22750890
```

Obrázek 8.1: Síťová komunikace vytvořená při exploitaci aplikace CesarFTP 0.99g

```
TCP      74 38818 > ftp [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=23480723 TSecr=0 WS=16
TCP      78 ftp > 38818 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=1 TSval=0 TSecr=0 SACK_PERM=1
TCP      66 38818 > ftp [ACK] Seq=1 Ack=1 Win=14608 Len=0 TSval=23480723 TSecr=0
FTP     170 Response: 220-InterVations FileCOPA FTP Server Version 1.01 6th April 2006
TCP      66 38818 > ftp [ACK] Seq=1 Ack=105 Win=14608 Len=0 TSval=23480724 TSecr=12445
FTP      82 Request: USER anonymous
FTP     142 Response: 331 Anonymous login ok, send your complete email address as your password.
FTP      92 Request: PASS mozilla@example.com
FTP     247 Response: 230-Welcome to the FileCOPA FTP Server.  Anonymous Access has been granted
TCP      66 38818 > ftp [ACK] Seq=43 Ack=362 Win=15680 Len=0 TSval=23480735 TSecr=12445
FTP     919 Request: LIST A \025\032⍰4vu7⍰zG2⍰⍰⍰~I⍰q-\206⍰\034\227⍰\226\233⍰H\223⍰\230⍰\237}\020⍰'\222\
TCP      66 38818 > ftp [FIN, ACK] Seq=896 Ack=362 Win=15680 Len=0 TSval=23480742 TSecr=12445
TCP      62 ftp-data > ftp-data [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
```

Obrázek 8.2: Síťová komunikace vytvořená při exploitaci aplikace FileCOPA FTP 1.01

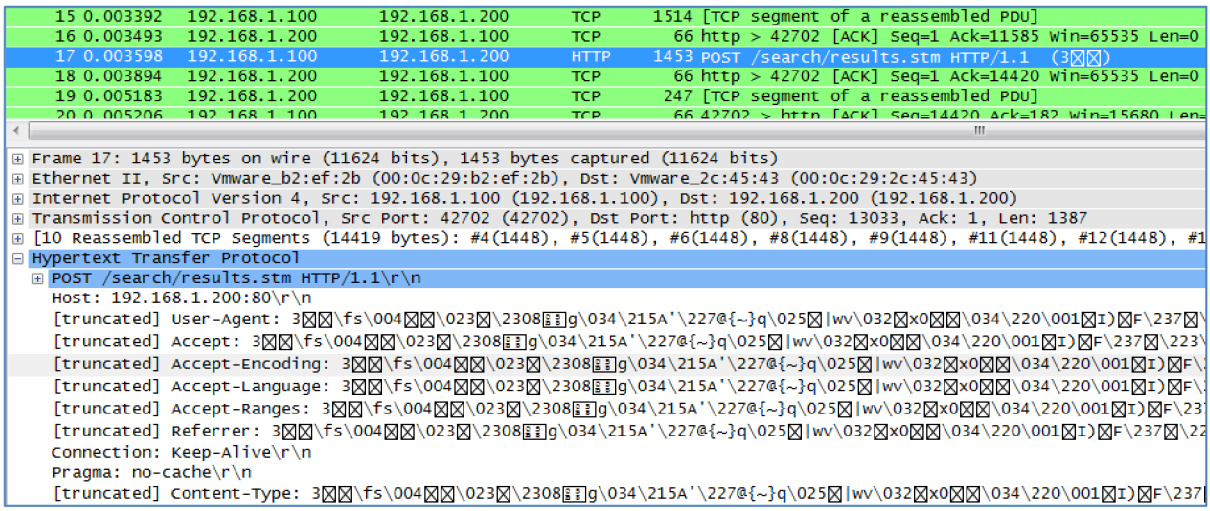
```
TCP      74 47831 > ftp [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=23862070 TSecr=0 WS=16
TCP      78 ftp > 47831 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=1 TSval=0 TSecr=0 SACK_PERM=1
TCP      66 47831 > ftp [ACK] Seq=1 Ack=1 Win=14608 Len=0 TSval=23862070 TSecr=0
FTP     154 Response: 220- Jgaa's Fan Club FTP Service WAR-FTPD 1.65 Ready
TCP      66 47831 > ftp [ACK] Seq=1 Ack=89 Win=14608 Len=0 TSval=23862071 TSecr=27699
FTP      82 Request: USER anonymous
FTP     129 Response: 331 User name okay. Give your full Email address as password.
TCP      66 47831 > ftp [ACK] Seq=17 Ack=152 Win=14608 Len=0 TSval=23862089 TSecr=27700
FTP    1063 Request: PASS ⍰⍰\002⍰%/1⍰⍰G\025⍰s{~\t⍰⍰$OB=⍰\231\024⍰\230⍰\205⍰⍰\fH,\004rI⍰;⍰⍰-9⍰\034\206⍰\
TCP      66 47831 > ftp [FIN, ACK] Seq=1014 Ack=152 Win=14608 Len=0 TSval=23862089 TSecr=27700
TCP      66 ftp > 47831 [ACK] Seq=152 Ack=1015 Win=64522 Len=0 TSval=27700 TSecr=23862089
```

Obrázek 8.3: Síťová komunikace vytvořená při exploitaci aplikace WAR-FTPD 1.65

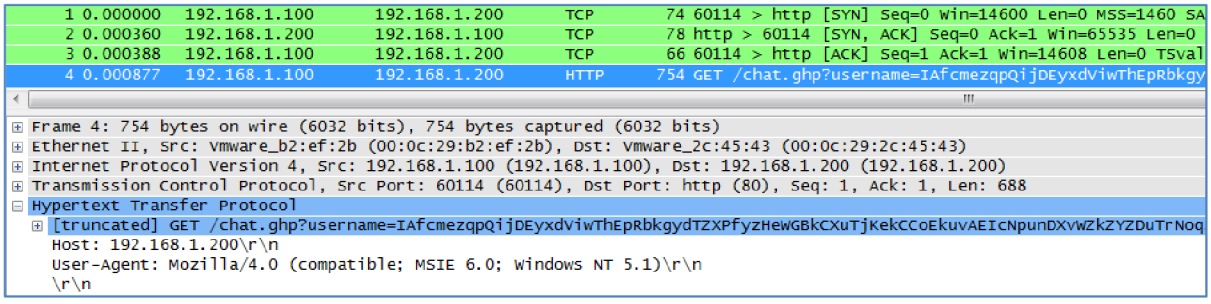
8.1.3 Ostatní zachycené útoky

Z grafu 8.1: Počet vygenerovaných alarmů pro jednotlivé exploity vyplývá, že polovina útoků byla detekována korektně pouze jednou. Jedná se většinou o útoky vedené na služby, které se v testovacím vzorku vyskytují pouze jednou nebo dvakrát. Jejich chování je od ostatních útoků značně odlišné, proto nedochází k vícenásobné detekci.

Výjimku tvoří aplikace využívající protokol HTTP, které jsou v testovacím vzorku zastoupeny šestkrát a k jejímž vícenásobným detekcím došlou pouze ve dvou případech. Důvodem přesné detekce je stavový prostor protokolu HTTP, který umožňuje širší možnosti pro vedení útoku (například v porovnání se službou FTP). To značně odlišuje chování jednotlivých útoků vedených pomocí tohoto protokolu. Při pohledu na komunikaci rozdílných exploitů zneužívajících zranitelnosti aplikací komunikujících pomocí protokolu HTTP je patrné jejich rozdílné chování.



Obrázek 8.4: Útok vedený na aplikaci Sambar Server 6.01



Obrázek 8.5: Útok vedený na aplikaci Easy Chat Server 2.2

8.2 Počet falešných detekcí

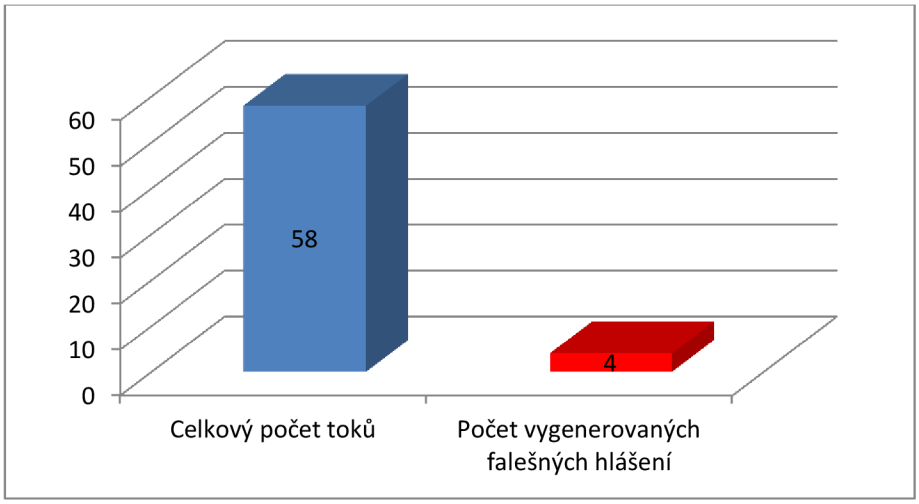
Pro zjištění úrovně vygenerovaných falešných poplachů byly vytvořeny tři testovací množiny. První množina obsahovala legální síťový provoz, který měl podobné chování jako provoz vytvořený jednotlivými útoky. Zbylé dvě množiny obsahovaly legální provoz zaměřený na službu FTP resp. na legálně vytvořený síťový provoz komunikující pomocí protokolu HTTP.

8.2.1 Síťový provoz imitující útoky

Pro vytvoření této testovací množiny byla nejdříve zjištěna povaha všech provedených útoků jejich detailní analýzou. Následně byl generován legální provoz, jehož cílem bylo co nejvěrnější napodobení chování provedených útoků. Pokud tedy útok zneužíval například službu FTP a příkaz LIST (vypsání obsahu složky), bylo simulováno chování, které představovalo legální připojení uživatele na službu FTP, zobrazení obsahu dané složky a odpojení od služby. Tato testovací množina obsahovala ve výsledku 58 spojení, které napodobovala chování všech testovaných útoků.

Počet vytvořených legálních spojení	Počet generovaných falešných poplachů	Procento vygenerovaných falešných poplachů
58	4	6.89%

Tabulka 8.1: Počet vygenerovaných falešných hlášení pro legální provoz napodobující útoky



Graf 8.2: Počet vygenerovaných falešných hlášení pro legální provoz napodobující útoky

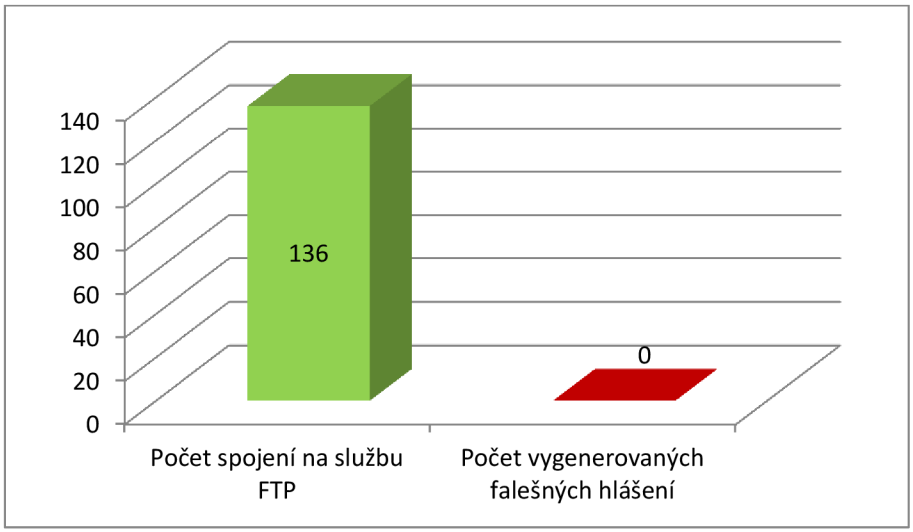
Falešná hlášení byla vygenerována třikrát pro protokol HTTP a jednou pro protokol GDS DB běžící na portu 3050.

8.2.2 Legální provoz generovaný na službě FTP

Jelikož téměř třetina všech testovacích útoků byla vedena vůči službě FTP, rozhodl jsem se vytvořit testovací množinu pouze pro tuto službu. Ze 136 testovaných legálních spojení nevygeneroval nástroj žádný falešný poplach.

Počet vytvořených legálních spojení	Počet generovaných falešných poplachů	Procento vygenerovaných falešných poplachů
136	0	0.00%

Tabulka 8.2: Počet vygenerovaných falešných hlášení pro legální provoz na službě FTP



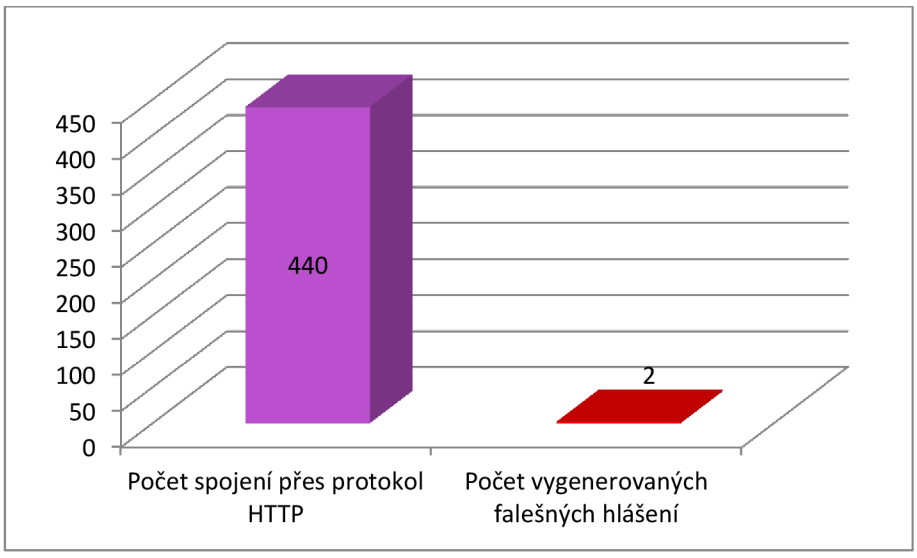
Graf 8.3: Počet vygenerovaných falešných hlášení pro legální provoz na službě FTP

8.2.3 Legální HTTP provoz

Pro otestování falešných poplachů pro služby běžící nad protokolem HTTP, které tvořily více než třetinu všech testovaných služeb, byla vytvořena testovací množina, která obsahovala 440 spojení. Tato spojení byla vytvořena především prohlížením webových stránek. Výsledný počet generovaných falešných hlášení je zobrazen na následujícím grafu 8.4.

Počet vytvořených legálních spojení	Počet generovaných falešných poplachů	Procento vygenerovaných falešných poplachů
440	2	0.45%

Tabulka 8.3: Počet vygenerovaných falešných poplachů v legálním HTTP provozu



Graf 8.4: Počet vygenerovaných falešných poplachů v legálním HTTP provozu

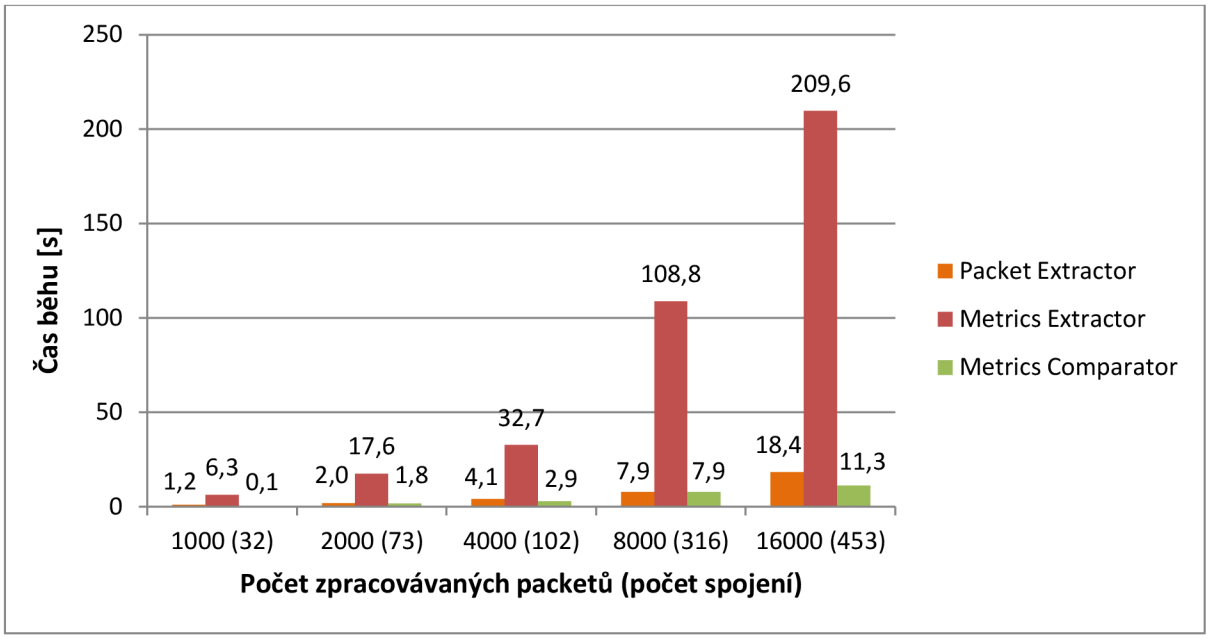
8.3 Rychlost nástroje

Pro zjištění časového úseku, který vytvořený nástroj spotřebuje pro svůj běh, byly provedeny testy rychlosti. Celkový čas běhu nástroje byl vypočítán jako součet času spotřebovaného po běh jednotlivých modulů. Ze součtu byl odstraněn čas spotřebovaný modulem Traffic Collector, jehož běh trvá zanedbatelnou dobu, protože pakety jsou do souboru ukládány průběžně a není tak potřeba režie na výsledné uložení dat do souboru. Pro automatizaci celého procesu testování rychlosti byl vytvořen skript, který jako jediný parametr přebírá název modulu. Tento skript spustí zadaný modul desetkrát za sebou a vypočítá průměrnou hodnotu z jednotlivých časů jeho trvání.

Důležitým parametrem při provádění testů rychlosti je objem zpracovávaných dat. Pro účely testování jsem generoval legální provoz (prohlížení webu), který byl monitorován a po zadaném počtu zachycených paketů byl uložen do souboru. Rozhodl jsem se vytvořit vzorky dat o objemu 1000, 2000, 4000, 8000 a 16000 zachycených paketů.

Pro modul Metrics Comparator není důležitý počet odchycených paketů, ale počet spojení, které se v daném síťovém provozu vyskytují. Hodnota počtu spojení je na grafu 8.5 uvedena v závorce za počtem odchycených paketů. Další důležitou informací, související s rychlostí modulu Metrics Comparator je počet signatur zachycených útoků, vůči kterým probíhalo porovnání. Jelikož byl použit testovací Attack dataset uvedený v kapitole 7.2.3 Vyhodnocení úrovně detekce je tato hodnota rovna číslu 12.

Porovnání rychlosti bylo prováděnou pouze pro verzi určenou na koncové stanice. Testování probíhalo na virtuálním stroji se softwarovým vybavením uvedeném v kapitole 6.1 Vývojové prostředí a podpůrné nástroje. Stroj měl přiřazena dvě výpočetní jádra o frekvenci 1.6Ghz a 1GB operační paměti.



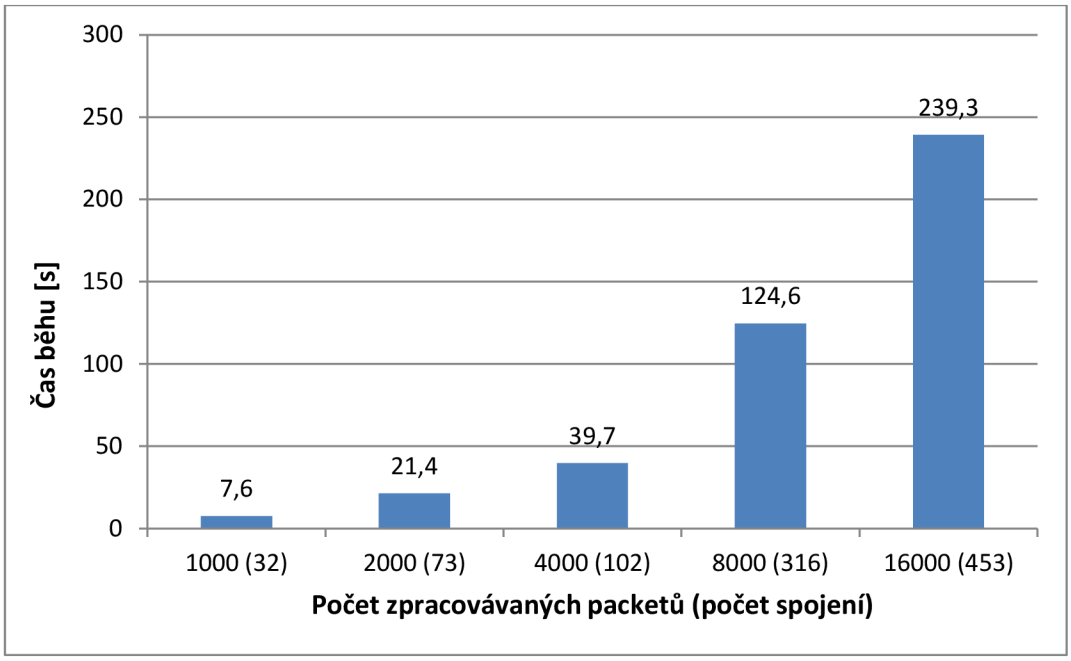
Graf 8.5: Časová náročnost jednotlivých modulů v závislosti na zpracovávaném objemu dat

Z předchozího grafu je patrné, že všechny moduly mají lineární časovou složitost. Výkonově na tom je nejhůře modul Metrics Extractor, jehož průměrná doba běhu při vytváření metrik z 16000 paketů odpovídá téměř hodnotě 210 sekund.

Při pohledu na časovou náročnost modulu Metrics Comparator je nutné vzít v úvahu množství porovnání, které musel modul provést. Počet vykonaných porovnání získáme vynásobením počtu

spojení s hodnotou 12, která odpovídá výskytu škodlivých toků v testovacím datasetu. V reálném použití bude však počet signatur škodlivých toků daleko vyšší. Při porovnání vzorku obsahujícího 16000 paketů (453 spojení) s vytvořeným testovacím datasetem, ve kterém bylo uloženo 200 signatur, trval běh modulu Metrics Comparator v průměru okolo tří minut.

Rychlost běhu celého nástroje v závislosti na objemu zpracovávaných dat je zobrazena na následujícím grafu 8.6.



Graf 8.6: Celkový čas běhu nástroje

9 Závěr

V dnešní době roste množství nově vytvořeného malware vysokou rychlostí a proto je důležité, aby reakce na takové útoky byla co nejpohotovější. Dosavadní bezpečnostní technologie detekující malware pomocí signatur reagují na nově vytvořené útoky se značnou časovou prodlevou, během které může škodlivý software infikovat tisíce počítačových systémů a dokáže tak napáchat velké škody. K řešení této situace by mohla přispět bezpečnostní řešení, která dokáží odchytávat nové hrozby pomocí honeypotů a automaticky generují a distribuují signatury, které chování nově objeveného škodlivého software detailně popisují. Tyto systémy by měly rapidně snížit reakční dobu na obranu proti novému malware, čímž by měly zabránit jeho rychlému rozšíření a napáchané škody by tak měly být minimalizovány.

V této práci byl nejdříve uveden aktuální problém rostoucího výskytu malware v síti internet a nevýhody dosavadního procesu tvorby signatur pro detekční zařízení. Řešením tohoto problému by mohl být systém pro automatické generování signatur, který je diskutován v průběhu celé této práce. V teoretické části práce byly popsány a analyzovány existující metody automatického generování signatur, především pro rychle se šířící internetové červy. Znalosti získané z této analýzy byly použity při návrhu metody pro automatické generování signatur později implementované ostatními členy projektu AIPS.

Projekt AIPS je aktuálně řešen na FIT VUT v Brně a jeho záměrem je vytvoření systému, který automatizuje proces rozpoznání a zpracování útoků. V současné době má systém implementovánu serverovou část založenou na honeypotech, pomocí kterých odchytává nové vzorky malware a z těchto vzorků poté automaticky vytváří signatury. Každá signatura je složena z metrik identifikujících chování zachyceného útoku. Systém nemá doposud implementovanou detekční část, která vytvořené metriky ze serverové části porovná s metrikami probíhajícího síťového provozu a která při nalezení vzorků škodlivého software generuje odpovídající upozornění.

Vytvoření detekční části systému AIPS bylo hlavním cílem této diplomové práce. Za tímto účelem byl navržen a implementován nástroj Attack Detector, který odchytává probíhající síťový provoz, vytváří z něj metriky a ty následně porovnává s testovacími metrikami získanými ze serverové části systému AIPS. V rámci práce byly vytvořeny dvě verze nástroje Attack Detector. První verze využívá pro porovnání metrik databázový systém a její nasazení je plánováno v síťových sondách, které zpracovávají velké množství dat a jejichž náročnost na rychlost nástroje je vysoká. Druhá verze nástroje má porovnávací modul implementován čistě v jazyce Python a její využití je plánováno na koncových stanicích, kde je objem síťového provozu v porovnání s předchozí verzí daleko nižší. Obě verze podporují operační systémy Unix a Windows. Pro zajištění funkčnosti na platformě Windows musel být návrh nástroje přepracován a byly provedeny odpovídající implementační změny.

Vytvořený nástroj Attack Detector dokázal zachytit všechny vzorky testovaných útoků. Některé útoky byly nástrojem detekovány vícekrát. Tato funkcionalita vyplývá z podobného chování některých testovaných útoků, jejichž vytvořené metriky mají velmi podobné charakteristiky. Nástroj korektně detekuje škodlivý síťový tok při nalezení shody s každou signaturou, která identifikuje útok s podobným chováním. Za získaných výsledků je patrné, že nástroj generuje malé procento falešných poplachů. Nejvyšší počet těchto poplachů byl zaznamenán při testování legálního provozu, který se snažil co nejvěrněji napodobit chování útoků. V tomto případě vygeneroval nástroj 6.89% falešných poplachů. Při testování legálního provozu na protokolu HTTP nástroj špatně označil 0.45% provozu a při testech služby FTP nebyly vygenerovány žádné falešné poplachy.

Všechny cíle vytyčené v úvodu této práce byly splněny. Byl navržen a implementován nástroj, který úspěšně detekoval všechny testované útoky, přičemž počet vygenerovaných falešných

hlášení je minimální. Nástroj Attack Detector byl vytvořen z několika samostatných modulů a k jeho implementaci byly použity volně dostupné nástroje.

9.1 Možnosti rozšíření

Jelikož se jedná o první verzi nástroje, možnosti rozšíření jsou poměrně široké. Hlavní nevýhodu spatřuji v omezení detekce útoku probíhajícího pouze nad protokolem TCP. Při testování exploitace zranitelného software bylo zaznamenáno několik útoků, které pro účel kompromitace systému používaly nespojovaného protokolu UDP. Tato funkcionalita však musí být nejdříve podporována na serverové části a až poté může být implementována do vytvořeného nástroje Attack Detector.

Pro správnou funkčnost celého systému musí být vytvořen modul, který bude zpracovávat nově vytvořené metriky zaslané ze serverové části a bude automaticky aktualizovat lokální databázi signatur známých útoků. Potřebná funkcionalita musí být opět implementována nejdříve na serverové části systému.

Dalším důležitým rozšířením může být implementace učícího se systému, který by zajišťoval přizpůsobení nástroje na dané prostředí. Počet generovaných falešných poplachů by tak byl postupem času minimalizován, čímž by byly sníženy náklady na obsluhu systému.

Jako další vylepšení by mohlo být implementováno uvedení názvu útoku při jeho detekci. V současné době se operátor nástroje z generovaného výstupu dozví zdrojovou a cílovou IP adresu spolu se zdrojovým a cílovým portem útoku. Nemá však žádnou znalost o povaze útoku ani o službě, na kterou byl útok veden (může jí odhadnout z cílového portu, ale v reálném prostředí mnohdy aplikace běží na nestandardních portech). V případě nově detekovaných útoků by v signatuře vygenerované na serverové části systému AIPS mohl být uveden název služby (aplikace), na kterou byl útok veden.

Literatura

[1] AV-TEST, The Independent IT-Security Institute: *Malware Statistics*. [online], [cit. 2012-05-08]. Dostupné na URL: <http://www.av-test.org/en/statistics/malware/>

[2] KASPERSKY LAB: *Insights on Stuxnet worm*. [online], [cit. 2012-05-08]. Dostupné na URL: <http://www.kaspersky.com/about/news/virus/2010/Kaspersky_Lab_provides_its_insights_on_Stuxnet_worm>

[3] MOORE, D., PAXSON, V., SAVAGE, S., SHANNON, C., STANIFORD, S., WEAVER N.: *The Spread of the Sapphire/Slammer Worm*. Technical report, CAIDA, ICSI, Silicon Defense, UC Berkeley EECS and UC San Diego CSE, Jan 2003. [online], [cit. 2012-05-08]. Dostupné na URL <http:/www.caida.org/publications/papers/2003/sapphire/sapphire.html>

[4] SERVER UPI.COM: *Virus strikes 15 million PCs.* [online], [cit. 2012-05-08]. Dostupné na URL: <http://www.upi.com/Top_News/2009/01/25/Virus_strikes_15_million_PCs/UPI-19421232924206>

[5] CISCO SYSTEMS, Inc.: *The Science of Intrusion Detection System: Attack Identification*. [online], [cit. 2011-05-08]. Dostupné na URL: <http://www.cisco.com/warp/public/cc/pd/sqsw/sqidsz/prodlit/idssa_wp.htm>

[6] BARABAS, M.: *Malware a IDS*. 2011, Studijní materiály k předmětu Bezpečnost informačních technologií, FIT VUT Brno.

[7] LANDESMAN, M: *What is a Virus Signature?* [online], [cit. 2012-05-08]. Dostupné na URL: <http://antivirus.about.com/od/whatisavirus/a/virussignature.htm>

[8] AXELSSON, S.: *Intrusion detection systems: A survey and taxonomy*. 2000, Technical Report 99-15, Department of Computer Engineering, Chalmers University.

[9] WARAICH, R.: *Automated Attack Signature Generation: A Survey.* 2005, Technical report, ETH Zurich, Computer Engineering and Networks Laboratory.

[10] SCARFONE, K., MELL, P.: *Guide to Intrusion Detection and Prevention Systems (IDPS).* 2007, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg. [online], [cit. 2011-05-08]. Dostupné na URL: <http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf>

[11] *Intrusion Detection Systems.* [online], [cit. 2011-05-08]. Dostupné na URL: <http://fairizul-networksecurityfc.blogspot.com/2009/10/lecture-9-intrusion-detection-systemids.html>

[12] PROVOS, N., HOLZ, T.: *Virtual Honeypots: From Botnet Tracking to Intrusion Detection.* 2007, Addison Wesley, ISBN: 0-321-33632-1.

[13] THAKAR, U., VARMA, S., RAMMANI, A.K.: *HoneyAnalyzer – Analysis and Extraction of Intrusion Detection Patterns & Signatures Using Honeypot*. 2005, The Second International

Conference on Innovations in Information Technology (IIT'05). [online], [cit. 2011-05-08]. Dostupné na URL: <http://www.it-innovations.ae/iit005/proceedings/articles/a_4_iit05_thakar.pdf>

[14] KREIBICH, C., CROWCROFT, J.: *Honeycomb - creating intrusion detection signatures using honeypots*. 2003, In Proceedings of the Second Workshop on Hot Topics in Networks.

[15] NEWSOME, J., KARP, B., SONG, D.: *Polygraph: Automatically Generating Sig-natures for Polymorphic Worms*. 2005, In IEEE Security and Privacy Symposium.

[16] SINGH, S., ESTAN, C., VARGHESE, G., SAVAGE, S.: *Automated Worm Fingerprinting*. 2004, Department of Computer Science and Engineering, University of California, San Diego.

[17] KIM, H.A., KARP, B.: *Autograph: Toward Automated, Distributed Worm Signature Detection*. 2004, In Proceedings of the USENIX Security Symposium.

[18] NEWSOME, J., SONG, D.: *Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software*. 2005, In Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS).

[19] WANG, K., STOLFO, J.: *Anomalous payload-based network intrusion detection*. 2004, Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection.

[20] TANG, Y., CHEN, S.: *Defending Against Internet Worms: A Signature-Based Approach*. 2005, In Proceedings of IEEE INFOCOM'05, Miami, Florida, USA.

[21] YEGNESWARAN, V., GIFFIN, J. T., BARFORD, P., JHA, S.: *An architecture for generating semantic-aware signatures*. 2005, In Proceedings of the 14th USENIX Security Symposium.

[22] LIANG, Z., SEKAR, R.: *Fast and automated generation of attack signatures: A basis for building self-protecting servers*. 2005, In Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS).

[23] LAM, L., CHIUEH, T.: *Automatic extraction of accurate application-specific sandboxing policy*. 2004 In Proceedings of the 7th International Symposium in Recent Advances in Intrusion Detection (RAID).

[24] COSTA, M., CROWCROFT, J., CASTRO, M., ROWSTRON, A., et al.: *Vigilante: End-to-End containment of internet worms*. 2005, ACM Press, Proceedings of the 20th ACM symposium on Operating systems principles, New York, USA.

[25] RABEK, J. C., KHAZAN, R. I., LEWANDOWSKI S. M., CUNNINGHAM, R. K.: *Detection of injected, dynamically generated, and obfuscated malicious code*. 2004, ACM Press, In Proceedings of the 2003 ACM workshop on Rapid Malcode, New York, USA.

[26] DAGON, D., QIN, X., GU, G., LEE, W., GRIZZARD, J. B., LEVINE, J. G., OWEN, H. L.: *Honeystat: Local worm detection using honeypots*. 2004, Springer, In Proceedings of RAID'04, volume 3224.

[27] ANTAL, L.: *Analýza paralelizovaných nástrojů typu honeypot*. 2010, Bakalářská práce, Brno, FIT VUT v Brně.

[28] PORTOKALIDIS, G., SLOWINSKA, A., BOS, H.: *Argos: an Emulator for Fingerprinting Zero-Day Attacks*. 2006, In Proc. ACM SIGOPSEUROSYS'2006.

[29] BARABAS, M., DROZD, M., HANÁČEK, P.: *Behavioral signature generation using shadow honeypot*. Doposud nepublikované.

[30] HOMOLIAK, I.: *Metriky pro detekci útoků v síťovém provozu.* 2012, Diplomová práce, Brno, FIT VUT v Brně.

[31] *About Python.*[online],[cit. 2011-05-08]. Dostupné na URL: <http://www.python.org/about/>

[32] *PostgreSQL 9.1.3 Documentation.* [online],[cit. 2011-05-08]. Dostupné na URL: <http://www.postgresql.org/docs/9.1/static/index.html>

[33] *Manpage of TCPDUMP*. [online],[cit. 2011-05-08]. Dostupné na URL: <http://www.tcpdump.org/tcpdump_man.html>

[34] VARRAZZO, D.: *Psycopg – PostgreSQL database adapter for Python.* [online],[cit. 2011-05-08]. Dostupné na URL: <http://packages.python.org/psycopg2/>

[35] OBERHEIDE, J: *DPKT tutorial.* [online],[cit. 2011-05-08]. Dostupné na URL: <http://jon.oberheide.org/blog/2008/10/15/dpkt-tutorial-2-parsing-a-pcap-file/>

[36] SILVERMAN, J.*: My documentation on DPKT*. [online],[cit. 2011-05-08]. Dostupné na URL: <http://www.commercialventvac.com/dpkt.html>

[37] *TShark – The Wireshark Network Analyzer*. [online],[cit. 2011-05-08]. Dostupné na URL: <http://www.wireshark.org/docs/man-pages/tshark.html>

[38] KHANT, A.: *Virtual Hacking Lab.* [online],[cit. 2011-05-08]. Dostupné na URL: <http://sourceforge.net/projects/virtualhacking/files/_vuln_apps/>

Seznam příloh

Příloha 1: Obsah přiloženého DVD.
Příloha 2: DVD se všemi vytvořenými zdrojovými kódy a instalačními soubory.

Příloha 1 – Obsah přiloženého DVD

Přiložené DVD má následující adresářovou strukturu:

/AttackDetector

/MetricsExtractor	Složka obsahující nástroj pro extrakci metrik z databáze.
`attack_output.csv`	Soubor obsahující testovací metriky získané ze serverové části systému AIPS.
`Config.py`	Konfigurační soubor nástroje.
`CreateFuncCompareMetrics.py`	Skript vytvářející porovnávací uloženou proceduru v databázi.
`CreateLogTable.py`	Pomocný skript vytvářející PostgreSQL tabulku pro logovací data.
`CreateMetricsTable.py`	Pomocný skript vytvářející PostgreSQL tabulku pro metriky.
`CreateTcpdumpTable.py`	Pomocný skript vytvářející PostgreSQL tabulku pro tcpdump data.
`Logging.py`	Modul zajišťující logování informací.
`MetricsComparatorDB.py`	Skript porovnávající metriky – nasazení pro NIDS.
`MetricsComparatorPython.py`	Skript porovnávající metriky – nasazení pro HIDS.
`MetricsToDB.py`	Skript převádějící obsah CSV souboru do databáze.
`output.csv`	Soubor obsahující metriky z lokálně provedených útoků.
`PacketExtractor.py`	Skript extrahující obsah uloženého tcpdump souboru do databáze.
`TrafficCollector.py`	Skript ukládající síťový provoz do souboru.
/AttackDetector_win	Složka obsahující zdrojové kódy nástroje Attack Detector pro platformu Windows.

/diplomova_prace

`diplomova_prace.docx`	Elektronická podoba diplomové práce ve formátu MS Word.
/dumps	Složka obsahující všechny vzorky uloženého síťového provozu.
/results	Složka obsahující výsledné metriky a výstupy jejich porovnání.
/vulnerable_sw	Složka obsahující instalační soubory zranitelného software.
/windows_sw	Složka obsahující instalační soubory aplikací a knihoven potřebných pro běh nástroje na platformě Windows.