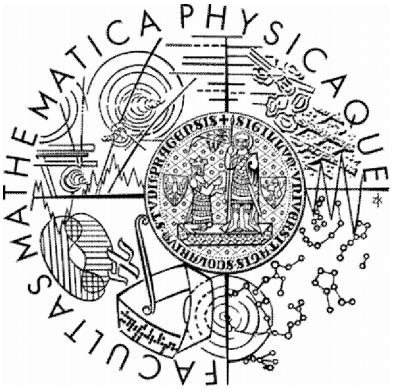
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Martin Choma

Cyclop – Systém detekce počítačů porušujících centrálně stanovena pravidla spravované sítě.

Středisko informatické sítě a laboratoří

Vedoucí bakalářské práce: Dan Lukeš
Studijní program: Informatika, Správa počítačových systémů

2007

Poďakovanie
Na tomto mieste by som rád poďakoval svojmu vedúcemu bakalárskej práce, Danovi Lukešovi, za vedenie tohto projektu.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičaním práce.

V Prahe dňa 25.5.2007 Martin Choma

Názov práce: Cyclop – Systém detekce počítačů porušujících centrálně stanovena pravidla spravované sítě.

Autor: Martin Choma

Katedra (ústav): Středisko informatické sítě a laboratoří

Vedúci bakalárskej práce: Dan Lukeš

e-mail vedúceho: Dan.Lukes@mff.cuni.cz

Abstrakt: Cyclop je jednoduchý IDS (systém na detekciu prienikov) pre malé siete. Umožňuje administrátorovi zachytávať nežiadúcu alebo podozrivú komunikáciu v sieti. To, čo je podozrivé a nežiadúce si konfiguruje správca siete sám v konfiguračnom súbore ako pravidlá. Cyclop dokáže detekovať portscany, pakety obsahujúce určitý reťazec, ktorý je možné zadať vo forme regulárneho výrazu a pakety splňujúce podmienky zadaného pcap filtru (filter, ktorý poznáme z programov tcpdump alebo ethereal). Typické použitie je nasadenie na router s prepnutím sieťovej karty do promiskuitného módu.

Kľúčové slová: IDS, počítačová sieť, administrácia siete, portscan, pattern matching, yacc

Title: Cyclop - Network administrator utility for detecting rule-breaking users

Author: Martin Choma

Department: Network and Labs Management Center

Supervisor: Dan Lukeš

Supervisors email address: Dan.Lukes@mff.cuni.cz

Abstract: Cyclop is intrusion detection system. This tool provide for network administrator way to easily monitor unwanted or potentially unsecure network traffic. Cyclop is able to detect portscans, can perform pattern matching against regular expression and filter traffic with pcap filter (as known from tools like tcpdump or ethereal). All this rules are configurable through smart configuration file**.**

Keywords: IDS, network, network administration, portscan, pattern matching, yacc

Kapitola 1 – Úvod

1.1 Cieľ

Cieľom môjho projektu bolo vytvorenie nástroja, ktorý by napomáhal administrátorom pri správe ich siete. Cyclop sa radí medzi monitorovaco-analytické nástroje. Nesnaží sa vykonávať žiadne protiopatrenia (ako zamedzenia prístupu užívateľa k sieti a pod.). Môže totiž dochádzať k zneužitiu týchto pravidiel, keď útočník umelo vygeneruje pakety, ktoré budú simulovať sieťovú komunikáciu porušujúci pravidlá siete. A ako zdroj týchto paketov určí iný počítač v sieti. Tým by sa nastavené protiopatrenie aplikovalo na nič netušiaceho užívateľa. V takom prípade by bolo potrebné písať pravidlá veľmi rozvážne, aby nedochádzalo k mnohým false positives. Termínom false positives označujeme prípady, keď monitorovací systém zdetekuje bezpečnostnú udalosť na základe úplne legitímnej činnosti užívateľa.

Cyclop preto iba detekuje sieťový tok, ktorý zodpovedá nastaveným pravidlám v konfiguračnom súbore a tieto udalosti loguje do súboru. A je na administrátorovi, aby na základe logov podnikol potrebné protiopatrenia, ak to uzná za vhodné. Použitý konfiguračný jazyka je prostý a ľahko pochopiteľný, ale napriek tomu sa s ním dá vyjadriť, a teda monitorovať, veľká množina možných ohrození. Niekoľko príkladov ukážem neskôr.

1.2 Siete

V tejto kapitole v krátkosti uvediem zopár všeobecných informácií o sieťach a najznámejčích protokoloch.

1.2.1 TCP/IP

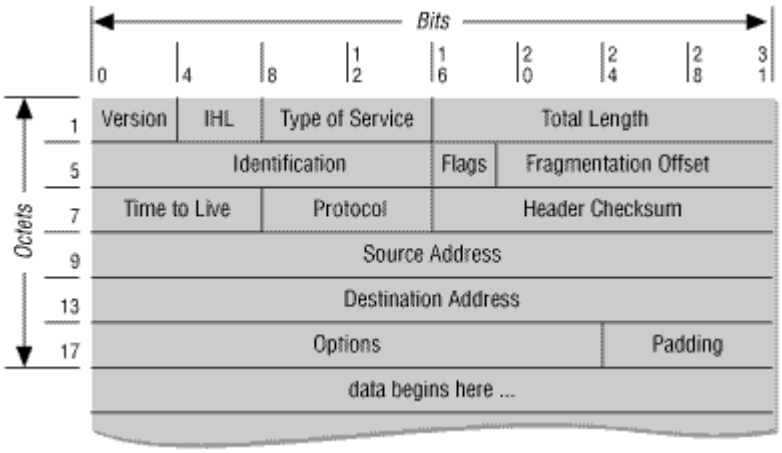
TCP/IP je rodina protokolov. Tie základné (IP, TCP, UDP) sa používajú v mnohých aplikáciách. Ostatné sa používajú na vykonanie špecifických úloh. Ako napríklad preposielanie súborov, posielanie mailov atď. TCP/IP má vlastný systém adresovania, ktorý umožňuje identifikovať a adresovať uzly(zariadenia) a dielčie entity (služby atd.) bez znalosti detailov ich pripojenia. TCP/IP je nezávislá na

fyzických (linkových) technológiách. V TCP/IP je zabudovaná podpora internetworkingu a smerovania. V zásade, TCP/IP aplikácie používajú 4 vrstvy. Každá je zodpovedná za niečo iné:

- **Aplikačná vrstva** - sem patria aplikačné protokoly ako SMTP, FTP, DNS, HTTP, SIP atď. Dáta sú poslané v špecifickom formáte a transportným protokolom poslané ďalej.
- **Transportná vrstva** - úlohou transportnej vrstvy je kontrola prenosu paketov. Transportná vrstva zo sebou prináša porty, takže v tejto vrstve už je možné rozlišovať entity aplikačnej vrstvy. Patria sem protokoly TCP a UDP. UDP je maximálne jednoduchou nadstavbou nad IP. Používa sa v takých aplikáciách, ktoré potrebujú čo najrýchlejšiu a najefektívnejšiu komunikáciu, pretože UDP nie je zaťažené réžiou ako TCP; poskytuje nespoľahlivé prenosové služby, funguje nespojovane a komunikácia je bezstavová. Väčšia réžia TCP je daň za služby, ktoré tento protokol poskytuje. TCP totiž funguje tak, že nadviaže a udržuje spojenie medzi dvoma uzlami siete. Takže protokol musí zabezpečiť efektívne fungovanie prenosu (používa systém kontinuálneho potvrdzovania), korektné naviazanie spojenia (rieši deadlocky a straty pokusov o naviazanie spojenia) a korektné ukončenie spojenia (zaisťuje, že pred ukončením spojenia sú prenesené všetky odoslané dáta). Protokol ďalej ošetruje chyby pri prenosoch, duplicity, straty a garantuje poradie doručovaných dát.
- **Sieťová vrstva** – súčasťou sieťovej vrstvy je aj mechanizmus adresovania a mechanizmy, ktoré prekladajú medzi fyzickými a IP adresami (ARP a RARP protokoly). Mechanizmy fragmentácie i protokoly na podporu fungovania sieťovej vrstvy (ICMP). So sieťovou vrstvou úzko súvisia protokoly podporujúce smerovanie (RIP, OSPF, IGP, EGP ...), mechanizmy prideľovania IP adries, mechanizmy prekladu medzi symbolickými doménovými menami a IP adresami, mechanizmy prekladu adries (NAT), koncept privátnych IP adries, mechanizmy delenia adries a združovania adries (subnetting, supernetting, CIDR) a bezpečnostné mechanizmy (IPSec).
- **Linková vrstva** - úlohou je prenášanie rámcov skrz fyzické média. Patrí sem Ethernet, PPP, Wi-Fi, Token ring, ATM, FDDI, Frame Relay.

Protokol IP

Protokol IP tvorí základňu pre vyššie prenosové protokoly a pre služobné protokoly Internetu. Sám sa opiera o protokoly sietí, ktoré prenášajú jeho pakety. (Ethernet, FDDI, PPP, ATM a ďalšie). Základným elementom protokolu IP je štruktúra prenášaného datagramu - IP paketu. IP paket je tvorený záhlavím a vlastnými prenášanými dátami. Záhlavie je zabezpečené jednoduchým kontrólnym súčtom, prenášané dáta nie sú zabezpečené. Štruktúru IP paketu uvádza Obr.1.1. Jednotlivé polia hlavičky IP paketu (alebo fragmentu) majú nasledujúcu funkciu :



Obr. 1.1 Štruktúra IP paketu

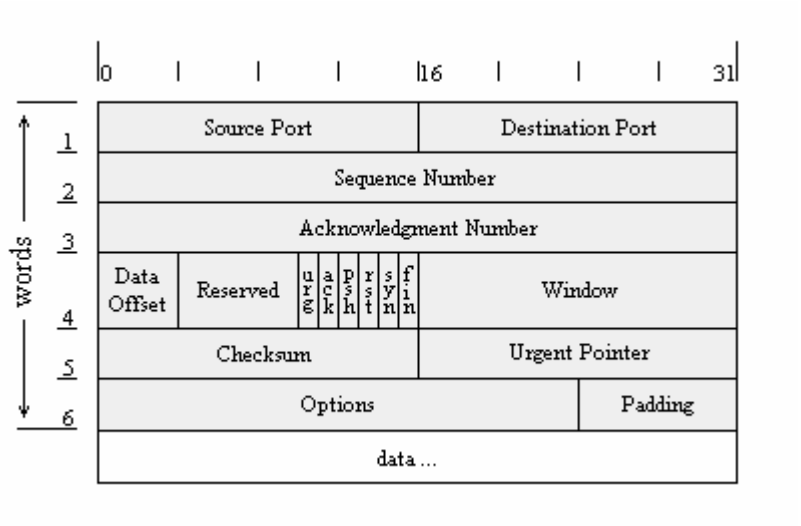
- **Version -** štvorbitové pole, definuje verziu IP protokolu
- **IHL -** štvorbitové pole, udáva dĺžku hlavičky v 32 bytových slovách, nevyužité slabiky sú označované ako výplň (PADD).
- **Type of Service -** osembitové pole udáva požiadavku na prioritu IP paketu (0-7), minimálne oneskorenie, kapacitu a vysokú spoľahlivosť. Tieto požiadavky dovolia vybrať najvýhodnejšiu cestu. Internet však ich splnenie negarantuje.
- **Total Length -** šestnásťbitové pole udáva celkovú dĺžku IP paketu (fragmentu) v slabikách, zahrňuje hlavičku aj dáta, najväčšia možná dĺžka IP paketu je 65535 slabík.
- **Identification -** šestnásťbitové pole identifikuje IP paket a podporuje fragmentáciu

- **Flags -** tri jednobitové príznaky podporujú fragmentáciu (zákaz fragmentácie, identifikácia posledného fragmentu v IP pakete)
- **Fragmentation Offset -** trinásťbitová relatívna adresa fragmentu v IP pakete v násobkoch ôsmich slabík.
- **Time to Live -** osembitové pole udávajúce dobu života IP, každý počítač, cez ktorý paket prejde zníži túto hodnotu o jedničku, pri nule je IP paket (fragment) likvidovaný.
- **Protocol -** šestnásťbitová identifikácia protokolu vyššej vrstvy (TCP, UDP)
- **Header Checksum -** šestnásťbitový kontrólný súčet záhlavia paketu (fragmentu).
- **Source Address -** tridsaťdvabitová IP adresa odosielateľa IP paketu.
- **Destination Address -** tridsaťdvabitová IP adresa adresáta IP paketu.
- **Options -** prípadné riadiace a ladiace funkcie, meranie v sieti.
- **Padding -** výplňové nulové slabiky.
- **Data -** prenášané dáta.

Detailný popis IP protokolu je možné nájsť v RFC 791 ([2]).

TCP

TCP je, dovolím si tvrdiť, najpoužívanejší transportný protokol, preto ho na tomto mieste priblížim trošku viac. Kontroluje prenos paketov - nesmeruje ich (za smerovanie je zodpovedný IP protokol), len si udržuje informáciu o tom, čo bolo poslané. V prípade, že sa niečo nepodarí "pretlačiť" na druhú stranu, tak sa o to pokúsi znovu. TCP funguje totiž na princípe „maximálnej snahy". Keďže tieto funkčnosti potrebuje veľa aplikácií, tak sú logicky v samostatnom protokole ako keby si to mal každý protokol zariaďovať sám. Tak ako veľa aplikácii využíva možnosti TCP, tak využíva TCP možnosti IP. Jednotlivé polia hlavičky TCP paketu majú tento význam:



Obr. 1.2: Štruktúra hlavičky TCP paketu

- **Source Port -** 16 bitový zdrojový port. Čísla portov sú rozdelené do troch kategórií (0-1023)
- **Destination Port -** 16 bitový cieľový port.
- **Sequence Number -** 32 bitov pre sekvenčné číslo prvého dátového bajtu v pakete. Ak je nastavený SYN flag, tak je to počiatočné sekvenčné číslo a prvý dátový bajt je počiatočné sekvenčné číslo + 1.
- **Acknowledgment Number -** 32 bitové potvrdzovacie číslo. Ak je nastavený ACK flag, tak táto položka obsahuje hodnotu ďalšieho sekvenčného čísla, ktoré odosieľateľ očakáva, že príjme v ďalšom kroku. Keď už sa raz podarilo vytvoriť spojenie, tak je toto číslo vyplnené stále.
- **Data Offset -** 4 bitové číslo udávajúce počet 32 bitových slov v hlavičke TCP paketu. Indikuje začiatok dát. Hlavička TCP paketu je vždy násobkom 32 bitov.
- **Reserved -** 3 bity, ktoré sú momentálne nevyužité a rezervované pre budúce využitie. Musia byť nastavené na nulu.
- **Control Bits.**
    - **U, URG.** Ak je nastavený, tak príjemca by mal data považovať za urgentné, tj mali by sa prednostne spracovať.
    - **A, ACK.** Ak je nastavený, tak pole a*cknowledgment number* obsahuje platnú hodnotu.

- **P, PSH.** Ak je nastavený, tak príjemca by mal doručiť tento paket cieľovej aplikácií v čo najkratšom čase. Príkladom na jeho použitie je prioritné spracovanie paketu obsahujúceho požiadavku na ukončenie programu (Ctrl-C)
- **R, RST.** Odosieľateľ oznamuje týmto flagom, že ukončuje spojenie a že naalokovaná pamäť môže byť uvoľnená.
- **S, SYN.** Využíva sa počas vytvárania spojenia medzi odosielateľom a príjemcom.
- **F, FIN.** Odosielateľ týmto flagom oznamuje, že už nemá žiadne data k odoslaniu.

- **Window -** 16 bitové číslo udávajúce počet bytov, ktoré môže odosielateľ poslať bez toho, aby dostal potvrdzujúci paket o ich prijatí. Hlavným dôvodom zavedenia tejto hodnoty je riešenie otázky zahltenia siete.
- **Checksum -** kontrólny súčet, ktorý sa vypočítava z informácií z hlavičky IP a TCP paketu a dát.
- **UrgentPointer -** ak je nastavený *URG* flag, tak táto položka obsahuje sekvenčné číslo posledného bajtu v sekvencii urgentných dát.

Detailný popis TCP protokolu je možné nájsť v RFC 793 ([3]).

1.3 Bezpečnosť servera v sieti

Každý počítač sa v okamžiku pripojenia do siete stáva potenciálnym terčom útokov. Uvediem zoznam niekoľkých najčastejších útokov, ktoré môžu byť vykonané. Väčšinu z nich dokáže cyclop úspešne detekovať.

Portscan nie je útokom v pravom slova zmysle, ale často ho predznamenáva. Indikuje útočníkov záujem o hostiteľský počítač. Portscan sa dá vykonať voľne dostupnými nástrojmi ako napríklad *nmap*. Zisťuje sa ním aké služby na danom počítači bežia. Na základe tohto zistenia útočník zvolí ďalší postup útoku. Princíp nástrojov vykonávajúcich portscany je jednoduchý – skúšajú sa pripojiť na zadaný rozsah portov a v závislosti na odpovediach servera si odvodia, či je port otvorený. A keďže nižšie čísla portov sú štandardne pridelené nejakej známej službe, tak určia aj službu, ktorá by tam mohla bežať.

DoS útok (denial of service) alebo odoprenie služby má za cieľ, ako názov napovedá, znefunkčniť určitú službu alebo odpojiť cieľ útoku od siete. Existuje niekoľko metód ako DoS útok vykonať. Pre ilustráciu uvediem tri známe prevedenia DoS útoku:

- **SYN flood útok –** pošle sa množstvo TCP/SYN paketov, typicky s neplatnou zdrojovou adresou. Každý z týchto paketov je obslúžený ako požiadavka na spojenie. To spôsobí, že server pre každý paket vytvoří „polo-otvorené" spojenie a pošle TCP/SYN-ACK paket a čaká na potvrdzujúci TCP/ACK paket ako odpoveď zo zdrojovej adresy. Samozrejme každé takéto „polo-otvorené" spojenie pohlcuje prostriedky servera a obmedzuje možnosti ďalších pripojení legitímnych užívateľov.
- **UDP flood -** útok je založený na zasielaní UDP paketov s falošnou adresou odosielateľa. Keďže UDP je protokol bez nadväzovania spojenia, útočník zasiela UDP pakety na náhodné porty cieľového počítača. Ak počítač príjme UDP paket adresovaný na port, ktorý nevyužíva nijaká aplikácia, zašle odosielateľovi ICMP paket s hlásením o chybe. Ak útočník použije dostatočne veľké množstvo UDP paketov, na sieti vznikne veľká (dvojnásobná) prevádzka a počítač zamrzne.
- **Smurf attack –** na útok využívá umelo vytvorený echo ping paket. Útočník nastaví tomuto paketu ako cieľovú adresu broadcast adresu a ako zdrojovú adresu adresu cieľa. To čo sa deje po poslaní týchto paketov je už asi zrejmé. V prípade, že sa tento paket podarí rozoslať na broadcast [1] tak všetky hosty, ktoré príjmu ping request sa budú samozrejme na neho odpovedať echo odpoveďou a tým sa cieľový host zahltí.

1.4 IDS

Samozrejme, že hlavnou časťou bezpečnostnej politiky serverov je ochrana pomocou firewallov alebo proxy serverov. Každý sieťový administrátor by mal

[1] Dnes už je toto nebezpečenstvo pomerne známe a preto aj ošetrené. Napríklad na zabezpečenie siete, ktorá používa Cisco router, pred touto hrozbou stačí použiť príkaz `no ip directed-broadcast`.

ovšem zvážiť nasadenie taktiež nejakého IDS systému, ktorý môže odhaliť nebezpečenstá, ktoré firewall nezablokuje.

1.4.1 Úvod

IDS (Intrusion detection system) je systém na detekciu prieniku do siete. Vo všeobecnosti detekuje nežiaducu sieťovú komunikáciu. Existuje mnoho rôznych spôsobov ako takúto komunikáciu detekovať a niektoré z nich popíšem v nasledujúcej časti. Od IDS sa očakáva, že bude detekovať všetky podozrivé aktivity, ktoré nie je možné odchytiť klasickým firewallom. To sa týka rôznych útokov voči zraniteľným službám, útokov na aplikácie vhodne zostavenými dátami ako aj útoky červov, trójskych koňov a vírusov. Je niekoľko spôsobov ako rozdeliť IDS; v závislosti na metóde detekovania bezpečnostných udalostí ako aj na umiestnení.

- **NIDS** (Network Intrusion Detection Systems) je systém strategicky umiestnený (typicky na routeri), tak aby mohol monitorovať všetky data idúce do a zo siete. V prípade veľkého objemu dát môže toto monitorovanie spôsobiť spomalenie celého systému. V tom prípade sa NIDS umiestni na samostatnú stanicu a router bude preposielať pakety na tento stroj.
- **HIDS** (Host Intrusion Detection Systems) bežia na samostatných strojoch alebo zariadeniach na sieti. Monitorujú len prichádzajúce a odchádzajúce pakety zo zariadení, na ktorých sú nasadené.
- **Signatúrové IDS** - IDS založené na porovnávaní paketov so sadou pripravených signatúr alebo atribútov dobre známych ohrození. Je to podobný spôsob ako detekuje vírusy väčšina antivírových programov. Samozrejme existuje určité časové oneskorenie od objavenia ohrozenia po zaradenie pravidla do IDS. Počas tohto intervalu IDS nebude schopné detekovať toto ohrozenie.
- **Anomálne IDS** monitoruje dátový prúd a porovnáva ho voči normálnej sieťovej aktivite. Normálnou sieťovou aktivitou sa mysli predefinovaná sada údajov typu; aký typ prenosu sa zvyčajne používa, aké protokoly sa používajú, aké porty a zariadenia sa medzi sebou

zvyčajne pripájajú. V prípade, že IDS zdetekuje rozdiel voči tejto normálnej aktivite, upozorní na to.

- **Pasívne IDS** jednoducho zdetekuje a ohlási administrátorovi podozrivú udalosť na sieti. To znamená, že sa nesnaží o žiadne protiopatrenie a je na administrátorovi, aby sa o to postaral.
- **Proaktívne IDS** nielenže zdetekuje podozrivý dátový tok a upozorní administrátora, ale taktiež učiní preddefinované kroky na zamedzenie ďalšieho nebezpečenstva – typicky to znamená zablokovanie akejkoľvek ďalšej sieťovej aktivity stroja, ktorý je zdrojom tejto podozrivej aktivity.

1.4.2 False positives, false negatives

Sú to veličiny, ktorými môžeme merať a porovnávať kvality rôznych systémov. False positives nastane, keď systém vygeneruje alarm aj keď žiaden pokus o prienik nenastal. False negatives, naopak označujeme situáciu, keď prienik nastal a IDS to nezaregistroval. Jedným z hlavných príčin, prečo sa falošné alarmy vyskytujú je bezstavovosť IDS systémov (pri monitorovaní paketov sa hľadí iba na aktuálny paket, neberie sa do úvahy predchádzajúci sietový tok). Väčšina nástrojov, tak ako aj cyclop detekuje prieniky pomocou signatúr (preddefinovaná sada pravidiel). V prípade, že pravidlo nie je dobre navrhnuté, tak môže spôsobiť generovanie false negatives. Napríklad taká údržba siete (aktivita administrátora obecne) sa môže mylne považovať za „nenormálnu“ alebo anomálnu a zalogovať sa ako bezpečnostný incident.

1.4.3 Existujúce nástroje

Snort je jednoznačne najznámejší a široko používaný IDS systém. Je dostupný pre mnoho platforiem a operačných systémov zahrňujúci Linux aj Windows. Snort na svojich stránkach ponúka veľké množstvo signatúr. Kombinuje výhody signatúrovej, protokolovej a anomálnej detekcie. Vyvíja sa už deväť rokov a za ten čas sa stal vlastne štandardom pre IDS a IPS (Intrusion prevention system) systémy. Okolo snortu sa na internete vytvorila pomerne veľká komunita, čo samozrejme ocení užívateľ v prípade riešenia problémov.

Bro je ďalšie open-sourcové IDS určené pre UNIX-like systémy. Bol vytvorený pre výskumné účely analýzy paketov a detekciu prienikov vo výskumnom centre Lawrence Berkeley National Lab. Má zadefinovaný vlastný konfiguračný jazyk a preddefinovanú sadu pravidiel. Bro môže byť bez obáv nasadený na gigabitové siete, keďže s týmto zámerom bol navrhovaný. Tak ako aj snort aj bro využíva na odchytávanie paketov pcap knižnicu. ([8])

Securepoint Intrusion Detection System je freewarové IDS určené iba pre OS Windows. Dodáva sa s viac než 800 pravidlami, ktoré je samozrejme možné editovať. Umožňuje analýzu sieťového toku a umožnuje odhaliť vírusy a trójske kone. Nástroj sa ovláda pomocou GUI.

Prelude sa pasuje za hybridný IDS framework. Hybridný znamená, že analyzuje informácie zo siete ako aj priamo z hostu. Umožňuje všetkým bezpečnostným aplikáciám, či už open-sourcovým alebo proprietárnym, zaznamenávať údaje do centralizovaného systému. Na to využíva jednotný jazyk na zápis udalostí. Prelude je distribuovaný pod GPL licenciou. Beží pod Linuxom a *BSD systémami, ale aj ďalšími.

Kapitola 2 – Užívateľská dokumentácia

V tejto kapitole ukážem, čo je môžné od cyclopa čakať a ako ho možno používať. Cyclop je proces bežiaci na pozadí. Neexistuje nad ním žiadne grafické rozhranie. Rozhranie medzi užívateľom a programom tvorí konfiguračný súbor ako vstup, kde sa definujú filtre a ostatné nastaviteľné parametre programu a logovací súbor ako výstup.

2.1 Funkcionality

Pri rozhodovaní o tom, čo zahrniem do funkcionalít cyclopa som bral do úvahy užitočnosť a náročnosť implementácie. Nakoniec sa do cyclopa dostalo detekovanie portscanu a hľadanie reťazcov v dátovej časti paketu a filtrovanie paketov pomocou pcap filtra.

2.1.1 Portscany

Skenovanie portov je v drtivej väčšine prvý krok útočníka pred pokusom o preniknutie do siete. Týmto spôsobom zistí aké služby bežia na danom serveri a teda aké možnosti na preniknutie existujú. Čiže každé zdetekovanie skenovania portov je pre administrátora varovným signálom a je vysoko pravdepodobné, že bude nasledovať ďalšia aktivita tohto „zvedavého“ užívateľa a mal by si dať na neho pozor.

Najznámejším skenerom portov je *nmap*, ktorý vytvoril legendárny programátor známy pod menom Fyodor. Nmap poskytuje skenovacie techniky na zistenie dostupných služieb a operačného systému bežiaceho na skenovanom hoste. Pre ilustráciu uvediem výčet niekoľkých z nich; TCP SYN, Connect(), UDP Scan, TCP null a ďalšie. Pomocou nmap-u je možné vykonať portscan aj s podvrhnutím zdrojovej, resp. zdrojových adries (tzv. decoy) pomocou prepínača *–D*. A zaujímavá je aj možnosť špecifikovať časový interval medzi pokusmi pomocou prepínača *--scan-delay*, čo je vhodné hlavne na predídenie detekcie IDSkom. Nmap je typicky súčasťou linuxových distribúcií. Detailnejšie informácie o možnostiach nmapu sa dajú nájsť v dobre napísanej manuálovej stránke k nmapu. ([7])

2.1.2 Pattern-matching

Umožňuje administrátorovi detekovať pakety, ktoré obsahujú v dátovej časti určitý reťazec. Tento reťazec je možné popísať regulárnym výrazom (konkrétne POSIX rozšírenie regulárneho výrazu). Hľadané regulárne výrazy sa deklarujú v konfiguračnom súbore v premennej *regex*. Týmto spôsobom je možné detekovať napríklad prístup k niektorým nežiadúcim webovým stránkam. Nevýhodou je, že zdetekuje nie len prístup k tejto stránke, ale aj stránky, na ktorých sa inkriminovaný odkaz nachádza.

2.1.3 Paket Filtering

Umožňuje výber paketov na základe portu, čísla siete, protokolu a podobne. Filter sa zadáva vo formáte pcap filtru, ktorý sa používa aj v *tcpdump-e* alebo *ethereal-e*.

Pcap výraz pozostáva z aspoň jednej primitívy. Primitívy sa skladajú z ***id*** a niekoľkých kvalifikátorov. ***Id*** udáva čoho sa primitíva týka, môže to byť počítač, sieť, port alebo rozsah portov. Kvalifikátor môže byť jeden z týchto troch druhov;

- **typ id -** špecifikuje, čo ***id*** popisuje. Možné hodnoty sú ***host***, ***net*** a ***port***. Ak nie je zadaný žiaden kvalifikátor, použije sa ***host***.
  Príklady: `*host foo*', `*net 128.3*', `*port 20*'.
- **smer prenosu** - špecifikuje smer prenosu ku alebo od ***id***. Je možné použiť aj logické spojky ***and*** alebo ***or***. Takže možné smery sú ***src***, ***dst***, ***src or dst*** a ***src and dst***. Ak nie je špecifikovaný žiaden kvalifikátor smeru, tak sa použije ***src or dst***.
  Príklady: `src foo', `dst net 128.3', `src or dst port ftp-data'.
- **protokol -** obmedzuje výber filtra na určitý protokol. Možné protokoly sú: ***ether***, ***fddi***, ***tr***, ***wlan***, ***ip***, ***ip6***, ***arp***, ***rarp***, ***decnet***, ***tcp*** and ***udp***. Napr. `*ether src foo*', `*arp net 128.3*', `*tcp port 21*'. Ak nie je špecifikovaný žiaden ***proto*** kvalifikátor – všetky protokoly konzistentné s typom sú použité. Napr. `*src foo*' znamená `*(ip or arp or rarp) src foo*', `*net bar*' znamená `*(ip or arp or rarp) net bar*' a `*port 53*' sa interpretuje ako `*(tcp or udp) port 53*'

Detailný popis toho, čo môže pcap filter obsahovať je možné nájsť v manuálovej stránke tcpdump-u. ([4]).

2.2 Inštalácia a spustenie

Pre úspešné skompilovanie zdrojových súborov je potrebné mať nainštalovanú knižnicu libpcap – (Packet Capture library), ktorá poskytuje API na odchytávanie paketov zo siete (aj v promiskuitnom móde). Je súčasťou väčšiny Linuxových distribúcií. V prípade, že vo Vašej distribúcií chýba, tak sa da stiahnuť zo stránky *http://www.tcpdump.org/*. Ak je táto knižnica nainštalovaná, tak po zadaní *make all* v adresári so zdrojovými súbormi (adresár *src*) sa cyclop skompiluje a je pripravený na inštaláciu. Inštalácia sa vykoná po spustení *make install* a jej výsledkom je vytvorenie adresára *run* o úroveň vyššie, ktorý obsahuje spustiteľný súbor *cyclopd*, konfiguračný súbor *cyclop.conf* a logovací súbor *cyclop.log*.

Na to, aby mohol cyclop korektne fungovať musí mať užívateľ, pod ktorým sa proces spúšťa dostatočné práva na odchytávanie paketov zo sieťového zariadenia, na väčšine systémov ich bude mať super užívateľ, **root**.

2.3 Konfiguračný súbor

Konfiguračný súbor je základným rozhraním medzi užívateľom a programom. Preto som mu prikladal veľkú pozornosť. Pre konfiguračný súbor je zadefinovaná presná syntax, ktorú je potrebné dodržiavať. Nesmierne užitočné je, že parser dokáže v prípade chyby syntaxe konfiguračného súboru určiť riadok, kde sa chyba nachádza;

```
cyclop.conf: error in filter 1, line 9: syntax error
```

2.3.1 Jazyka konfiguračného súboru

Konfiguračný súbor sa skladá z časti obecných parametrov programu a z definície filtrov.

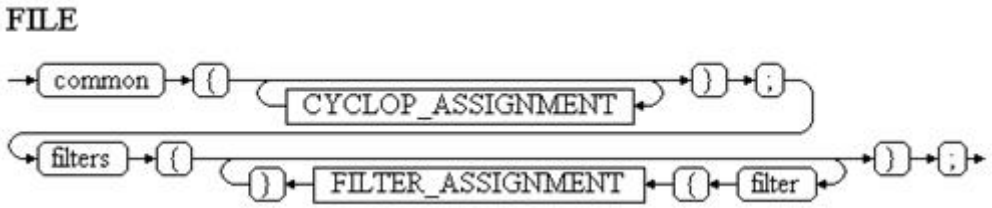


Diagram 1: Štruktúra konfiguračného súboru.

Tak ako ukazuje *Diagram 1* záleží na poradí, tzn. najprv sa očakáva *common* deklarácia a až po nej *filters* deklarácia

2.3.2 Nastavenie programu

Common sekcia obsahuje nastavenie premenných, ktoré sa vzťahujú na celý program.

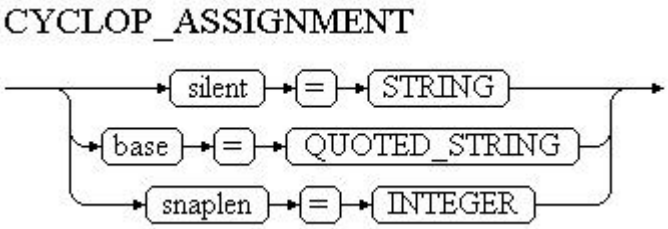


Diagram 2: Štruktúra nastavenia common sekcie

base = *base_filter;*

Reťazec obsahujúci pcap filter. Tento filter sa aplikuje na všetky odchytené pakety a do spracovania programu sa posunú len pakety, ktoré odpovedajú tomuto filtru.

snaplen = *data_to_retrieve;*

Číslo *data_to_retrieve* určuje maximálny počet bytov, ktoré cyclop z paketu odchytí. V prípade vyhľadávania reťazca v pakete je lepšie túto hodnotu nenastavovať.

2.3.3 Nastavenie filtrov

Konfiguračna sekcia *filters* obsahuje nastavenie jednotlivých filtrov

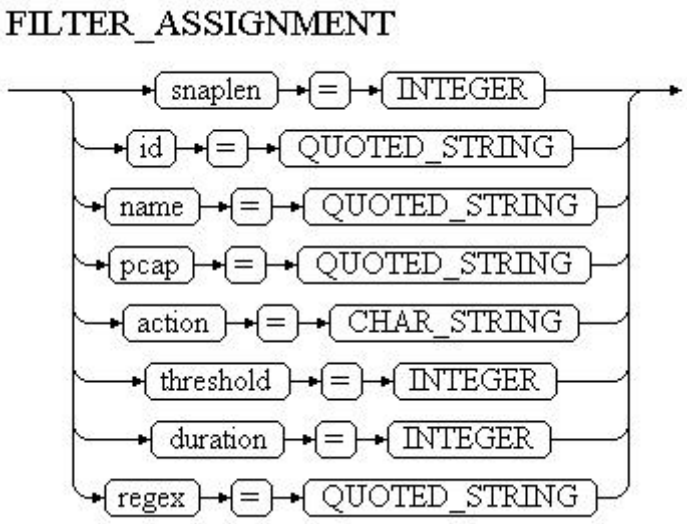


Diagram 3: Syntax filter sekcie.

Tak ako ukazuje *Diagram 3* každý popis filtra pozostáva z kľúčového slova *filter*, za ktorým následuje deklarácia filtra uzavretá v zložených zátvorkách a ukončená bodkočiarkou. Biele znaky, ktoré nie sú súčasťou reťazca v úvodzovkách sa ignorujú. Filter môže obsahovať tieto nastavenia:

action = *action* **;**

action špecifikuje čo má daný filter s paketom, ktorý je pre neho určený, spraviť. Filter sa na paket aplikuje, ak paket splní podmienku *base* pcap filtru a pcap filtru nastaveného na cyclop pravidle. Možné hodnoty sú:

- **IGNORE** - ignoruje ich. IGNORE filter má prednosť pred inými filtrami.
- **LOG** - zaloguje paket.
- **NULL** - nerob nič s paketmi.
- **SCAN** – spracuj paket modulom, ktorý detekuje portscany. Má zmysel iba ak deklarácia filtru obsahuje aj *threshold* a *duration* deklaráciu. Nájdené portscany zaloguj.

- **SEARCH** - zaloguj pakety, ktoré obsahujú daný vyhľadávací reťazec. Má zmysel iba ak je vo filtri zadaný aj *regex*

id = *identification* **;**

identification je reťazec, ktorý slúži na nastavenie unikátneho identifikátora pre filter.

name = *name* **;**

Nastaví meno pre filter. Toto meno bude zobrazované pri logoch a vo výpisoch, preto treba voliť výstižné názvy, aby sa ľahšie tieto záznamy dohľadávali.

pcap = *pcap_filter* ***;***

Reťazec, ktorý odpovedá pcap filtru. Filter bude analyzovať pakety, ktoré odpovedajú tomuto filtru. Je to povinný parameter.

duration = *seconds* ***;***

Má význam len pre *action = SCAN* pravidlá. Určuje čas v sekundách, ktorý filter pokrýva. Viď taktiež *threshold* nastavenie.

threshold = *count* ;

Má význam len pre *action = SCAN* pravidlá. Určuje počet paketov, ktorý stačí na ohlásenie portscanu. Podmienkou však je, že sa tieto pakety odchytili v priebehu *duration* sekúnd.

regex = *regex* **;**

Definuje regulárny výraz. *regex* musí byť validný POSIX extended regulárny výraz. Toto nastavenie má zmysel iba pre *action = SEARCH* pravidlá.

2.4 Parametre programu

Okrem nastavení v konfiguračnom súbore je možné cyclop parametrizovať aj prepínačmi zadanými pri spustení programu.

Zoznam prepínačov:

- **-h** Zobrazí výpis prepínačov programu a ich krátky a výstižný popis.
- **-L** *log_file* **-** Chybové hlášky a alarmy sa budú logovat do *log_file*. Defaultne sa loguje do *cyclop.log* súboru v aktuálnom adresári.
- **-C** *conf_file* **-** Ako konfiguračný súbor sa použije *conf_file*. Defaultne sa hľadá konfiguračný súbor v aktuálnom adresári s názvom *cyclop.conf*.

- **-c** *count* Odchytí a spracuje sa iba *count* paketov. Defaultne cyclop nie je obmedzený počtom.
- **-p** Nastaví, aby cyclop bežal v **ne**promiskuitnom móde. To znamená, že bude spracuvávať len pakety, ktoré sú určené pre host, na ktorom cyclop beží. To nie je typické využitie cyclopu. Skôr sa predpokladá, že bude žiadúce, aby bežal v promiskuitnom móde, preto je defaultne zapnutý promiskuitný režim.

- **-f** *input_file* Nastaví, aby cyclop neodchytával reálne pakety zo siete, ale vstupom budú pakety zo „savefilu". Takýto savefile si môžeme jednoducho pripraviť pomocou jedného z programov tcpdump, tethereal alebo pomocou wireshark (v minulosti známy ako ethereal).

2.5 Logovací súbor

Ak sa prepínačom *–L* nešpecifikuje ináč, tak sa loguje do súboru *cyclop.log* v aktuálnom adresári. Vo Výpise 2.1 môžeme vidieť príklad logovacieho súboru. V prvom stĺpci sa zobrazuje meno filtra, ktoré sme špecifikovali nastavením *name* v konfiguračnom súbore. Druhý stĺpec je čas, v ktorom bol paket odchytený. Potom nasleduje zdrojová adresa, port, cieľová adresa a port. Posledný stĺpec je číselná hodnota protokolu paketu.

```
Test2 1152379721.288101 192.168.9.197:3338 -> 82.254.127.221:4662 6
Test1 1152379721.288101 192.168.10.1:2536 -> 82.229.212.22:21921 17
Test2 1152379721.288101 192.168.10.1:2536 -> 82.229.212.22:21921 17
Test1 1152379721.288102 192.168.11.197:4414 -> 81.197.120.45:40408 6
Test2 1152379721.288102 192.168.11.197:4414 -> 81.197.120.45:40408 6
```

Výpis 2.1: Ukážka logovacieho súboru

2.6 Príklady nastavení filtrov

V tejto kapitole ukážem sériu príkladov, ktoré môžu slúžiť pre predstavu, ako je možné cyclop nastaviť. Nasledujúci jednoduchý príklad zaloguje všetky pakety s ip protokolom.

```
filter
{
        id = "EXMP1";
```

```
        name = "Hello World";
        action = LOG;
        pcap = "ip";
};
```

Tento filter má za úlohu logovať všetky pokusy o nadviazanie spojenia na port 25 z ktoréhokoľvek hostu v sieti 192.168.1.0/24. Množiace sa takéto pokusy môžu indikovať prítomnosť víru na niektorom z hostov.

```
filter
{
        id = "EXMP2";
        name = "Mail virus";
        action = LOG;
        pcap = " ip and (tcp[13] & 2 != 0) and (dst port 25) and
        (dst net 192.168.1.0 mask 255.255.255.0)";
};
```

Tento filter slúži na monitorovanie portscanov vykonaných na host `192.168.1.10`. V tomto prípade sme nastavili, že za portscan budeme považovať to, ak sa v priebehu 4 sekúnd vyskytne viac ako 7 požiadavkov na spojenie.

```
filter
{
        name = "Port scan";
        id = " EXMP3";
        pcap = "(tcp or udp) and dst host 192.168.1.10";
        action = SCAN;
        threshold = 7;
        duration = 4;
};
```

Predstavme si, že máme sieť. Sieť máme navrhnutú tak ,že v demilitarizovanej zóne sa nachádza webový server a smtp server. Na webovom serveri sme povolili len port 80 a port 22 pre vzdialenú administráciu. Budeme chcieť logovať akýkoľvek dátový tok, ktorý prichádza na tento stroj mimo tieto porty, pretože to je podozrivé a môže to napríklad byť skenovanie portov útočníkom. Pre tieto účely nám poslúži tento filter:

```
filter
{
        id = "EXMP4;
        name = "Non – HTTP traffic to web server";
        action = LOG;
        pcap = "dst host 192.168.1.2 and not (tcp and dst port 80
        and dst port 22)";
```

```
};
```

Čisto analogicky môžeme nastaviť filter aj pre SMTP server.

```
filter
{
        id = " EXMP5";
        name = "Non-SMTP traffic to SMTP server";
        action = LOG;
        pcap = "dst host 192.168.1.3 and not (tcp and dst port 25)";
};
```

Poprípade podobné pravidlá pre DNS servre.

```
filter
{
        id = " EXMP6";
        name = "Non-DNS traffic to DNS server";
        action = LOG;
        pcap = "dst host 10.1.1.20 and not (udp and port 53)
        and not (tcp and dst port 53 and (src host 10.2.2.15 or
        src host 10.3.3.15))";
};

filter
{
        id = " EXMP7";
        name = "DNS traffic to non-DNS server";
        action = LOG;
        pcap = "(dst net 10.1.1.0 mask 255.255.255.0) and (not host
        10.1.1.20) and (dst port 53) and (udp or tcp)";
};
```

Pravidlo na odchytávanie TCP paketov z portu 80 alebo 453 a obsahujúcich jeden z reťazcov “http**://**” alebo “https**://**” by mohlo vyzerať takto:

```
filter
{
        id = "EXMP8";
        name = "Example 2c (TCP regex search)";
        action = SEARCH;
        regex = "http[s]*://";
        pcap = "tcp and (dst port 80 or dst port 443)";
};
```

Kapitola 3 – Programátorská dokumentácia

V tejto časti budem postupne predstavovať jednotlivé časti programu a spôsob ako som ich naimplementoval. Základný princíp fungovania programu je priamočiary. Cyclop zbiera pakety pomocou knižnice pcap priamo zo siete alebo zo súboru a podľa nastavenia filtrov v konfiguračnom súbore sa aplikuje alebo neaplikuje príslušná obslužná callback funkcia. Cyclop som napísal v jazyku C a je určený pre Linux. Využíva knižnice pcap a regex a utility flex a yacc na vygenerovanie parsera pre konfiguračný súbor.

3.1 Packet capture library (pcap)

Je API na odchytávanie paketov zo sieťového zariadenia. Pcap je veľmi populárny a používajú ho napríklad programy wireshark, tcpdump, snort alebo nmap. V krátkosti predstavím niekoľko najdôležitejších funkcií API, aby bolo zreteľnejšie ako sa pcap používa.

Zariadenie sa otvára pomocou funkcie **`pcap_t *pcap_open_live(const char *device, int snaplen, int promisc, int to_ms, char *errbuf).`** Táto funkcia navracia packet capture descriptor, ktorý slúži na následné odoberanie paketov zo siete. Vstupný parameter *device* je reťazec identifikujúci zariadenie (napr. "*eth0*" alebo "*xl1*"). Často sa stáva, že nepotrebujeme celé pakety, ale iba ich časti - typicky hlavičky. V tom prípade je možné špecifikovať *snaplen*, čo je veľkosť dát odobraných z paketa, s tým, že zvyšok sa zahodí. Parameter *promisc* je prepínač, ktorým sa nastavuje aby sieťovka bežala v promiskuitnom móde. Ak chceme, aby čítanie paketu zo sieťovej karty neprebiehalo okamžite po jeho príchode, ale až po istom čase a umožniť tak nazhromaždenie viacerých paketov, ktoré sa potom načítajú naraz, je potrebné nastaviť parameter *to_ms*.

Ak nemáme záujem získavať všetky pakety, ale len konkrétnu časť z nich, tak môžeme použiť funkcie *pcap_compile* a *pcap_setfilter* a vyfiltrovať len tie pakety, ktoré si želáme. Filter sa zadáva ako parameter *str* vo forme pcap výrazu, ktorý som

popisoval v kapitole 2.1.3. (napr; *port 20*, *host asterix.domain.com* ...) Funkcia **`int pcap_compile(pcap_t *p, struct bpf_program *fp, char *str, int optimize, bpf_u_int32 netmask)`** skompiluje zadaný *str* parameter do bpf programu. Parameter *p* je deskriptor, ktorý sme získali volaním funkcie *pcap_open_live*. Následne môžeme tento bpf program použíť vo funkcii **`int pcap_setfilter(pcap_t *p, struct bpf_program *fp).`** Po jej zavolaní budeme pomocou deskriptoru *p* dostávať len prefiltrovaný dátový tok.

Získavať pakety pomocou pcap-u je možné dvoma spôsobmi. Prvá metóda, ktorú je možno použíť je **`u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h)`**. Táto funkcia ako parameter príjma, už dobre známy, handler *p* a druhým parametrom je ukazateľ na štruktúru, ktorá nesie informácie o odchytenom pakete: čas, kedy bol odchytený, dĺžku celého paketu a dĺžku dostupnej časti. Návratovou hodnotou tejto funkcie je ukazateľ na samotný odchytený paket. Druhý spôsob je zložitejší, ale na druhú stranu užitočnejší a používanejší, použil som ho aj ja v cyclopovi. Jedná sa o funkciu **`int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)`**. Prvým argumentom je handler *p*, *cnt* udáva počet paketov, ktoré keď odchytí, tak skončí. V prípade, že je nastavený na -1, tak beží neustále. Štvrtým parametrom *user* môžeme, ak si to prajeme, predať svoje dáta do callback funkcie. Callback funkcia, ktorej meno zadáme ako tretí parameter *callback* sa vykoná pre každý paket. Píšeme si ju sami a musí byť typu **`void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)`**. Takže musí mať návratovú hodnotu `void` a štyri parametre rovnakých typov uvedených v rovnakom poradí ako funkcia *got_packet*. Prvý parameter *args* je posledným parametrom z funkcie *pcap_loop*. Parameter *header* obsahuje pre daný paket čas jeho odchytenia, celkovú dĺžku paketu a dostupnú časť. Na samotný odchytený paket ukazuje ukazateľ *packet*.

Detailnejšie informácie o programovaní s pcap sa dajú získať z tutoriálu na domovskej stránke ([10]) a manuálovej stránky pcap ([5]).

3.2 Parser

Konfiguračný súbor spolu s prepínačmi programu tvoria užívateľské rozhranie. Preto som parsovaniu konfiguračného súboru prikladal veľkú pozornosť. Pri rozhodovaní o formáte konfiguračného súboru som sa ešte okrem použitého riešenia zamýšľal nad použitím XML formátu. Takéto riešenie som ovšem po chvíľke zavrhol, keďže ručné editovanie XML formátu by bolo prinajmenšom netypické a XML formát nie je v C až tak prirodzeným formátom ako v objektových jazykoch.

Naopak parsovanie konfiguráku pomocou yaccu sa mi pozdávalo, keďže výhľadovo poskytuje možnosti načítavania aj zložitejších pravidiel, bez nutnosti zdĺhavej implementácie. Výhodou tohto riešenia je aj to, že v prípade porušenia zadefinovanej syntaxe konfiguračného súboru parser určí riadok s chybou, tak ako sme na to zvyknutý z iných kompilátorov.

3.2.1 Yacc

Yacc načíta špecifikáciu gramatiky a vygeneruje pre ňu LR parser (teda parser pre bezkontextové gramatiky). Parser sa skladá z parsovacích tabuliek a rutín napísaných v programovacom jazyku C. Generuje sa do *y.tab.c*. Cyclopovským vstupným súborom pre generovanie parsera je *parser.y*. Takže v prípade požadovaných zmien, stačí zeditovať tento súbor. V nasledujúcich riadkoch popíšem ako tento súbor obecne vyzerá.

```
... definitions ...
%%
... rules ...
%%
... subroutines ...
```

V sekci *definitions* sa nachádzajú definície tokenov a programový C kód uzávorkovaný v "%{" a "%}". Tokeny, ktoré sa definujú v tomto mieste sa vyexportujú do generovaného súboru *y.tab.h*, odkiaľ ich potom používa aj flex.

```
%token SILENT BASE FILTERS COMMON
%token ACTION DURATION FILTER ID NAME SNAPLEN THRESHOLD
%token PCAP PCRE REGEX SEVERITY SCAN_FIELD SNAPLEN
%token EQUAL LBRACE DBL_QUOTE RBRACE SEMICOLON
%token CHAR_STRING FLOAT INTEGER STRING
```

BNF gramatika sa nachádza v sekci *rules.* Ako je vidno v podstate ide o to, že v momente, ked parse narazí na niektorú z premenných, tak naplní príslušnú dátovu štruktúru, buď *struct config_data* alebo *struct filters*.

```
assignments:
      | assignments assignment
      ;
assignment:
      snaplen_a | id_a | name_a | pcap_a | action_a | threshold_a |
duration_a | regex_a
      ;
duration_a:
      DURATION EQUAL INTEGER SEMICOLON
      {
            conf_filter->duration = $3;
      }
      ;
threshold_a:
      THRESHOLD EQUAL INTEGER SEMICOLON
      {
            if(conf_filter->threshold)
              config_error(WARN_PARSE_ERROR,   "ERROR:      Multiple
              thresholds given in %s, line %d", ffname, f_lnum);
            CALLOCT(conf_filter->scan_state, struct scan_state, 1);
            conf_filter->threshold = $3;
      }
      ;
```

Uživateľské podprogramy sú v sekcii *subroutines.*

3.2.2 Flex

Flex je ďalšia implementácia známeho GNU nástroja lex, čo je lexikálny analyzátor. Je to nástroj na generovanie programov, ktoré vykonávajú vyhľadávanie podreťazcov v texte. Tento vygenerovaný súbor (*lex.parser.c*) používa yacc pri parsovaní. Súbor *parser.l* obsahuje reťazce, na ktoré môže parser naraziť a vráti preddefinovanú konštantu (z *parser.tab.h*). Najednoduchšie tokeny sa v *parser.l* zadefinujú takto;

```
<INITIAL>base           return BASE;
<INITIAL>pcap           return PCAP;
<INITIAL>action         return ACTION;
<INITIAL>silent         return SILENT;
```

Keď sa bude rozširovať konfigurák o ďalšie parametre, tak z hľadiska flexu je *parser.l* jediné miesto, čo bude treba prepisovať.

3.2.3 Generovanie parsera

Vygenerovaný parser je dodaný spolu s ostatnými zdrojákmi a nie je ho potreba pregenerovať. Pregenerovanie parsera dostane význam až keď sa zmení jeden zo vstupných súborov parsera, teda v prípade cyclopa ide o súbory *parser.l* alebo *parser.y*. Pre úspešne pregenerovanie je potrebné mať nainštalované programy *yacc* a *flex*. Oba nástroje sú väčšinou súčasťou linuxových distribúcií. Ak sú obe nainštalované, tak potom stačí z príkazovej riadky v adresári zdrojových súborov zadať *make gen_parser*. Výsledným produktom tohto targetu sú zdrojové súbory parsera *y.tab.c*, *l.parser.c* a *y.tab.h*.

3.3 Callback funkcie

Každý filter musí mať nastavenú jednu z akcií SCAN, LOG, NULL, IGNORE alebo SEARCH. Každej z týchto akcií prislúcha callback funkcia, ktorej prototyp musí vyzerať takto: `int callback_function( const struct packet *p, const struct f_rule *f, const struct pcap_pkthdr *h).` Vstupnými parametrami sú teda ukazateľ na paket, ukazateľ na štruktúru reprezentujúcu filter z konfiguračného súboru a pcap hlavička paketu. Implementácie jednotlivých callback funkcií je možné nájsť v súbore *cyclop.c*. Ďalej je v tomto súbore umiestnené pole *callback_list*, v ktorom je zadefinované mapovanie medzi akciami z konfiguráku na príslušnú callback akciu. A pre úplnosť uvediem, že vyhľadanie a nastavenie callback funkcie na filter v závisltosti na akcii vykonáva funkcia *get_callback*, ktorá je volaná z parsera v momente, keď narazí na nastavenie akcie vo filtry. Takže na pridanie novej akcie bude stačiť pridať záznam do mapovania a samozrejme naimplementovať príslušnú callback funkciu.

3.4 Chybové hlášky

Na prácu s chybovými hláškami som si vytvoril jednotný postup, ktorý využívam v celom programe. Priblížim v nasledujúcej časti štruktúry a API na prácu s chybovými hláškami.

Výčtový typ *err_num* definuje číselné kódy chýb.

```
enum err_num {
  ERR_OK = 0,
  ERR_DEBUG = 500,
  DBG_COMMON,
  ...
  ...
  ERR_ARCHITECTURE,
  ERR_NOCONFFILE
};
```

Výčtový typ *err_type* obsahuje kódy formátov chybových hlášok. Formát chybovej hlášky definuje typy premenných a ich poradie vo formátovacom reťazci chybovej hlášky. Uvediem zopár príkladov:
Chybová hláška `"Architecture error %s\n"` je typu `ERRT_S.`
Chybová hláška `"Unable to open conf file \"%s\". %s\n"` je typu `ERRT_SS.`

```
enum err_type {
   ERRT_VOID,         //
   ERRT_S,            //
   ERRT_FI,           //
   ERRT_FSI,          //
   ERRT_SI,           //
   ERRT_SIIS,         //
   ERRT_SS,           //
   ERRT_SSS,          //
   ERRT_CI            //
};
```

Štruktúra errs je základná štruktúra na popis chýb. Skladá sa z čísla chyby, typu chyby a chybovej hlášky.

```
struct errs {
    enum err_num    errn;     /* Cislo chyby */
    enum err_type   errt;     /* Typ chybovej hlasky */
    const char     *msg;      /* Chybova hlaska */
};
```

```
static struct errs errs[] = {
        { ERR_OK, ERRT_VOID,  "Everything is OK\n" },
        { DBG_COMMON,  ERRT_S, "Debug message: %s.\n"},
        { WARN_NOTNUMBER, ERRT_S, "Parsed text \"%s\" contains non-digit chars.\n"},
        . . .
        . . .
        { ERR_ARCHITECTURE, ERRT_S, "Architecture error %s\n"},
        { ERR_NOCONFFILE, ERRT_SS, "Unable to open conf file \"%s\". %s\n"}
}
```

Metóda error() slúži na logovanie chýb.

V kóde potom zahlásenie chybovej hlášky vyzerá napríklad takto:

```
/* log.h* /
void error(enum err_num err, ...);          //deklaracia

/* príklady volania  */
. . .
error(ERR_PCAPERR, "pcap_lookupdev()", errbuf);
. . .
error(ERR_FILTER_VERIFY, "ERROR:  Unknown callback invoked in filter %s", f->id);
...
```

Kapitola 4 – Záver

Program, ktorý som vytvoril samozrejme nemôže funkčne konkurovať existujúcim nástrojom, kde sa na vývoji podieľali väčšie tímy zložené zo sieťových expertov a k dispozícii mali podstatne viac času, ale aj tak si cením toho, čo som dosiahol - teda vytvoril použiteľný nástroj na analýzu dátového toku. V neposlednom rade si cením, že som osobne viac vnikol do oblasti bezpečnosti sietí a programovania sieťí.

4.1 Výsledok práce

V tejto práci som v krátkosti priblížil otázku bezpečnosti sietí, ukázal možné riziká a poskytol jednu z možností ochrany - podarilo sa mi vytvoriť malé IDS. Neovplýva možno mnohými funkčnostami, ale skôr si cením, že vznikla akási platformu pre ďalší vývoj IDS. Pridanie ďalších typov monitorovaní je už totiž jednoduché.

4.2 Možnosti ďalšieho vývoja

Program je ľahko rozšíriteľný o ďalšie funkčnosti. Po napísaní príslušnej callback funkcie je potrebné pridať len jeden riadok do mapovacej tabuľky a program bude o funkcionalitu bohatší. Parser je napísaný spôsobom, že bude veľmi jednoduché rozšíriť syntax konfiguračného jazyka, ak to bude potrebné.

Literatúra

[1] Bradley Tony: Introduction to Intrusion Detection Systems (IDS), *http://netsecurity.about.com/cs/hackertools/a/aa030504.htm*

[2] Information Sciences Institute University of Southern California: INTERNET PROTOCOL, *http://www.ietf.org/rfc/rfc0791.txt*

[3] Information Sciences Institute University of Southern California: TRANSMISSION CONTROL PROTOCOL, *http://www.faqs.org/rfcs/rfc793.html*

[4] Jacobson V., Leres C., McCanne S.: tcpdump man page, *http://www.die.net/doc/linux/man/man8/tcpdump.8.html*

[5] Jacobson V., Leres C., McCanne S.: pcap man page, *http://www.tcpdump.org/pcap3_man.html*

[6] Niemann T.: A Compact Guide to Lex & Yacc, *http://epaperpress.com/lexandyacc/index.html*

[7] Fyodor: Nmap Reference Guide (Man Page) , *http://insecure.org/nmap/man/*

[8] Bro Intrusion Detection System, *http://bro-ids.org*

[9] Snort - the de facto standard for intrusion detection/prevention, *http://www.snort.org/*

[10] Programming with pcap, *http://www.tcpdump.org/pcap.htm*

[11] Wikipedia. Worl Wide Web, *http://www.wikipedia.org/*