



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

OBFUSKACE SÍŤOVÉHO PROVOZU PRO ZABRÁNĚNÍ JEHO DETEKCE POMOCÍ IDS

NETWORK TRAFFIC OBFUSCATION FOR IDS DETECTION AVOIDANCE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. DANIEL OVŠONKA

VEDOUCÍ PRÁCE

SUPERVISOR

Mgr. KAMIL MALINKA, Ph.D.

BRNO 2013

Abstrakt

Tato práce se zabývá principy obfuskace síťového provozu tak, aby nedošlo k jeho detekci pomocí Intrusion Detection Systému (IDS) instalovaného v síti. V úvodu práce bude čtenář obeznámen s principem fungování základních typů IDS a uveden do problematiky obfuskáčnických technik, které budou sloužit jako odrazový můstek při tvorbě vlastní knihovny, které návrh je popsán v závěrečné části práce. Výsledek práce představuje knihovna, která poskytuje všechny implementované techniky na další využití. Knihovna může být tedy využita při penetračním testování nových systémů, případně použita samotným útočníkem.

Abstract

This thesis deals with the principles of network traffic obfuscation, in order to avoid its detection by the Intrusion Detection System installed in the network. At the beginning of the work, reader is familiarized with the fundamental principle of the basic types of IDS and introduced into the matter of obfuscation techniques, that serve as stepping stone in order to create our own library, whose design is described in the last part of the work. The outcome of the work is represented by a library, that provides all the implemented techniques for further use. The library can be well utilized in penetration testing of the new systems or used by the attacker.

Klíčová slova

Obfuskace, Intrusion Detection Systém, IDS, Bezpečnost, Maskování protokolu, Detekce protokolu, Síťové útoky

Keywords

Obfuscation, Intrusion Detection System, IDS, Security, Protocol masking, Protocol detection, Network attacks

Citace

Daniel Ovšonka: Obfuskace síťového provozu pro zabránění jeho detekce pomocí IDS, diplomová práce, Brno, FIT VUT v Brně, 2013

Obfuskace síťového provozu pro zabránění jeho detekce pomocí IDS

Prohlášení

Tímto prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Mgr. Kamila Malinku, Ph.D.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Daniel Ovšonka

21. května 2013

Poděkování

Chtěl bych poděkovat vedoucímu práce panu Mgr. Kamilovi Malinkovi za odborní pomoc a rady při řešení této diplomové práce.

© Daniel Ovšonka, 2013.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

| | |
|--|-----------|
| Úvod | 2 |
| 1 Intrusion Detection System | 3 |
| 1.1 Obecný popis IDS | 3 |
| 1.2 Rozdelenie IDS | 5 |
| 1.3 Network-Based IDS | 9 |
| 1.4 Implementácia <i>Snort</i> | 11 |
| 1.5 Zhrnutie | 12 |
| 2 Princípy a klasifikácia obfuskácie | 13 |
| 2.1 Obfuskácia sieťovej komunikácie | 13 |
| 2.2 Obfuskácia konkrétnych útokov | 17 |
| 2.3 Zhrnutie | 20 |
| 3 Návrh a implementácia knižnice | 21 |
| 3.1 Architektúra cieľovej siete | 21 |
| 3.2 Princíp fungovania a použitia knižnice | 23 |
| 3.3 Štruktúra knižnice | 25 |
| 3.4 Štruktúra proxy modulu | 30 |
| 3.5 Nástroje použité pri implementácii | 34 |
| 3.6 Odchytávanie paketov dátových tokov | 36 |
| 3.7 Realizácia obfuskovanej komunikácie | 37 |
| 3.8 Konfigurácia a autentizácia modulov | 40 |
| 4 Experimenty s knižnicou a dosiahnuté výsledky | 42 |
| 4.1 Testovacia virtuálna sieť | 42 |
| 4.2 Experimenty s legitímnou komunikáciou | 44 |
| 4.3 Experimenty s exploitačnými dátami | 46 |
| 4.4 AIPS systém | 50 |
| 4.5 Zhrnutie získaných výsledkov | 52 |
| Záver | 53 |
| A Príklady konfiguračných súborov | 58 |
| B Obsah CD | 60 |

Úvod

Úlohou tohto textu je priblížiť čitateľovi možnosti obfuskácie sieťovej komunikácie s cieľom vyhnúť sa detekcii pomocou *Intrusion Detection Systému*. Obfuskácia sieťových útokov na zraniteľné vzdialené služby a sieťovej komunikácie je pomerne mladým odvetvím spadajúcim do počítačovej bezpečnosti. Skúmanie týchto princípov je osožné hlavne pri návrhu nových prístupov detekcie nelegálnej komunikácie v sieti. Jedná sa o pomerne rýchlo sa rozvíjajúcu oblasť, keď zlepšovanie pomocou nových prístupov k obfuskácii vedie, na druhej strane, k novým prístupom analýzy a detekcie kompromitujúcich dát v sieti. Zvýšenie schopnosti detekcie potenciálne nebezpečných akcií v sieťovej komunikácii pomocou IDS, by malo zabezpečiť možnosť odhaliť aj dosiaľ neznáme, prípadne *zero-day* útoky, čím sa zvýši odolnosť daného systému.

Cieľom práce je vytvorenie *knižnice*, ktorá umožní komunikáciu so vzdialenou chránenou sieťou tak, aby prenášané dáta neboli rozpoznateľné pomocou *Intrusion Detection Systému*. Nasledujúci text sa zameriava na ozrejenie základných teoretických znalostí, ktoré sú následne rozvinuté do popisu vlastného návrhu a implementácie obfuskacej knižnice. Prvá kapitola je venovaná *Intrusion Detection Systémom*, ktorých princípy a spôsoby riešenia detekcie boli východiskovými znalosťami, pri návrhu spôsobu vlastnej obfuskácie. Kapitola sa okrem základného uvedenia do problematiky a klasifikácie rôznych systémov, venuje aj popisu metód, ktoré sú najčastejšie využívané na detekciu. Záverečná časť tejto kapitoly (1.4) sa podrobne venuje konkrétnej implementácii *Snort*, ktorá je pre túto prácu referenčná.

Druhá kapitola je zameraná na vysvetlenie významných pojmov a metód obfuskácie, pričom rozdeľuje prístupy na základe spracovávaných dát na dve triedy, konkrétne obfuskáciu sieťovej komunikácie a obfuskáciu útokov. Vedomosti získané z tejto kapitoly ďalej slúžia ako odrazový mostík k návrhu a implementácii vlastnej knižnice. Ako najvýznamnejšie prístupy sa v kontexte tejto práce javia metódy založené na *maskovaní protokolu* v sieti (2.1.3) a riešenie postavené na *šifrovaní* určitých dát (2.1.1).

Jadro práce predstavuje tretia kapitola, orientujúca sa na samotnú realizáciu a štruktúru obfuskacej knižnice. V tejto časti je predstavený základný koncept princípov knižnice, rozvinutý do predstavenia formálneho návrhu a popisu možného použitia. Záverečná časť kapitoly je venovaná detailom implementácie, kde sú uvedené jednotlivé etapy vývoja aplikácie až po získanie výsledného riešenia.

Štvrtá kapitola práce slúži na ukážku testovania vytvorenej aplikácie, pričom sa zameriava na popis testovacieho prostredia, metodiky a vlastných výsledkov testovania. Experimenty boli rozdelené na dve časti, keď v prvom prípade boli vykonávané s *legitímnou komunikáciou* (4.2), kým druhý prípad testoval obfuskáciu *exploitačných dát* (4.3). Záver kapitoly je venovaný analýze získaných výsledkov a návrhu ďalších možných experimentov.

Posledná kapitola práce sa stručne venuje zhrnutiu poznatkov a výsledkov získaných pri riešení uvedenej problematiky a uvádza ďalšie možnosti rozšírenia knižnice v budúcnosti. Pri tvorbe boli použité rôzne zdroje a publikácie, ktoré sú uvedené a citované v zodpovedajúcich kapitolách.

Kapitola 1

Intrusion Detection System

Intrusion Detection Systémy (ďalej len IDS) patria v súčasnosti k pomerne rozvíjajúcim sa odvetviám počítačovej bezpečnosti sietí. Základný princíp činnosti IDS sa najčastejšie prirovnáva k alarmu, teda, ak útočník nejakou spôsobom naruší náš obranný periméter, IDS zašle alarm systémovému administrátorovi, ktorý tak môže promptne vykonať odpovedajúce kroky. Vykonávanie takéhoto monitorovania by pri bežnom využití siete systémom nebolo v rozumnej miere možné manuálne.

Z popisu základného princípu fungovania je zrejmé, že útočníci sa budú snažiť vyhnúť sa novej detekcii. Týmto sa dostávame do kolobehu zlepšovania schopností IDS a na druhej strane taktiež k vzniku nových techník vyhýbania sa detekcii. IDS nemôže v súčasnosti poskytovať kompletnú ochranu, ale slúži na doplnenie prvkov v užívateľskej časti monitorovaného systému, ako sú firewally a antivírusové programy.

Jedným z kľúčových problémov týchto systémov je, že útočníci sa často dokážu prispôbiť bezpečnostným opatreniam, ktoré pomocou IDS zavedieme. Z tohto dôvodu je nedostatočné vytvorenie IDS, ktoré je schopné odhaľovať len útoky známe v čase vytvárania systému, ale je nutné, aby bol systém schopný zaznamenať aj novo vytvorené hrozby.

IDS je vzhľadom na svoju štruktúru a schopnosť kontrolovať sieťové dátové toky, používaný aj na detekciu využívania určitých protokolov. V mnohých sieťach je používanie niektorých protokolov zakázané a IDS môže umožniť administrátorovi rýchlejšie odhalenie nebezpečného účastníka a jeho zablokovanie. Medzi nežiadúce protokoly mnohokrát patria protokoly rôznych Peer-to-Peer sietí, prípadne v krajinách s rozvinutou cenzúrou sa sem radí napríklad aj komunikácia prostredníctvom *Tor*¹ siete.

Počas svojho vývoja sa IDS pretransformovali z jednoduchého, monolitického dávkovo orientovaného systému, na systém schopný pracovať v reálnom čase, distribuovaný do viacerých sieťových komponent. V nasledujúcich podkapitolách si podrobnejšie popíšeme obecnú schému IDS a jednotlivé typy systémov používaných v súčasnosti. Informácie pre túto kapitolu boli čerpané najmä z publikácie [6] a práce [28].

1.1 Obecný popis IDS

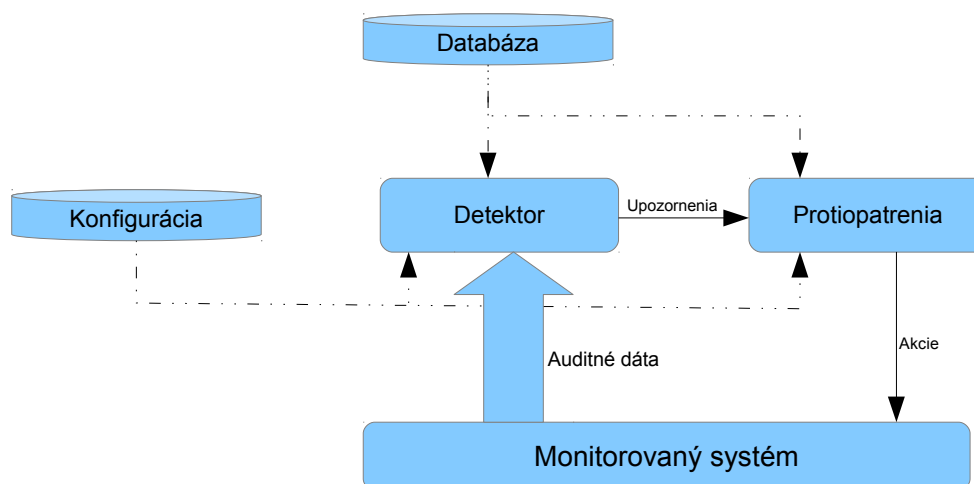
Úlohou IDS je dynamicky monitorovať udalosti a akcie vykonávané v skúmanom prostredí a po ich preskúmaní určiť, či sú legitímne alebo sa jedná o *útoky na dané prostredie*. Pri vyššej forme abstrakcie môžeme IDS označiť ako detektor spracujúci informácie získané zo skúmaného systému.

¹<https://www.torproject.org/>

Obecne používa IDS tri typy informácií na svoje vyhodnocovanie, a to konkrétne:

- Informácie získané dlhodobým *skúmaním známych útokov* na skúmaný systém, z ktorých je vytvorená databáza. Údaje z tejto databázy môže potom IDS použiť na porovnanie s údajmi, ktoré získal pri samotnom behu (*Signature-based IDS*).
- *Konfiguračné informácie* o skúmanom systéme a o jeho aktuálnom stave. IDS v tomto prípade analyzuje, či sa systém po vykonaní akcie nedostane do stavu, ktorý môže byť spôsobený útokom. Tieto princípy využívajú napríklad systémy založené na *skúmaní systémových volaní*.
- *Udalosti*, ktoré sa odohrávajú priamo v skúmanom systéme. Pod týmto si najčastejšie predstavujeme samotnú sieťovú komunikáciu. Úlohou IDS je v tomto prípade vyextrahovať z množstva dát tie podstatné a vykonávať vyhodnocovanie najmä podľa nich. Na tomto princípe sú postavené napríklad systémy založené na *vzoroch chovania*.

Z tohto popisu sme schopní vytvoriť schému obecného systému na obrázku 1.1, na ktorej môžeme vidieť spomenuté základné prvky a výmenu informácií medzi nimi.



Obrázok 1.1: Obecná schéma IDS [6]

Pre obecný IDS môžeme taktiež sformulovať základné požiadavky, ktoré je pri samotnej implementácii a nasadení v reálnom prostredí nutné rešpektovať. Hlavné požiadavky [6] sú:

- *Presnosť* - Pod presnosťou rozumieme schopnosť odlíšiť *anomálie* a reálne pokusy o *prienik do monitorovaného systému*, od *legálnych operácií*, vykonávaných legitímnymi užívateľmi. Za nepresnosť teda považujeme stav, kedy IDS označí korektnú operáciu v systéme ako anomáliu, a tým sa nám zvýši počet záznamov, ktoré je nutné vo väčšine prípadov kontrolovať ručne administrátorom.
- *Výkonnosť* - U výkonnosti systému skúmame, v akej miere je IDS schopné vyhodnotiť dáta získané od monitorovaného systému. Rozhodujúcim faktorom v tomto prípade

je schopnosť vykonávať analýzu týchto dát v *reálnom čase*, ktorá je v súčasnosti pomerne bežná, avšak v minulosti to nebolo pravidlom.

- *Úplnosť* - Jedná sa o najťažšie skúmanú vlastnosť, pretože určuje mieru neschopnosti odhaliť útok daným IDS. Je pomerne ťažké vyčísliť túto metriku, pretože nepoznáme kompletnú množinu možných útokov na daný skúmaný systém.

Okrem týchto kľúčových vlastností každého IDS by mal byť schopný systém spĺňať aj doplnkové požiadavky, ktoré taktiež ovplyvňujú jeho kvalitu. Jedná sa konkrétne o tieto vlastnosti [6]:

- *Odolnosť voči chybám* - Udáva schopnosť bezporuchovej prevádzky systému. Do tejto skupiny spadá aj miera odolnosti voči útokom miereným priamo na IDS (napr. *DoS* a podobne). Odolnosť je pomerne dôležitá, pretože IDS musia taktiež bežať v prostredí bežného operačného systému, ktorý býva často sám zraniteľný.
- *Rýchlosť reakcie* - IDS musí byť schopné promptne reagovať na potenciálne odhalené útoky. Nejedná sa iba o rýchlosť vyhodnotenia a teda samotnú výkonnosť systému, ale zahŕňame sem aj schopnosť propagácie vygenerovaného varovania zodpovedajúcemu administrátorovi. Rýchlosť reakcie je pomerne dôležitý ukazovateľ, pretože čím skôr je potenciálny útok odhalený, tým sa znižuje riziko škody vykonanej útočníkom.

V tejto kapitole sme uviedli obecný popis a princíp IDS doplnený o popis základných skúmaných vlastností týchto systémov. V nasledujúcej kapitole bude stručne uvedená ich formálna klasifikácia a možné delenie na základe rôznych kritérií.

1.2 Rozdelenie IDS

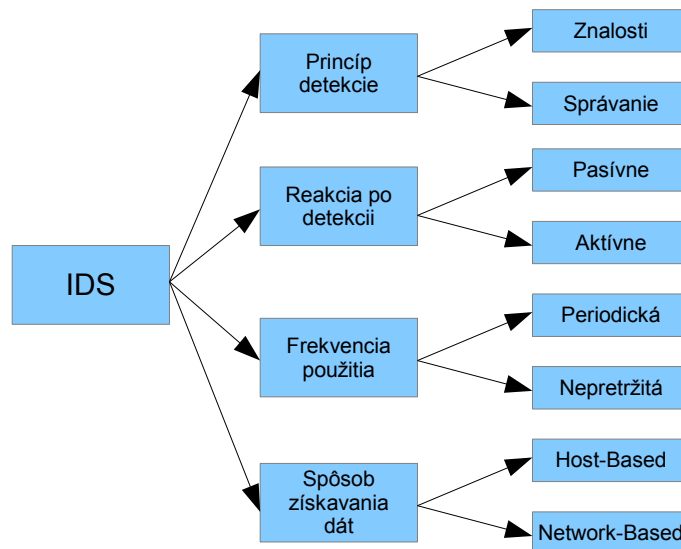
IDS môžeme rozdeľovať na základe rôznych kritérií, vychádzajúcich z líšiacich sa princípov práce jednotlivých typov. Súhrnné rozdelenie je zobrazené na obrázku číslo 1.2. V tejto kapitole si tieto základné spôsoby informačne popíšeme, aby sme mohli v ďalšej práci podrobnejšie rozvinúť problematiku IDS systémov a princípov obfuskácie s cieľom vyhnutia sa detekcii. Informácie v nasledujúcich podkapitolách boli čerpané z publikácií [6] a [28].

1.2.1 Delenie na základe princípu detekcie

Základná a asi najpoužívanejšia klasifikácia IDS je založená na *princípe detekcie potenciálnych útokov*. IDS podľa týchto kritérií delíme obyčajne na systémy založené na *znalostiach o útokoch* (zvyčajne uložené vzory v databáze) a na systémy založené na *skúmaní správania systému* (napr. kontrola systémových volaní), ktoré sa líšia od uloženého modelu štandardného správania chráneného systému.

Systémy založené na znalostiach sa v literatúre nazývajú aj „*misuse detection*“ prípadne „*detection on appearance*“. Ich činnosť je postavená na databáze, v ktorej sú nazhromaždené informácie o známych útokoch a zraniteľnostiach ochraňovaného systému. Systém pri svojej činnosti skúma, či nedochádza k pokusom zneužitia niektorého z týchto slabých miest. V prípade, že dôjde k podozreniu, že k tomu prišlo, dochádza k vyvolaniu upozornenia pre administrátora.

Nevýhodou tejto metódy detekcie je, že útoky, ktoré sa nenachádzajú v databáze IDS, nebudú rozpoznané a IDS ich bude považovať za legítimnú operáciu so systémom. Odhliadnuc od toho, je presnosť tohto princípu považovaná za pomerne dobrú v prípade,



Obrázok 1.2: Delenie IDS [6]

že dochádza k pravidelnej aktualizácii databázy. Aktualizácia databázy je na druhej strane taktiež pomerne problematická, pretože je potrebné dôkladné preskúmanie možných útokov a následné vytvorenie špecifických pravidiel, ktoré uložíme do databázy. V prípade nových útokov musí teda najskôr prísť k ich odhaleniu, čo môže nejakú dobu trvať. Za ďalšiu nevýhodu môžeme považovať striktnú orientáciu IDS na daný systém, pretože pravidlá vytvárané na jednotlivé zraniteľnosti sú striktné späté s platformou, operačným systémom, bežiacimi službami a ďalšou softwarovou výbavou skúmaného systému. Z týchto vlastností vyplýva, že sa jedná o princíp pomerne otvorený možnosti obfuskácie.

Na druhej strane toto riešenie ponúka aj výhodu. Pomerne podstatnou vlastnosťou je už spomenutá presnosť riešenia, čo ako už bolo spomenuté, znamená nízky počet vygenerovaných *falošných varovaní*. Toto uľahčuje prácu systémového administrátora, pretože nemusí kontrolovať toľko záznamov. Táto vlastnosť je podpísaná pod pomernú rozšírenosť tohto princípu medzi rôznymi implementáciami IDS.

Za druhý typ v tejto klasifikácii považujeme systémy založené na skúmaní *správania systémov* a jeho odchýľok od štandardného modelu, a teda očakávaného správania chráneného systému. Princíp často nazývaný aj „*anomaly detection*“. Princíp činnosti spočíva v tom, že systém sa musí pred nasadením nejakým vhodným spôsobom naučiť vyextrahované správanie skúmaného systému. IDS potom môže porovnávať tento referenčný model systému so súčasným stavom a v prípade, že zaznamená odchýľku, dôjde k vygenerovaniu varovania o možnom nelegitímnom konaní v systéme. Súhrnne by sa dalo povedať, že akékoľvek akcie, ktoré sa odlišujú od pôvodného naučeného chovania, sú považované za nelegálnu akciu.

Hlavnou výhodou tohto riešenia je, že IDS je často schopné odhaliť aj nové útoky, ktoré

by v prípade predošlého spôsobu detekcie ostali neodhalené. Podobne, IDS založené na správaní dokážu zachytiť aj pokusy o exploítácie na zatiaľ neobjavené zraniteľné miesta. Za výhodu môžeme považovať aj schopnosť odhaliť nelegálne akcie vo vnútornej sieti od legitímnych užívateľov, ktoré zvyčajne neútočia priamo na zraniteľné miesta samotného systému. Tento prístup je oproti predošlému menej prístupný možnostiam obfuskácie.

Ako najvýznamnejšiu nevýhodu tohto riešenia detekcie IDS sa obecne považuje vysoký počet *falošných alarmov*, pretože pri extrakcii a učení sa správania systému, nie sme schopní obsiahnuť kompletnú množinu možných stavov, a tak môže systém vygenerovať varovanie aj pre korektné akcie. Ako riešenie tohto prípadu sa používa postupné dynamické učenie sa systému, keď sú nové stavy pridávané za behu už po nasadení. Takto môžeme postupne vyladiť IDS na konkrétny systém a dosahovať pomerne vysokú presnosť a kompletnosť detekcie. Pri tomto riešení si avšak musíme dať pozor, aby sme do IDS nezanesli aj správanie sa skutočných neodhalených útokov, pretože tie by potom boli považované za legálne a neboli by v ďalšom behu vôbec odhalené.

V tejto podkapitole sme popísali základné princípy dvoch základných prístupov detekcie pokusov o preniknutie do systému pomocou IDS. Ako môžeme z popisu vidieť, jedná sa o pomerne odlišné princípy aj čo sa týka výsledkov. Konkrétne markantné rozdiely môžu nastať v počte chybných hlásení (nižší u systémov založených na znalostiach) a na druhej strane v schopnosti odhaliť nové útoky (lepšia schopnosť u systémov skúmajúcich správanie).

1.2.2 Delenie na základe reakcie po detekcii

IDS podľa odozvy na odhalenú nelegitímnu operáciu v systéme môžeme obecne rozdeliť na *aktívne* a *pasívne*. V súčasnej dobe existujú prevažne pasívne riešenia, ktoré, ako bolo spomenuté, po odhalení útoku vygenerujú iba chybové hlásenie do logovacieho súboru. Na druhej strane aktívne IDS sú schopné vykonať jednoduché *protiopatrenia*, ako napríklad ukončenie sieťového spojenia a podobne. Nevýhodou aktívneho riešenia je, že v prípade väčšieho množstva chybných detekcií dôjde k zníženej dostupnosti daného systému. U aktívnych IDS taktiež môže vzniknúť polemika o tom, či by nemali byť zaradené priamo do skupiny *Intrusion Prevention Systémov*, ktoré však vykonávajú často komplexnejšie protiopatrenia ako aktívne IDS.

Na rozhraní medzi aktívnymi a pasívnymi technikami môžeme považovať prístup, keď IDS periodicky kontroluje nejaké konfiguračné nastavenia a v prípade, že prišlo k ich zmene, je schopné opraviť túto konfiguráciu. Tento princíp je použiteľný napríklad na kontrolu prístupových práv k súborom a podobne, keď sa snažíme dosiahnuť prístup iba pre určitú skupinu užívateľov, ktorí by mohli tieto práva zmeniť. Návrat do pôvodného stavu je takto o poznanie rýchlejší, ako keby sa vygenerovalo varovanie, ktoré by bolo nutné ručne prekontrolovať. Týmto prístupom je teda pravdepodobnosť potenciálnej škody oveľa nižšia, pretože náprava bude niekoľkonásobne rýchlejšia.

Z pohľadu obfuskácie sú oba varianty ekvivalentné, pretože aktívny resp. pasívny mód nemá vplyv na schopnosť rozpoznávania systému. Nevýhodou u aktívneho systému avšak môže byť zablokovanie obfuskovaného toku v prípade detekcie.

Klasifikácia na základe tejto metriky je v súčasnosti ešte pomerne exotická, pretože existuje veľmi málo reálnych implementácií, ktoré by používali aktívny princíp reakcie pri detekcii nelegitímneho konania v skúmanom systéme. Časom sa však predpokladá, že dôjde k rozvoju kategórie aktívnych IDS, pretože môžu uľahčiť prácu správcov systémov.

1.2.3 Delenie na základe frekvencie použitia

Ďalšie možné rozdelenie IDS môžeme vytvoriť na základe toho, s akou frekvenciou prebieha samotná analýza získaných dát. Konkrétne poznáme *systémy s periodickou analýzou* a *systémy s nepretržitou analýzou*. Jednotlivé riešenia si stručne popíšeme.

IDS založené na periodickej analýze vykonávajú kontrolu iba v určitých *časových intervaloch*, keď dôjde k nasnímaniu súčasného stavu chráneného systému a na kontrolu sa použije už iba tento snímok systému. Takýto typ analýzy sa používa typicky na občasnú kontrolu systému, kedy sa skúma napríklad, či má chránený systém legálnu konfiguráciu, či sú nainštalované potrebné aktualizácie a podobne. Nevýhodou je, že komponenty chráneného systému môžu mať rôznu softwarovú výbavu, čo znamená, že kontroly by museli byť vytvorené konkrétne pre jednotlivé elementy systému. Za ďalšiu nevýhodu môžeme považovať to, že nelegálne udalosti vykonané medzi jednotlivými nasnímaniami nemôžu byť odhalené. Toto riešenie poskytuje pomerne slabé zabezpečenie a najčastejšie sa používa iba ako doplnková služba ochrany.

Na druhej strane systémy s nepretržitou analýzou, nazývané aj ako *dynamické systémy*, pracujú v reálnom čase a monitorujú všetky udalosti v okamžiku, keď sa stanú. Najčastejším využitím dynamických systémov je ochrana nejakej vnútornej siete, kde sú zdrojom dát samotné pakety dátových tokov. Takáto analýza je pomerne náročná na výkon IDS, pretože pri súčasných rýchlostiach dátových sietí musí byť systém schopný spracovať častokrát veľké množstvo auditných dát v reálnom čase.

Obfuskácia s cieľom vyhnúť sa detekcii je samozrejme podľa popisu jednoduchšia u systémov, ktoré vykonávajú iba periodickú analýzu, pretože útok nemusí byť vo väčšine prípadov vôbec odhalený. Na druhej strane najpoužívanejším riešením v reálnych implementáciách je použitie dynamického prístupu a analýzy všetkých udalostí v reálnom čase. Z tohto dôvodu bude nutné, aby sa naša knižnica zameriavala na vyhnutie sa detekcii dynamickým systémom.

1.2.4 Delenie na základe spôsobu získavania auditných dát

Tento spôsob klasifikácie je pravdepodobne najpoužívanejší. Rozdeľuje systémy na tzv. „*Host-Based IDS*“, ktoré získavajú dáta priamo od chráneného systému, ktorý spravujú a na ktorom súčasne aj bežia. Druhú skupinu tvoria tzv. „*Network-Based IDS*“, ktoré ako auditné dáta používajú priamo sieťové dáta získané z rozhrania skúmaného systému.

Host-Based IDS považujeme z hľadiska historického vývoja za prvý spôsob implementácie, ktorý bol používaný. Tento princíp vznikol hlavne preto, že v minulosti neboli siete a internet príliš rozšírené a väčšina práce so strojom prebiehala na lokálnej úrovni. IDS v tomto prípade skúma napr. dáta z logovacích súborov, systémové volania operačného systému a podobne. Veľmi dôležitou požiadavkou je, aby bol systém schopný spracovávať dáta v reálnom čase, pretože v opačnom prípade môže dôjsť k úspešnému útoku a zahladeniu stôp útočníkom. Útočník by taktiež v tomto prípade po úspešnom útoku mohol odstaviť samotné IDS.

Host-Based IDS môžeme opäť rozdeliť podľa spôsobu detekcie na systémy založené na znalostiach a systémy založené na správaní, ktoré boli popísané v kapitole 1.2.1. V prvom prípade implementácie je obfuskácia pomerne jednoduchá záležitosť, keď stačí útok modifikovať tak, aby sme sa vyhli porovnaniu s bežným vzorom útoku. Na túto úlohu môžeme použiť rôzne techniky, ako napríklad vkladanie NOP („*no-operation*“) operácii, používanie skokov a podobných operácií, ktoré nemenia sémantiku pôvodného kódu. Podrobnejšie princípy útokov budú popísané v kapitole 2.

S postupným rozvojom sieťovej komunikácie sa začali vo väčšej miere vyskytovať útoky z vonkajšej siete, respektíve z internetu. Host-Based IDS prestali postupne stačiť na odhaľovanie týchto útokov, pretože útoky (napr. *skenovanie portov*, DNS spoofing a podobne) nebolo prakticky možné z auditných dát zistiť. Z tohto dôvodu vznikali rôzne hybridné riešenia, keď medzi sebou jednotlivé IDS komunikovali a zasielali si auditné dáta. Toto riešenie sa však ukázalo ako nevhodné, hlavne z dôvodu vysokého zaťaženia prenosového pásma siete a taktiež aj nedostačujúcimi výsledkami. Ďalším krokom vývoja reagujúcim na tieto okolnosti boli *Network-Based IDS*.

Network-Based IDS patria v súčasnosti k najpoužívanejšiemu riešeniu. Ako už bolo spomenuté, pre svoju činnosť využívajú sieťové dáta extrahované priamo zo sieťového toku. *Network-Based IDS* sú teda na základe analýzy sieťového toku schopné odhaliť podozrivé príkazy zasielané po sieti, prípadne klasifikovať nebezpečné alebo nelegálne protokoly. Keďže sa jedná o najrozšírenejšie riešenie v moderných IDS a táto práca sa bude zameriavať najmä obfuskáciou sieťovej prevádzky, popíšeme si podrobnejšie princípy v samostatnej kapitole 1.3.

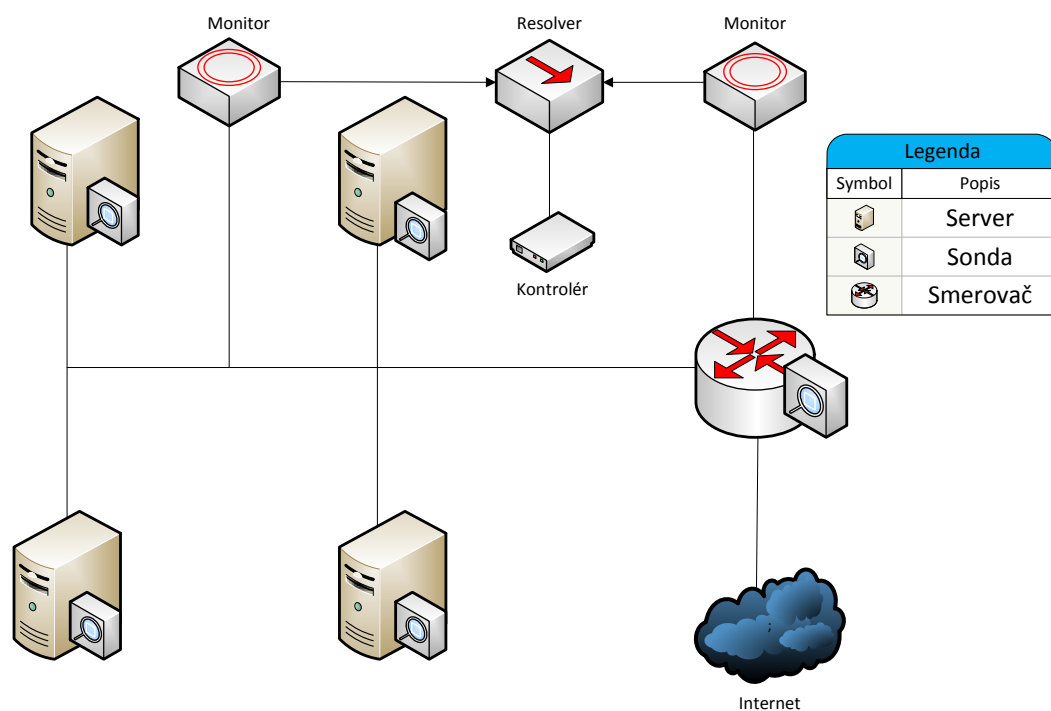
1.3 Network-Based IDS

Vývoj v oblasti IDS prebiehal od monolitických rozsiahlych systémov, bežiacich na jednom hostiteľskom stroji až po súčasné *Network-Based IDS*, ktoré môžeme označiť ako rozsiahle distribuované systémy pracujúce v reálnom čase.

Obecne sa *Network-Based IDS* skladá z týchto základných blokov [28], vypísaných podľa ich hierarchickej úrovne:

- *Sonda (senzor)* - Jedná sa o komponent, ktorý vykonáva jednoduchý zber informácií pre IDS. Implementácia sondy môže byť vytvorená presne na mieru prostrediu, z ktorého bude snímať dáta. Po nasnímaní potrebných dát je možné vykonať transformáciu týchto dát na jednotný formát používaný daným IDS. IDS môže na svoju činnosť samozrejme používať rôzne typy sond, s cieľom získania kompletnejších dát z rozdielnych zdrojov.
- *Monitor* - Predstavuje hlavnú výpočtovú jednotku IDS. Monitor preberá dáta od sond a vykonáva vyhodnocovanie. Zvyčajne realizuje porovnávanie s modelom správania systému, prípadne môže na základe auditných dát tento model aktualizovať. V prípade odhalenia potenciálnej nebezpečnej udalosti, monitor vytvorí varovanie, ktoré najčastejšie zasiela *resolveru*.
- *Resolver* - Preberá jednotlivé varovania z monitorov a na základe typu varovania vykoná odpovedajúcu akciu. Najčastejšou akciou je zápis podozrivej udalosti do logovacieho súboru, avšak existujú ako bolo spomenuté v kapitole 1.2.2 aj aktívne IDS, kde resolver môže napríklad pridať pravidlá do firewallu, zablokovať dátový tok, zmeniť konfiguráciu systému a podobne.
- *Kontrolér* - Považujeme v hierarchii za najvyšší prvok. Jedná sa o riadiaci modul, slúžiaci na synchronizáciu jednotlivých distribuovaných komponentov a ich jednoduchšiu správu. Kontrolér teda nemá vplyv na detekčné schopnosti samotného systému, ale uľahčuje jeho administrátorom napríklad reštartovanie, aktualizáciu, prípadne vloženie ďalších komponent do IDS.

Z obecného popisu je možné vytvoriť obecnú schému rozloženia komponentov v skúmanom systéme, ktorá je zobrazená na obrázku 1.3. Uvedené distribuované riešenie je využívané hlavne v rozsiahlych sieťach, kde sa používajú komerčné IDS implementácie, často upravené na mieru danej siete. V reálnych implementáciách sa mnohokrát niektoré komponenty spájajú do jednej (napr. monitor a kontrolér), prípadne v jednoduchých implementáciách sú všetky komponenty integrované do jedného modulu, ktorý je potom nasadený napríklad na sieťovú bránu, oddeľujúcu vnútornú chránenú sieť od vonkajšej.



Obrázok 1.3: Obecná schéma rozloženia komponent IDS [28]

1.3.1 Princípy získavania auditných dát

Ako bolo spomenuté, IDS systém dokáže pracovať s väčším počtom rôznych sond, ktoré extrahujú dáta zo systému. Najvýznamnejšie princípy získavania dát sondami [28] sú:

- *Monitorovanie hostiteľských zariadení* - sondy získavajú dáta priamo z hostiteľských počítačov obdobne ako Host-Based IDS, keď skúmajú logovacie súbory hostiteľa, systémové volania a podobne.
- *Monitorovanie siete hostiteľských zariadení* - sondy extrahujú dáta priamo zo sieťového rozhrania hostiteľa podobne ako napríklad firewall. Sonda pri tomto prístupe môže skúmať všetky protokolové úrovne v odchytenom pakete. Nevýhodou oboch týchto riešení je, že majú dopad na výkonnosť hostiteľského zariadenia tým, že samotná extrakcia dát prebieha u samotného hostiteľa.

- *Monitorovanie siete v promiskuitnom móde* - pravdepodobne najrozšírenejší prístup u Network-Based IDS, keď sa sonda inštaluje ako samostatné a nezávislé zariadenie do siete. Sonda potom môže odchytať kompletnú komunikáciu medzi pripojenými zariadeniami. V tomto prípade teda môžeme priamo odchytiť komunikáciu medzi útočníkom a napádaným systémom.

Tento prístup nám zabezpečuje mnohé výhody oproti predchádzajúcim dvom riešeniam, ako, napríklad, *nemá dopad na výkonnosť* žiadneho z hostiteľských zariadení v sieti, umožňuje monitorovanie celého sieťového segmentu jednou sondou, schopnosť odhaliť a preskúmať udalosti, ktoré sa odohrajú priamo v sieti a podobne. Nevýhodou tohto riešenia je na druhej strane, že sonda musí byť pomerne výkonná, aby bola schopná odchytať pakety v reálnom čase aj pri plnom využití siete. Ďalšou nevýhodou je použitie *kryptografie*, keď nie je sonda schopná preskúmať obsah získaných dát.

Za špecifický prípad získavania dát slúžiacich na klasifikáciu protokolov, ktoré sú zabezpečené pomocou kryptografie, môžeme považovať *štatistické hodnoty* získané z dátových tokov. Skúmajú sa *veľkosti paketov* a *časové intervaly* medzi nimi. Na základe týchto hodnôt je potom možné vytvoriť štatistický popis jednotlivých protokolov. Z tohto dôvodu pri obfuskácii protokolu nestačí vykonať zašifrovanie obsahu jednotlivých paketov, ale je potrebné vhodným spôsobom upraviť aj tieto štatistické vlastnosti.

1.4 Implementácia *Snort*

V tejto práci sa ako referenčná implementácia IDS používa open-source nástroj *Snort*, ktorý patrí v súčasnosti k najrozšírenejším nástrojom používaným v malých a stredných sieťach. Hlavnou výhodou tohto riešenia je možnosť vlastnej úpravy detekčných pravidiel a ďalšieho rozšírenia pomocou rôznych doplnkov a preprocesorov. Informácie v tejto podsekcii vychádzajú z rovnomennej publikácie [4].

Snort nasadený ako network-based IDS, je v počítačovej bezpečnosti pomerne používaný variant, pretože je považovaný za jednoduchý, malý, ale zároveň efektívny a užitočný nástroj, ktorý je v dnešnej dobe označovaný za štandard v monitorovaní malých sietí. Na rozdiel od rozsiahlych komerčných nástrojov, ktoré sú hardwarovo a finančne náročné, poskytuje *Snort* dostatočnú funkcionálnosť pre malé privátne siete, je nenáročný na výkon hostujúceho stroja a umožňuje jednoduché a efektívne vytváranie pravidiel.

Princíp spracovania prichádzajúcich paketov spočíva v prevedení nasledujúcich krokov:

1. *Spracovanie dekodérom paketov* - Zachytené dáta sú v prvej fáze analyzované dekodérom paketov, ktorý samotné dáta zachytáva priamo z rozhrania sieťovej karty, načítavajúcej v *promiskuitnom režime*. Analyzátor v tomto kroku zisťuje, aké protokoly sú používané vo vyšších vrstvách a na základe týchto hodnôt porovnáva tieto dáta s dostupnými pravidlami, ktoré popisujú správanie komunikácie daným protokolom, získané štatistickou analýzou. Dekodér paketov môže vygenerovať varovanie v prípade ak napríklad zistí, že je poškodená hlavička paketu, ak je paket príliš dlhý, v prípade nezvyčajného alebo chybného nastavenia volieb v hlavičke TCP paketu a podobne.
2. *Spracovanie preprocesormi* - Analyzované dáta sú zaslané preprocesorom, ktoré je možné pridávať a ovplyvňovať pomocou konfiguračných nastavení *Snort-u*. Preprocesor umožňuje, obecné povedané, spracovať prichádzajúce dáta s iným uhlom pohľadu.

Snort môže teda s použitím preprocesorov skúmať komunikáciu nielen na úrovni paketov, ale aj na vyššej úrovni komunikácie. V prípade, že pri konfigurácii nie je povolený žiaden preprocesor, Snort funguje iba ako statický filter paketov a skúma každý paket jednotlivo - používa takzvané *atomické vzory*. Tento prístup nie je príliš ideálny, pretože existujú rôzne techniky, ktoré umožňujú rozdelenie jedného paketu do viacerých tak, aby sa vyhli pozitívnemu porovnaniu s atomickým vzorom. Tieto techniky budú podrobnejšie popísané v nasledujúcej kapitole 2.1.3.

3. *Vyhodnotenie detekčným modulom* - Porovnávanie s uloženými pravidlami pomocou detekčného modulu predstavuje primárnu časť samotného spracovania paketu. Snort postupne prehľadáva zoznam pravidiel a v prípade, že dáta z paketu odpovedajú niektorému z pravidiel, generuje sa varovanie. Snort umožňuje pokračovať v porovnávaní aj keď je nájdená zhoda, tým je možné pre jeden paket vygenerovať aj viac rôznych varovaní.
4. *Logovanie varovaní* - Poslednú fázu predstavuje spracovanie vygenerovaných varovaní, keď je možné pomocou rôznych doplnkov definovať, ako bude s varovaniami ďalej naložené. Najčastejšie používané spôsoby sú jednoduché ukladanie do textového súboru alebo databázy, na druhej strane, existujú aj pokročilé varianty, ako napríklad odoslanie požiadavky na otvorenie vyskakujúceho okna na zvolený počítač, prípadne odoslanie e-mailu administrátorovi počítača.

Z uvedeného popisu môžeme vidieť, že Snort predstavuje pomerne robustný, efektívny, individuálne prispôsobiteľný a rozšíriteľný nástroj, ktorý býva nasadzovaný aj do reálnych sietí. Pri testovaní implementovanej knižnice bude, vo väčšine prípadov, schopnosť vyhnúť sa detekcii Snort-u považovaná za univerzálne riešenie.

1.5 Zhrnutie

V tejto kapitole sme sa zamerali na stručný popis základných princípov *Intrusion Detection Systémov*, ktorý však pokrýva iba úvod do problematiky, pretože existuje množstvo iných experimentálnych riešení, ktoré používajú nové, prípadne hybridné kombinácie spomenutých princípov. Z týchto dôvodov je aj uvedená klasifikácia iba orientačná, keďže kritéria rozdelenia sú pomerne nejednoznačné a v rôznej literatúre môže dochádzať k drobným odlišnostiam v niektorých detailoch, pričom ale fundamentálne princípy sú zhodné.

Keďže v našej práci sa budeme venovať obfuskácii sieťovej prevádzky s cieľom vyhnúť sa detekcii IDS, bolo nutné vymedziť tieto základné pojmy a princípy fungovania detekčných systémov, aby bolo možné problematiku podrobnejšie rozvinúť v ďalšom texte. V záverečnej časti kapitoly boli podrobnejšie popísané princípy implementácie IDS *Snort*, ktorá bude v tejto práci považovaná za *referenčnú* a bude využívaná pri testovaní implementovanej výslednej knižnice.

Kapitola 2

Princípy a klasifikácia obfuskácie

Obfuskácia je pojem pomerne náročný na popísanie, pretože sa v praxi používa v rôznych kontextoch. Pojem jednoducho povedané, označuje *transformáciu* určitých dát tak, že výsledné dáta majú rovnakú *sémantiku* a na druhej strane, majú tieto dáta inú *syntax*, s cieľom zmiatť nepovolaného užívateľa pri ich čítaní. Najčastejšie sa pojem spája s obfuskáciou zdrojového kódu u jazykov distribuovaných v zdrojovej podobe (napr. *JavaScript*), keď sa snažíme zamedziť získaniu zdrojového kódu inými užívateľmi, prípadne konkurenciou.

V tejto práci, ktorá sa zameriava na vyhnutie sa detekcii IDS, budeme však pod pojmom obfuskácia rozumieť transformáciu sieťového toku tak, aby IDS nebolo schopné rozpoznať tento tok, prípadne tak, aby neoznačilo tok ako potenciálne nebezpečný.

V nasledujúcej kapitole si stručne popíšeme súčasné trendy v obfuskácii sieťových dát, protokolov a sieťových útokov, ktoré boli získané štúdiom odbornej literatúry. Z týchto získaných znalostí bude v ďalšej práci vytvorená knižnica schopná obfuskovať sieťovú komunikáciu tak, aby nebola odhalená zvolenými IDS systémami.

2.1 Obfuskácia sieťovej komunikácie

V nasledujúcej podkapitole sa zameriame a obfuskáciu sieťovej komunikácie, keď je hlavným cieľom zamaskovať celú komunikáciu, prípadne, minimálne skryť výmenu informácií pomocou zakázaného protokolu v danej sieti. Možné využitie tohto typu obfuskácie môže byť teda najmä zamaskovanie celého komunikačného protokolu alebo zabezpečenie, že dáta prenášaná po sieti budú anonymné a prípadný útočník nebude schopný zrekonštruovať topológiu siete a identitu komunikujúcich entít.

Pri návrhu knižnice je dôraz kladený na čo najuniverzálnejšie zamaskovanie komunikačného protokolu, a preto sú tieto metódy kľúčovým odrazovým mostíkom pre návrh vlastného riešenia. Pozornosť v tejto kapitole zameriame na tri najpoužívanejšie a najrozšírenejšie metódy, ktoré sa objavujú v literatúre v najväčšej miere.

2.1.1 Obsfuskácia šifrovaním dát

Typickým príkladom tohto typu obfuskácie je vedecká práca [24], ktorá sa zaoberá *anonymizovaním dátových sád*, ktoré boli získané z významných a pomerne vyťažovaných sieťových uzlov. Cieľom autorov je transformovať dáta tak, aby nebolo možné zistiť *identitu* komunikujúcich užívateľov. Takéto dátové sady sú pomerne cenné v rôznych výskumoch, pretože sa dajú použiť pri experimentoch, napríklad pri skúmaní modelu správania siete,

štúdiu nových sieťových útokov, prípadne experimentami s novými protokolmi. Na druhej strane, v prípade voľného uvoľnenia takýchto dát, by mohlo dôjsť k silnému narušeniu súkromia užívateľov. Prípadný útočník by bol schopný na základe dát zrekonštruovať topológiu siete a rozpoznať jednotlivé komunikujúce entity. Tieto znalosti by mohli útočníkovi uľahčiť útoky na sieť ako napríklad *DoS* útok, na najslabší článok v danej sieti.

Autori tejto práce vychádzali z predošlých techník postavených na jednoduchom šifrovaní zdrojovej a cieľovej IP adresy. Avšak táto technika nie je považovaná za dostatočnú, pretože útočník je aj v tomto prípade schopný zrekonštruovať topológiu siete na základe ostatných dostupných dát v dátovej sade. Keďže šifrovanie IP adres, kde každá unikátna adresa má jednu zašifrovanú variantu (1 : 1) sa ukázalo ako nevyhovujúce, autori navrhli takzvanú (k,j) -obfuskáciu.

Princíp (k,j) -obfuskácie je postavený na transformácii každej unikátnej IP adresy za takzvaný „*groupID*“. Takýto princíp bol zvolený hlavne z dôvodu, že mapovanie unikátnej IP adresy na rôzne zašifrované hodnoty by viedlo k podstatnej strate informačnej hodnoty a prakticky k znehodnoteniu dát. Autori teda navrhli techniku, keď sa vykoná mapovanie $M : 1$ medzi hodnotou unikátnej IP adresy a pseudonáhodným *groupID*. Výsledkom tohto riešenia je, že vo výstupnej obfuskovanej dátovej sade je každá unikátna IP adresa nahradená za jednu z k možných IP adres.

Riešenie muselo byť ešte doplnené o ochranu voči takzvaným „*fingerprint*“ útokom, keď by bol útočník na základe *štatistických vlastností* dátových tokov jednotlivých užívateľov schopný stále zrekonštruovať sieťovú topológiu. (k,j) -obfuskácia teda používa princíp, keď sa do jednej skupiny, z ktorej sa generuje *groupID*, zaraďujú IP adresy, ktoré majú podobné štatistické vlastnosti. Z takto vytvorených skupín sa vytvoria funkcie, ktoré jednoznačne mapujú IP adresu na *groupID*. Uvedená technika je v práci podrobne formálne dokázaná s uvedením základného algoritmu novej obfuskácie. Metóda zaručuje dostatočnú ochranu pred zvoleným modelom útočníka, ktorý zohľadňuje reálne znalosti, ktoré môže útočník pri skutočnom nasadení nadobudnúť.

Obdobný spôsob je použitý aj v práci [9], kde sa vykonáva kódovanie *multicastových dát*, ktoré sú smerované po sieti, taktiež za účelom ich *anonymizovania*. Táto metóda teda nezabezpečuje samotné zamaskovanie komunikujúceho protokolu, ale poskytuje prostriedky na zašifrovanie prenášaných informácií tak, aby šifrovacie kľúče nemuseli poznať multicastové uzly po ceste medzi zdrojovým a cieľovým uzlom. Metóda je špecifická tým, že sa nešifrujú samotné prenášané dáta, ale šifrovanie je uplatnené iba na špeciálny vektor, ktorý uchováva informácie o samotnej obfuskácii prenášaných dát. Tento návrh umožňuje, že aj keď uzly nepoznajú šifrovacie kľúče, môžu agregovať pakety so spoločným cieľom do väčších, čo umožní zlepšenie využívania sieťových zdrojov.

Šifrovacie metódy je samozrejme možné použiť aj na zašifrovanie kompletných dát, čím sa podstatne zníži schopnosť odhaliť tento zašifrovaný tok detekčným systémom. Na druhej strane stále existuje šanca, že detekčný systém rozpozna komunikáciu na základe štatistických vlastností, prípadne bude šifrovaný tok, ak nebude vhodne zamaskovaný, označený za podozrivý. Šifrovanie a zamaskovanie komunikácie bude jednou z východiskových metód použitých pri návrhu obfuskacej knižnice. Návrhu vlastného riešenia bude podrobne venovaná zodpovedajúca kapitola 3.

2.1.2 Obfuskácia genetickými algoritmami

Zaujímavou možnosťou využitia *genetických algoritmov* sa zaoberá práca [15], ktorá sa orientuje na obfuskáciu sieťovej komunikácie z iného uhľa pohľadu ako v podkapitole 2.1.1. Práca demonštruje možnosti použitia genetických algoritmov pri transformácii sieťovej komunikácie. Tvorcovia tohto riešenia sa snažili ukázať tieto možnosti pri obfuskácii *skenovania portov* cieľového systému. V prípade, že útočník vykonáva takéto skenovanie systému, ktorý je chránený pomocou IDS (Network-Based), bude takýto útok s najväčšou pravdepodobnosťou odhalený a zaznamenaný.

Ako referenčné IDS bol v tomto prípade použitý voľne šíriteľný nástroj *Snort*, popísaný v kapitole 1.4, ktorý pri svojej detekcii skenovania portov používa metódu, pri ktorej skúma počet zaslaných paketov na porty jedného hostiteľského počítača v určitom časovom intervale. V prípade, že je prekročená určitá prahová hodnota tohto počtu, Snort označí komunikáciu ako skenovanie portov.

Princíp riešenia je možné v jednoduchosti popísať tak, že nástroj (obsahujúci kvôli vyhodnocovaniu aj samotný Snort) vytvorí počiatočnú populáciu jedincov (reprezentujúcich jednotlivé toky vykonávajúce skenovanie portov) s náhodnými hodnotami, pozostávajúcimi z hodnôt dopredu definovanej inštrukčnej sady (operácie ako nastavenie čísla portu, príznačkov paketu, odoslanie paketu a podobne). Následné je pre jednotlivých jedincov vyhodnotená ich *fitness funkcia* [11]. Výsledná hodnota fitness funkcie je zložená z počtu portov, ktoré dokáže daný jedinec preskúmať a počtu varovaní, ktoré vygeneruje pre túto postupnosť Snort. Pomocou nástrojov *lineárneho genetického programovania* (kríženie, mutácia, výmena) sa potom snažíme minimalizovať počet varovaní IDS a zároveň maximalizovať počet portov, ktoré preskúmame. Víťazný jedinec potom predstavuje postupnosť paketov, ktorá dosiahla najlepšie výsledky fitness funkcie pre dané nastavenie siete.

Výsledky v tomto prípade môžeme ovplyvniť rôznymi nastaveniami veľkosti populácie a počtu iterácií. Tvorcovia testovaním zistili, že vyhnutie sa detekcii je v tomto prípade jednoduchšie, ako dosiahnutie maximálneho preskúmania hostiteľských portov, ktoré zhoršovalo celkové výsledky výslednej fitness funkcie. Úspešnosť sa pri reálnych experimentoch pohybovala pri vyčíslení fitness funkcie približne v okolí 90%.

Keďže Snort je pomerne rozšírená implementácia IDS a mnoho z ďalších implementácií používa podobné metódy detekcie, prípadne, je na samotnom nástroji Snort priamo postavených, môžeme toto riešenie považovať za univerzálne.

Uvedená problematika obfuskácie pomocou genetických algoritmov môže byť použitá aj na poli transformácie konkrétnych útokov, nie len na modifikáciu komunikácie určitým protokolom. Tieto princípy sú využívané hlavne pri takzvaných „*mimicry útokoch*“, ktoré si podrobnejšie popíšeme v odpovedajúcej kapitole 2.2.3.

2.1.3 Maskovanie protokolov

Maskovanie protokolov v sieťovom toku patrí k špecifickým spôsobom využitia obfuskácie sieťových dát. Cieľom tejto metódy je transformovať určitý protokol v sieti na iný protokol tak, aby nebolo možné v sieti odhaliť, že k tejto transformácii došlo. Na detekciu a blokovanie nežiaducich protokolov v sieti, sa v súčasnosti využívajú nasledujúce princípy [22]:

- *Blokovanie portov* - Najzákladnejší druh blokovania protokolu, keď sieťové firewally jednoducho blokujú čísla portov známych protokolov. Napríklad, v prípade služby *telnet*, sú blokované všetky prichádzajúce komunikácie na port 23.

- *Statické filtrovanie paketov* - Metóda vychádzajúca z porovnávania vzorov, keď je potrebné vytvoriť ich databázu na základe známych signatúr, vyskytujúcich sa pri komunikácii protokolmi, ktoré chceme blokovať. Signatúry sú ale tvorené iba na úrovni jedného paketu - označované aj ako atomické.
- *Dynamické filtrovanie paketov* - Rozširuje statické filtrovanie tým, že ukladá a skladá aj fragmentované pakety a tie sa potom kontrolujú voči databáze vzorov. Vzory sú pri tomto prístupe zvyčajne označované aj ako zložené.
- *Stavové filtrovanie paketov* - Najefektívnejšia metóda, keď sa na rozdiel od predošlých variant skúmajú okrem syntaktických informácií z paketov, aj obsiahnuté sémantické informácie. Pomocou stavového automatu sa kontrolujú možné stavy, ktoré môže daný protokol pri komunikácii dosiahnuť. Tento prístup sa používa na kontrolu komunikácie protokolmi z aplikačnej vrstvy (napríklad FTP).
- *Filtrovanie založené na štatistických informáciách* - Pri tomto prístupe sa snažíme identifikovať komunikujúcu službu na základe rôznych štatistických informácií, ako približné veľkosti paketov, časy medzi príchodmi paketov, jitter a podobne. Jedná sa o pomerne výpočetne nenáročnú metódu, ktorá však nemusí dosahovať dobré výsledky, ak je charakteristika komunikácie pozmenená - napríklad pri preťažení siete.

Spomenuté prístupy detekcie sú v nasadzovaných systémoch implementované zvyčajne kombinovane, s cieľom dosiahnuť najefektívnejšiu detekciu. Napríklad implementácia IDS Snort využíva statické filtrovanie paketov, ktoré je doplnené aj o dynamické filtrovanie zloženými vzormi, doplnené čiastočne o štatistické filtrovanie. Stavové filtrovanie sa používa napríklad v programe *iptables*, ktorý je podrobnejšie popísaný v kapitole 3.5.2.

Vzhľadom na popísané princípy detekcie maskovania protokolov, je možné vytvoriť aj základné techniky slúžiace na vyhnutie sa detekcii. Jednotlivé techniky [22] si stručne popíšeme:

- *Migrácia služieb* - Jednoduchá metóda, zameraná na vyhnutie sa blokovania známych portov v sieti, keď je potrebné rekonfigurovať aplikácie tak, aby komunikovali po iných portoch, o ktorých sa vie, že sú *povolené*.
- *Fragmentovanie paketov* - Prístup snažiaci sa o vyhnutie sa porovnaniu s atomickými vzormi tým, že pôvodný paket sa rozdelí na viac fragmentovaných paketov. Riešenie, ako vyplýva z popisu, je účinné voči statickému filtrovaniu paketov, na druhej strane, dynamické filtrovanie je priamo na tento prístup zamerané.
- *Šifrovanie* - Jedná sa o priamočiary prístup, keď sa využíva šifrovanie celej komunikácie daným protokolom. Použitie šifrovania zabezpečí vyhnutie sa pozitívnemu porovnaniu, či už so statickým alebo dynamickým vzorom. Teoretická možnosť detekcie je možná však aj pri tejto metóde, keď môže systém komunikáciu rozpoznať na základe štatistických informácií.

Spomenuté základné techniky môžeme samozrejme taktiež kombinovať, aby sme detekčnému systému v maximálnej miere sťažili možnosti rozpoznania. Pokročilejšie metódy si stručne popíšeme v nasledujúcom texte.

Zaujímavé riešenie maskovania protokolu je možné nájsť v práci [17], ktorá využíva obfuskáciu protokolu *Tor siete* tak, aby sa v sieti javil ako šifrovaný video hovor služby

Skype. Táto technika skrývania dát v iných dátach sa nazýva *steganografia*. Cieľom autorov je touto technikou zamedziť blokovaníu a cenzurovaníu *Tor* protokolu, slúžiaceho na anonymné a šifrované pripojenie do internetu.

Tvorcovia v tomto prípade zvolili protokol služby *Skype* ako cieľový protokol hlavne z dôvodu, že sa jedná o pomerne rozšírený protokol (nepredpokladá sa teda, že by došlo k blokovaníu a cenzúre masívne používaného protokolu), ktorý obsahuje šifrované dáta. Na druhej strane použitie tohto protokolu je nevýhodou hlavne z dôvodu, že sa jedná o *proprietárny protokol*, ktorý nemá verejne dostupnú svoju implementáciu.

Blokovanie prístupových *Tor* uzlov v krajinách s rozvinutou cenzúrou je zapríčinené hlavne tým, že zoznam takzvaných „*onion routrov*“, ktoré zabezpečujú prístup do siete, je dostupný verejne na tzv. „*directory serveroch*“, ktorých adresy sú taktiež verejne známe. S týmito znalosťami môže cenzor veľmi jednoducho zablokovať prístup k týmto uzlom v sieti. *Tor* protokol, ako reakciu na takéto blokovanie, začal používať takzvané „*bridge uzly*“, ktorých adresy nie sú verejne známe. Ale aj v tomto prípade je možné pri podrobnej analýze cenzorom odhaliť tieto uzly.

Tvorcovia tejto experimentálnej techniky vytvorili doplnok pre štandardného *Tor* klienta, kde si môže užívateľ zvoliť, či chce používať túto obfuskáciu. Princíp spočíva v tom, že ak chce užívateľ nadviazať komunikáciu, nástroj najskôr inicializuje *fiktívne prihlásenie* sa do služby *Skype* a inicializuje fiktívny hovor smerujúci na *bridge uzol*. V momente, keď *bridge uzol* prijme hovor, samotný hovor sa zruší a vytvorený kanál sa používa na zasielanie obfuskovaného dátového toku. Samotná komunikácia potom prebieha pomocou *UDP* protokolu, preto bolo nutné implementovať princípy bezpečného prenosu ako znovu zasielanie a potvrdzovanie paketov, čo čiastočne znižuje efektívne využívanú šírku pásma.

Keďže, ako bolo spomenuté v kapitole 1.3.1, typ šifrovaného protokolu je možné zistiť aj na základe štatistických informácií, do nástroja bolo nutné implementovať aj pokročilé metódy prispôsobovania rýchlosti a kvality hovoru, ktorú *Skype* protokol používa na efektívne využitie šírky prenosového kanála. Implementácia tejto vlastnosti zabezpečuje, že časové intervaly medzi odoslanými paketmi a aj samotná veľkosť paketov, bude približne odpovedať skutočnému toku protokolu *Skype* a cenzor by nemal byť schopný zistiť ani pomocou podrobnej analýzy štatistických vlastností, že sa jedná o obfuskovaný sieťový tok.

Výsledné riešenie v experimentoch dosahovalo pomerne dobré výsledky. Nevýhodou bolo už spomenuté slabšie využitie prenosového pásma, ktoré bolo znížené o približne 25%. Aj napriek tejto nevýhode sa jedná o perspektívne riešenie.

Podobnými problémami sa zaoberá aj práca [10], odkiaľ boli čerpané doplňujúce informácie pre túto sekciu, ktorá skúma možnosti klasifikácie šifrovaných obfuskovaných protokolov na základe štatistických vlastností a hodnôt doplnkových polí v hlavičke paketov.

2.2 Obfuskácia konkrétnych útokov

V nasledujúcej kapitole si stručne popíšeme metódy, ktoré boli vytvorené priamo na obfuskáciu konkrétnych útokov a nie na maskovanie určitej komunikácie. Pri tomto type obfuskácie sa využívajú známe zraniteľnosti cieľových systémov a nedokonalosti samotných *IDS* systémov. Tieto techniky sú z tohto dôvodu často závislé na použitom hardwarovom a softwarovom vybavení cieľovej stanice, na ktorú je útok smerovaný. Výhodou týchto riešení je možnosť vyladiť transformáciu špecificky na daný útok, a tým zabezpečiť najvyššiu pravdepodobnosť toho, že útok nebude odhalený. Na druhej strane je nevýhodné, že automatické generovanie obfuskovaných útokov nie je mnohokrát prakticky možné, pretože je nutné

najskôr preskúmať cieľový počítač, prípadne celú cieľovú sieť a až na základe získaných vedomostí vytvoriť transformáciu daného útoku. Základné princípy týchto techník si popíšeme v tejto kapitole.

2.2.1 Obfuskácia HTTP protokolu

Techniky obfuskácie HTTP protokolu sa začali rozvíjať s nástupom útokov na HTTP servery, keď boli útoky zamerané hlavne na prienik do súborového systému stroja, na ktorom samotný server bežal. Prevencia proti týmto útokom sa preniesla do IDS systémov, pretože IDS bolo schopné kontrolovať dátové pakety, a tým odhaľovať možné útoky. Z tohto dôvodu bolo potrebné riešiť obfuskáciu týchto útokov. Spomínanou problematikou sa zaoberá autor v článku [25].

Hlavným cieľom detekcie podozrivých paketov je nájdenie hodnoty URL adresy v dátovej časti. IDS pri kontrole paketov HTTP protokolu používa dve základné techniky, a to konkrétne *porovnávanie vzorov* (pattern matching) a *kompletnú analýzu protokolu*. V súčasnosti sa používajú obe techniky súčasne, aby sa zvýšila pravdepodobnosť úspešnej detekcie podozrivej URL.

Porovnávanie vzorov pri svojej činnosti používa dáta z celého paketu, ktoré porovnáva so vzormi uloženými v databáze. Na druhej strane, kompletná analýza protokolu je schopná odhaliť URL v pakete a ku kontrole sa používa iba táto adresa. Z tohto dôvodu je protokolová analýza úspešnejšia v prípadoch, keď je URL prenášaná v zakódovanej forme a metóda je schopná použiť dekodovacie mechanizmy a kontrolovať dekodovanú adresu.

S cieľom obfuskácie sa používajú dve základné techniky, a to konkrétne *nesprávne parsovanie protokolu* (invalid protocol parsing) a *nesprávne dekódovanie protokolových polí* (invalid protocol field decoding). Princípy týchto techník [25] si v stručnosti popíšeme:

- *Nesprávne parsovanie protokolu* sa využíva v prípadoch, keď IDS úplne nespĺňa štandardy protokolu. Ako príklad sa dá uviesť vlastnosť, keď v jednom HTTP pakete môže byť obsiahnutých viac požiadaviek. IDS je v tomto prípade najčastejšie schopné vyparsovať adresu z prvej požiadavky a zvyšné adresy zostanú nepreskúmané. Útočníkovi teda stačí zadať kompromitujúcu adresu do druhej požiadavky a v mnohých prípadoch nedôjde k jej odhaleniu.
- *Nesprávne dekódovanie protokolových polí* využíva toho, že v HTTP štandarde je navrhnutých veľké množstvo spôsobov zakódovania jednotlivých polí požiadavky pre rôzne typy webových serverov. Útočník v tomto prípade využije rôzne druhy kódovaní, ktoré potom IDS v mnohých prípadoch nevie dekódovať, a preto nemôže odhaliť ani samotnú adresu. Do tejto kategórie zaradujeme aj útoky s využitím väčšieho množstva lomítok v adrese namiesto použitia jediného, ktoré môže útočníkovi zabezpečiť prístup do súborového systému servera. Proti týmto útokom už sú však v súčasnej dobe samotné servery zabezpečené.

Popísané techniky ukazujú základné metódy obfuskácie HTTP protokolu, ktoré boli používané hlavne v minulosti, na druhej strane názorne ukazujú princípy obfuskácie protokolu, ktoré môžeme aplikovať aj na iné komunikačné protokoly. Ako vyplýva z popisu, pri obfuskácii útoku na úrovni protokolu, je nutné odhaliť a využiť slabé miesta, ktoré vznikajú hlavne pri nekorektnom spracovávaní detekčným systémom.

Tieto uvedené znalosti je ďalej možné využiť pri obfuskácii iných protokolov. Ako príklad je možné uviesť techniky ako obfuskácia FTP protokolu, prípadne útoky na iné aplikačné služby (MySQL, Samba a podobne).

2.2.2 Obfuskácia shellkódu a systémových volaní

Tento typ obfuskácie patrí k pomerne rozšíreným technikám používaným v praxi. Jedná sa o obfuskáciu konkrétneho útoku, zameriavajúceho sa na určitú zraniteľnosť v systéme. Pri bežných útokoch, ktoré sú známe a používané, vo väčšine prípadov dochádza k ich rozpoznaní pomocou IDS, ktoré ich majú uložené vo svojej databáze. Z tohto dôvodu je nutné modifikovať tieto útoky do nerozpoznaiteľnej podoby.

Príklad použitia môžeme nájsť v práci [30], ktorej princípy si stručne popíšeme. Autori sa v tomto prípade zamerali na obfuskáciu zameranú na IDS systémy používajúce sledovanie *systémových volaní* - možné v *Host-Based IDS*. Táto detekčná technika je postavená na skúmaní postupnosti systémových volaní a jej porovnávaní s uloženými vzormi z databázy. Ak aktuálna postupnosť odpovedá nejakému útoku, systém vygeneruje varovanie. Keďže systém nemôže ukladať nekonečnú postupnosť, zvyčajne sa používa konštantná veľkosť okna systémových volaní, ktorá sa porovnáva.

Obfuskáčna technika je založená na tom, že sa snaží medzi systémové volania, potrebné na útok, vložiť ďalšie volania, ktoré nezmenia pôvodnú funkčnosť kódu. Týmto krokom dôjde k prekročeniu dĺžky vyrovnávacej pamäte systémových volaní IDS. Jednotlivé volania útoku nebudú teda nikdy spolu vo vyrovnávacej pamäti a preto IDS nemôže útok odhaliť.

Podobný princíp je popísaný aj v článku [1], s tým rozdielom, že v tomto prípade obfuskujeme *shellkód*. Autori v tejto práci navrhujú možný spôsob detekcie takéhoto upraveného kódu pomocou *Network-Based IDS*, ktoré nemôže priamo sledovať systémové volania cieľového systému. Táto detekcia je postavená v prvom kroku na detekcii shellkódu priamo v dátovej časti paketu. Po úspešnej detekcii je paket zaslaný druhému modulu na analýzu.

Pri analýze je vyextrahovaný samotný shellkód a tento kód je vykonaný v samostatnom vlákne priamo v IDS. Pomocou systémového nástroja `ptrace` je vyextrahovaná postupnosť systémových volaní, ktorú môžeme testovať obdobne ako v predošlom prípade. Nevýhodou tohto riešenia je, že shellkód sa pre rôzne platformy a operačné systémy môže líšiť a z tohto dôvodu ho nebude možné v prostredí daného IDS vykonať.

2.2.3 Mimicry útoky

Mimicry útoky predstavujú zovšeobecnenie princípov popísaných v predošlej kapitole venovanej obfuskácii shellkódu a systémových volaní, keď je obfuskácia použiteľná na ľubovoľné dáta. Táto myšlienka obfuskácie využíva vlastnosť IDS systémov, keď sa predpokladá, že IDS pri svojom spracovaní využíva vyrovnávaciu pamäť o určitej veľkosti, do ktorej sa ukladajú aktuálne spracovávané dáta. Útočník sa snaží modifikovať dáta tak, aby sa dáta útoku do tejto vyrovnávacej pamäte nezmestili a útok nemohol byť porovnaný s odpovedajúcim pravidlom.

V počiatočnej fáze obfuskácie disponuje útočník takzvaným *jadrom útoku* (*core attack*), ktoré obsahuje nevyhnutné dáta na vykonanie tohto útoku. Tieto dáta sa potom snaží transformovať do podoby, ktorá bude z pohľadu IDS korektná a IDS nebude schopné odhaliť v komunikácii žiadne stopy po útoku.

Túto činnosť je možné vykonávať čiastočne automaticky, využívané sú najmä genetické algoritmy [14], podrobnejšie popísané v kapitole 2.1.2, prípadne obfuskácia shellkódu [13], ak je útok založený na prenose a vykonaní tohto shellkódu. Často je však potrebný ľudský zásah do vytváraného útoku tak, aby došlo k využitiu nejakej vlastnosti v cieľovom systéme.

Základné používané techniky môžeme rozdeliť na základe sieťových vrstiev, kde sú aplikované [29], na:

- *Sieťové techniky obfuskácie* - Patria sem techniky, ktoré sú aplikované na sieťovej a transportnej vrstve. Ako základný princíp možno považovať rozdeľovanie IP paketov, keď je útok fragmentovaný do menších častí, prípadne existujú techniky postavené na IPv6 komunikácii (ak je dostupná), pretože v mnohých prípadoch podpora pre IPv6 do detekčných systémov nastupuje pomaly a systémy nie sú schopné kontrolovať túto komunikáciu na zodpovedajúcej úrovni.
- *Aplikačné techniky obfuskácie* - Do tejto kategórie spadajú metódy aplikované na relačnú, transportnú a aplikačnú vrstvu sieťového modelu. Jedná sa teda vo väčšej miere o metódy využívajúce vlastnosti niektorých protokolov, napríklad SSL, HTTP, FTP a podobne.
- *Techniky obfuskácie exploitu* - V tomto prípade dochádza k modifikácii priamo na úrovni samotného exploitu. Patria sem metódy, ako v kapitole 2.2.2 spomínaný, *polymorfny shellkód*, prípadne techniky postavené na rôznej zmene kódovania (napríklad *base64*) dát.

Uvedené techniky je možné samozrejme vzájomne kombinovať na jednotlivých vrstvách tak, aby bola dosiahnutá maximálna pravdepodobnosť, že k detekcii na IDS nedôjde. Z popisu však vyplýva aj to, že techniky nie sú úplne univerzálne a zvyčajne je potrebný zásah človeka do procesu obfuskácie aj v prípade snahy o automatické generovanie obfuskovaného toku.

2.3 Zhrnutie

V tejto kapitole sme sa zamerali na popis základných obfuskačných techník a ich klasifikácie, ktoré sú používané v súčasnosti. Klasifikáciu jednotlivých metód je možné vykonať na základe rôznych kritérií, pretože princípy jednotlivých techník sa čiastočne prekrývajú, čo umožňuje vytvoriť rôzne interpretácie rozdelenia. Z tohto dôvodu, uvedené rozdelenie predstavuje iba jeden možný variant. Získané znalosti o jednotlivých metódach budú následne použité pri návrhu vlastnej knižnice, ktorá by mala byť schopná obfuskovať určité protokoly v sieťovej komunikácii.

Pre potreby navrhovanej knižnice sa ako kľúčové javia metódy postavené na maskovaní protokolov, kryptografické metódy a čiastočne aj mimicy útoky. Na druhej strane knižnica má byť čo možno najuniverzálnejšia. Z tohto dôvodu nie je možné použiť tieto metódy v plnej miere. Návrh a implementácia preto využíva tieto techniky iba ako odrazový mostík pri vytváraní *vlastného spôsobu maskovania*, ktoré má za cieľ obfuskovať komunikáciu univerzálne, vzhľadom k používanému IDS a komunikačnému protokolu. Návrh knižnice bude podrobnejšie popísaný v nasledujúcom texte.

Kapitola 3

Návrh a implementácia knižnice

Nasledujúca kapitola sa podrobnejšie zameriava na oboznámenie s návrhom, implementáciou a vlastným fungovaním obfuskačnej knižnice. Pri návrhu je kladený dôraz na univerzálnosť knižnice, ktorá by mala byť schopná transformovať či už *legitímnu komunikáciu*, prípadne aj komunikáciu, ktorá nesie *kompromitujúce dáta*, teda rôzne útoky a exploity. Druhou významnou požiadavkou pri návrhu je tiež jednoduchosť používania užívateľom, ktorý nemusí potrebovať dôkladné znalosti princípu obfuskácie.

Dôležitou časťou návrhu je definovanie predpokladanej štruktúry a topológie siete, v ktorej môže byť obfuskácia používaná. Z požiadavky, na používanie obfuskácie v rozsiahlom sieťovom prostredí, vyplýva preto nutnosť distribuovať jednotlivé moduly v rámci danej siete. V popise sa z tohto dôvodu zameriame oddelene na špecifikáciu nosnej obfuskačnej časti programu a na druhej strane, na definovanie distribuovaného, podporného proxy modulu, ktorý bude dekodovať dáta vo vnútornej sieti.

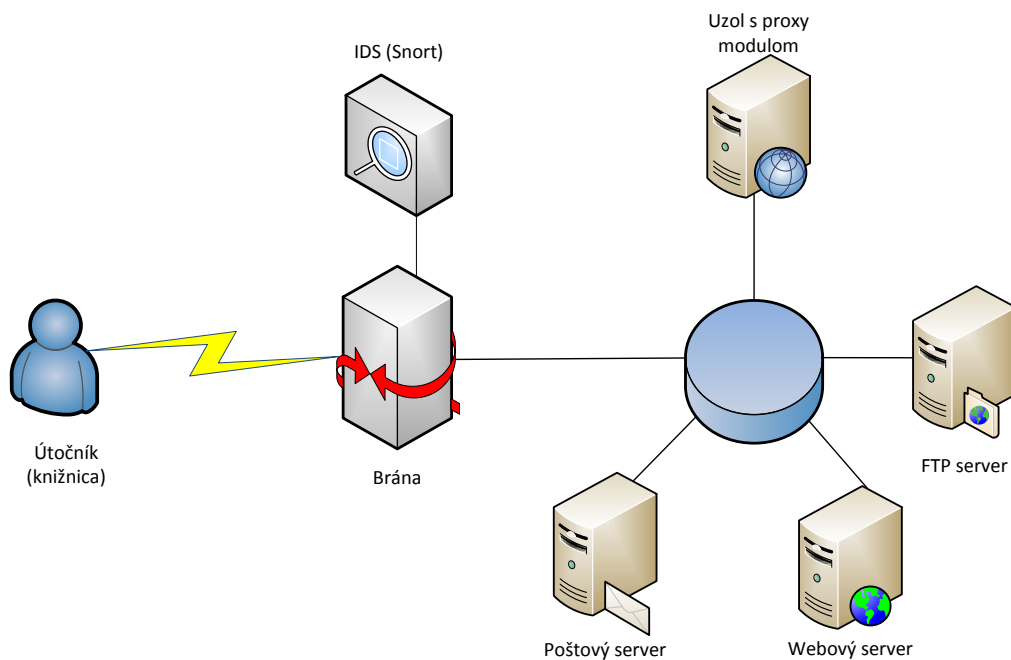
3.1 Architektúra cieľovej siete

Pri návrhu knižnice bolo potrebné určiť približnú *topológiu siete*, do ktorej môže byť obfuskačná knižnica nasadená, pretože od tejto charakteristiky sa odvíja celý ďalší návrh knižnice. Očakávaná štruktúra siete je uvedená na obrázku 3.1.

Nasadenie knižnice, ako vyplýva z obrázku 3.1, očakáva prostredie, v ktorom existuje nejaká vnútorná sieť, komunikujúca s okolitým svetom prostredníctvom sieťovej brány. Predpokladá sa, že sieťová brána je vybavená IDS systémom, prípadne sondou, a teda všetka komunikácia prechádzajúca z vonkajšej siete do vnútornej, a naopak, je kontrolovaná IDS systémom, ktorý je schopný kontrolovať a odhaliť prípadné pokusy o prenos kompromitujúcich dát. Na počítače v samotnej chránenej vnútornej sieti nie sú kladené ďalšie špecifické požiadavky, knižnica by teda mala byť schopná pracovať bez ohľadu na architektúru vnútornej siete.

Významným prvkom zabezpečujúcim realizáciu samotnej obfuskácie je aj *proxy modul*, ktorý sa musí nachádzať vo vnútornej chránenej sieti. Predpokladom teda je, že na niektorom hostiteľskom počítači vo vnútornej sieti tento proxy modul beží. Úlohou proxy modulu je dekodovať prichádzajúcu obfuskovanú komunikáciu a zabezpečiť jej ďalšie šírenie vo vnútornej sieti.

Obfuskácia by prakticky bez proxy modulu nebola možná, pretože pri zasielaní obfuskovaného toku *priamo na cieľový počítač*, by boli možnosti transformácie komunikácie značne obmedzené, prípadne reálne nerealizovateľné, pretože by príjemca nebol schopný obfuskované dáta dekodovať. Dôvodom je, že v tomto prípade by museli byť dodržiavané



Obrázok 3.1: Príklad topológie siete

štandardy komunikácie obfuskovaného protokolu a pre nemodifikovaného príjemcu by bolo dekódovanie dát nereálne. Vzhľadom na tieto vlastnosti, bol zvolený prístup riešenia s proxy modulom, ktorý je efektívnejší ako špecifická modifikácia klienta príjemcu.

Vlastné zavedenie proxy modulu do chránenej siete táto práca priamo nerieši, pretože je špecifické vzhľadom na konkrétnu štruktúru chránenej siete a tiež závisí aj od spôsobu využitia knižnice. V prípade používania knižnice pri penetračnom testovaní, stačí modul jednoducho nainštalovať v danej sieti. Na druhej strane, pri ofenzívnom využívaní útočníkom, je potrebné odhaliť zraniteľné miesta niektorého z hostiteľov, prelomiť počítač a po eskalovaní prístupových práv zabezpečiť distribúciu a spustenie proxy modulu. Potom je možné obfuskovať útoky na ďalšie počítače v rámci danej siete.

Ako vyplýva z popisu, tak celá obfuskovaná komunikácia je smerovaná z vonkajšej siete cez sieťovú bránu do proxy modulu. Proxy modul zabezpečí na základe definovaných parametrov dekódovanie prichádzajúceho dátového toku a rozhodne, kam bude dokódovaná komunikácia vo vnútornej sieti smerovaná. To znamená, že úlohou knižnice je zabezpečiť a zamaskovať odchádzajúcu komunikáciu v smere do vnútornej chránenej siete tak, aby nedošlo k jej odhaleniu a pri analýze v detekčnom systéme sieťovej brány nevykazovala žiadne známky neštandardnej komunikácie (*anomálie*), a teda javila sa vo všetkých prípadoch ako skutočná korektná komunikácia daným protokolom.

3.2 Princíp fungovania a použitia knižnice

Základný princíp fungovania obfuskačnej knižnice je postavený na transformácii odchádzajúcej komunikácie, ktorá spadá pod pravidlá obfuskačie a jej následnom zaslaní v transformovanej podobe na proxy modul. Proxy modul musí poskytnúť dekódovanie tejto prichádzajúcej komunikácie a na základe definovaných parametrov jej ďalšiu distribúciu vo vnútornej sieti.

Proxy modul v tomto prípade zabezpečuje aj, aby boli prípadné odpovede od cieľového počítača *zasielané proxy modulu* a nie priamo von zo siete. Z tohto dôvodu je potrebné vykonať dynamickú modifikáciu hlavičiek zasielaných paketov tak, aby obsahovali namiesto skutočnej IP adresy, *podvrhnutú adresu* proxy modulu. Týmto sa dosiahne, že proxy modul bude fungovať ako prostredník komunikácie medzi užívateľom knižnice a cieľovým počítačom.

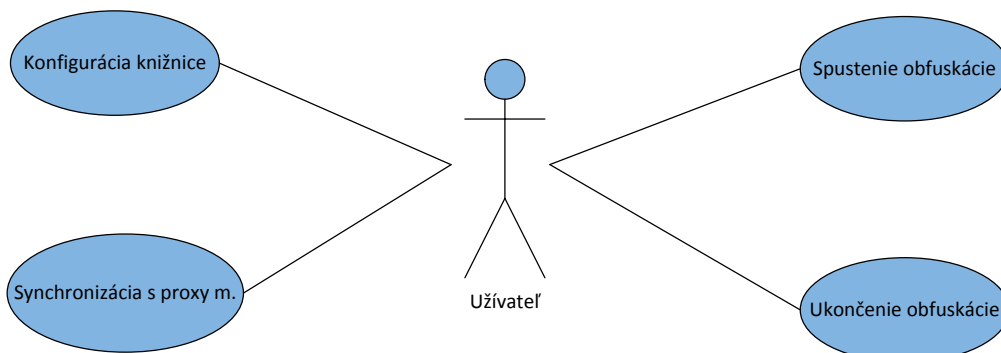
Na strane programu, využívajúceho knižnicu, je potrebné vykonávať odchytyvanie odchádzajúcej komunikácie pokročilým spôsobom, pretože cieľom je, aby užívateľ nemusel nijako modifikovať, prípadne konfigurovať vlastný komunikačný klient, ktorý chce využiť pri zasílení dát do vnútornej chránenej siete. Knižnica z tohto dôvodu zabezpečí odchytyvanie odosielaných paketov na *systémovej úrovni*, keď dôjde k zakódovaniu/dekódovaniu obfuskovaného toku na úrovni *jadra operačného systému*, čím sa stane obfuskačia pre vyššie vrstvy prakticky neviditeľná. Tieto vlastnosti predstavujú dôležitý prvok pri návrhu knižnice, aby sme dosiahli možnosť čo najefektívnejšieho a najintuitívnejšieho spôsobu možného využitia užívateľom.

3.2.1 Diagram prípadov použitia

Možnosti používania knižnice sú značne ovplyvnené dôrazom na jej jednoduché, intuitívne a efektívne rozhranie. Užívateľ z tohto dôvodu môže pri implementácii využiť jediné rozhranie knižnice, ktoré poskytuje možnosti nastavenia a riadenia obfuskačného procesu. Interakcia užívateľa s rozhraním knižnice je zobrazená pomocou diagramu prípadov použitia, ktorý je súčasťou modelovacieho jazyka UML, na obrázku 3.2.

Jednotlivé prípady použitia knižnice si stručne popíšeme:

- *Konfigurácia knižnice* - Združuje kompletne nastavenie knižnice, keď môže užívateľ nastaviť základné parametre obfuskačie, ako IP adresa cieľového počítača, prípadne maska celej cieľovej siete. Užívateľovi je taktiež umožnené špecifikovať parametre obfuskačie, ako výber používaného obfuskačného modulu a jeho konfigurácia, prípadne podrobnejšie špecifikovanie odosielaných obfuskovaných dát. Vlastná konfigurácia prebieha pomocou *konfiguračného súboru*, podrobnosti o jeho syntaxi a sémantike sú popísané v odpovedajúcej kapitole 3.8.
- *Synchronizácia s proxy modulom* - Pred spustením samotnej obfuskačie je potrebné vykonať synchronizáciu s proxy modulom, ktorý sa nachádza vo vnútri chránenej siete. Pri synchronizácii sa v prvej fáze nadviaže spojenie, vykoná sa *autentizácia* proxy modulu voči užívateľskému programu využívajúceho knižnicu, zabezpečí sa synchronizácia obfuskačných nastavení a obe strany sa pripraví na vzájomnú komunikáciu.
- *Spustenie obfuskačie* - Umožňuje užívateľovi zahájiť obfuskačný proces, keď dôjde k zavedeniu prvkov knižnice do hostiteľského systému, pričom dôjde k presmerovaniu komunikácie spadajúcej do obfuskačných kritérií do proxy modulu a komunikácia bude zároveň transformovaná zvoleným obfuskačným modulom. Knižnica v tomto



Obrázok 3.2: Diagram prípadov použitia

prípade zabezpečí kompletnú obsluhu spracovávaných dát, správu obslužných vlákien slúžiacich na spracovávanie dátových paketov a podobne.

- *Ukončenie obfuskácie* - Zabezpečí užívateľovi ukončenie obfuskáčneho procesu, keď dôjde k uvoľneniu zdrojov využívaných knižnicou, ukončeniu obslužných vlákien používaných na obsluhu a návratu používaných prvkov operačného systému do *pôvodného stavu* pred spustením.

Hlavným dôvodom vytvorenia takéhoto jednoduchého rozhrania, je snaha o čo najvyššie zapuzdrenie samotnej obfuskácie pred užívateľom, pretože samotná obfuskácia bude musieť prebiehať na nižšej systémovej úrovni, hlavne z dôvodu, že na transformáciu komunikácie sa využijú čiastočne aj prvky hostiteľského operačného systému. Podrobnejšie na implementáciu sa zameriava odpovedajúca kapitola 3.3.

Výhodou tohto riešenia je, že užívateľ po počiatočnom nastavení knižnice, pomocou konfiguračného súboru a spustení obfuskáčneho procesu, nemusí vykonať žiadne ďalšie operácie pri komunikácii so vzdialeným počítačom. Na komunikáciu postačujú *bežné aplikácie*, ktoré by sa využívali aj v prípade, že sa nepoužíva obfuskáčna knižnica. Toto riešenie tiež poskytuje možnosť použiť knižnicu útočníkom aj reverzným spôsobom, keď by došlo k distribúcii nakonfigurovaného obfuskáčneho programu svojej obei. Pomocou čiastočne upraveného proxy modulu by potom mohol útočník odpočúvať komunikáciu určitým protokolom, prípadne vykonať nejaký *man-in-the-middle* útok.

3.3 Štruktúra knižnice

Na základe prvotného návrhu popisu očakávaného fungovania riešenej obfuskácie, bolo potrebné vytvoriť abstraktný model, popisujúci rozloženie jednotlivých komponent obfuskacej knižnice a jej rozhranie určené na použitie užívateľom. Dôraz pri vytváraní sa kládol na jednoduchosť tohto rozhrania tak, aby bolo jeho používanie intuitívne a efektívne. Na druhej strane významným kritériom bolo vytvorenie návrhu, ktorý by umožňoval ďalšie možnosti pre budúce rozšírenie o nové moduly. V súčasnej verzii obsahuje knižnica implementáciu dvoch obfuskacích modulov, konkrétne modulu zameraného na obfuskáciu zvolených komunikačných protokolov a modulu slúžiaceho na maskovanie komunikácie vykonávajúcej skenovanie otvorených portov cieľového počítača.

Ako najefektívnejší spôsob popisu štruktúry sa javí použitie diagramu tried jazyka UML, v ktorom zobrazíme jednotlivé triedy figurujúce v obfuskacej knižnici a vzájomné vzťahy medzi nimi. Vytvorenú štruktúru môžeme vidieť na obrázku 3.3, jednotlivé triedy si podrobnejšie popíšeme v nasledujúcich podsekcích.

3.3.1 Trieda *ObfusLibInterface*

Trieda *ObfusLibInterface* tvorí jediné rozhranie knižnice, ktoré by malo byť používané užívateľom, jedná sa teda o najvyššiu triedu vo vytvorenej hierarchii. Trieda zabezpečuje inicializáciu podriadených komponent, ich vzájomnú komunikáciu, synchronizáciu, riadenie obfuskácie na základe vstupných nastavení a správu spoločných využívaných zdrojov knižnice.

Jednotlivé významné metódy tejto triedy si stručne popíšeme:

- **config** - umožní užívateľovi definovať nastavenie knižnice na základe požadovaných kritérií obfuskácie. Ako vstup sa očakáva názov konfiguračného súboru, ktorý obsahuje definíciu potrebných nastavení. Podrobnejší popis štruktúry konfiguračného súboru je zahrnutý v odpovedajúcej kapitole 3.8. Implementácia tejto metódy teda odpovedá prípadu použitia „*Konfigurácia knižnice*“, popísanej v diagrame prípadov použitia knižnice z kapitoly 3.2.1.
- **syncProxy** - vykoná synchronizáciu knižnice s proxy modulom, ktorý sa nachádza vo vnútornej chránenej sieti. Súčasťou synchronizácie je aj autentizácia proxy voči knižnici, vytvorenie vzájomného komunikačného kanála a zjednotenie parametrov obfuskácie medzi oboma stranami. Obdobne, ako v predošlom prípade odpovedá implementácia metódy prípadu použitia „*Synchronizácia s proxy modulom*“.
- **start** - zabezpečí spustenie samotného obfuskacej procesu, keď sa uvedie do činnosti zvolený nakonfigurovaný obfuskacích modul. Kompletne riadenie behu prevezme daný použitý obfuskacích modul a rozhranie knižnice pasívne čaká na ukončenie. Ukončenie môže nastať dvomi spôsobmi, konkrétne zaslaním signálu SIGINT programu, prípadne, ak dôjde k prerušeniu naviazaného spojenia s proxy modulom. Implementácia tejto metódy odpovedá prípadu použitia „*Spustenie obfuskácie*“ z popísaného diagramu prípadov použitia.
- **stop** - jedná sa o pomocnú privátnu metódu, ktorá slúži na bezpečné ukončenie obfuskacej procesu v prípade, ak zašle užívateľ signál na zastavenie. V tomto prípade dôjde k bezpečnému ukončeniu používaných modulov a vrátenie riadenia užívateľskému programu, ktorý knižnicu používa.

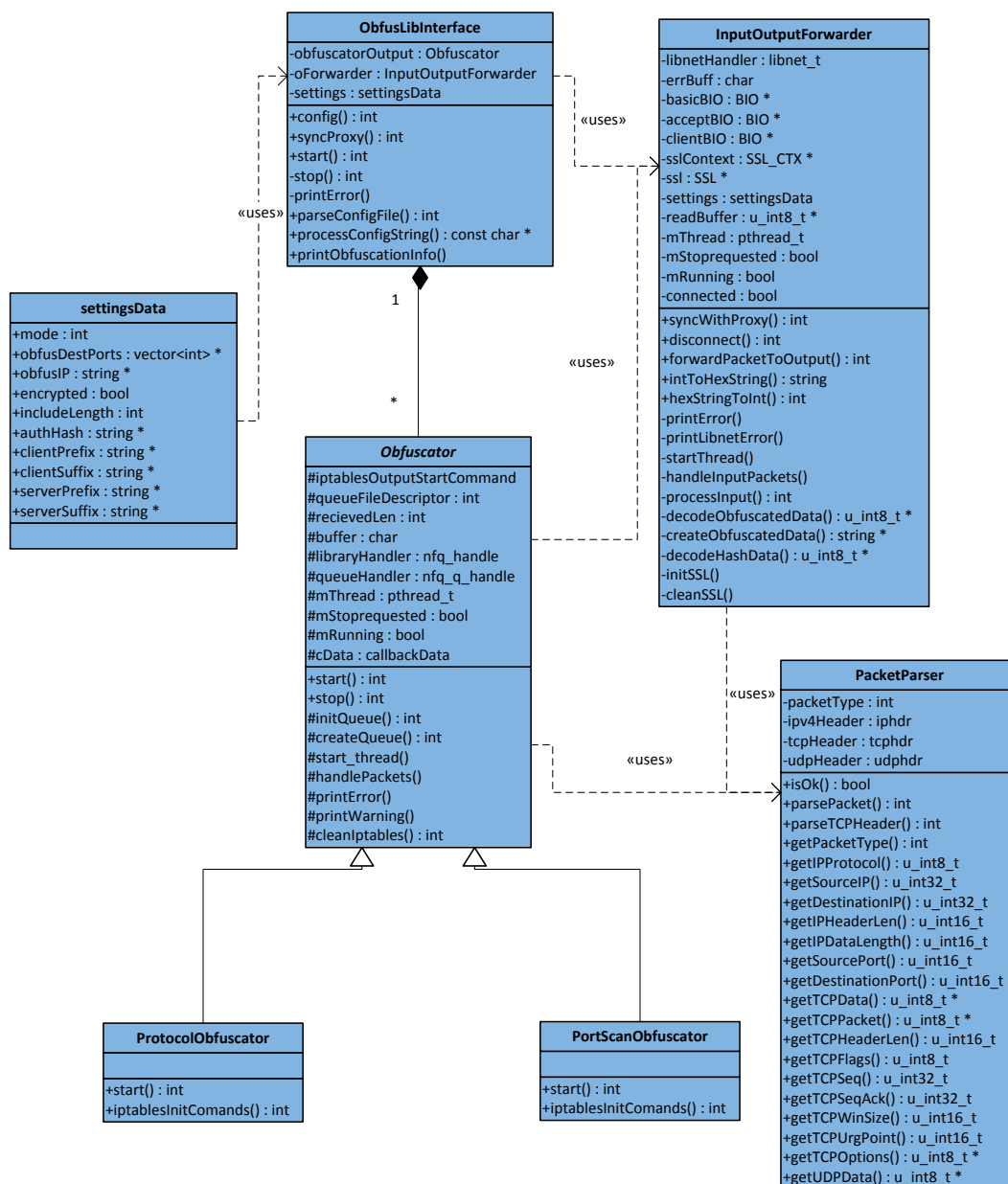
- `parseConfigFile` - pomocná metóda slúžiaca na extrakciu dát z konfiguračného súboru zadaného pri inicializácii knižnice. Metóda zabezpečí kontrolu konfiguračného súboru, odstránenie sémanticky nevýznamných riadkov a získanie konkrétnych hodnôt parametrov obfuskácie.
- `processConfigString` - metóda využívaná pri spracovaní jednotlivých príkazov získaných z konfiguračného súboru, slúžiaca najmä na rozdelenie konfiguračného riadku na hodnotu príkazu a hodnoty parametrov. Metóda je využívaná pri spracovávaní pomocou predošlej metódy `parseConfigFile`.

Zvyšné implementované metódy triedy `obfusLibInterface` nemajú z hľadiska samotnej funkčnosti valný význam, slúžia iba na jednotný formát výstupu knižnice. Konkrétne `printError` umožňuje formátovaný výpis chybového hlásenia, napríklad v prípade chybovej inicializácie a podobne. Metóda `printObfuscationInfo` vypisuje súhrnné informácie pre užívateľa o aktuálnom nastavení knižnice, ktoré sú vypísané pred spustením vlastného obfuskačného procesu.

3.3.2 Trieda *Obfuscator*

Trieda `Obfuscator` predstavuje nosnú časť implementácie obfuskácie. Jedná sa o *abstraktnú triedu*, obsahujúcu spoločné prvky jednotlivých obfuskačných modulov, ktorá ďalej slúži pri vytváraní konkrétnych implementácií nových modulov, keď každý modul musí dediť rozhranie tejto triedy. Implementácia tejto triedy je zameraná na vykonanie potrebných inicializácií, ktoré sú nutné pre beh konkrétneho obfuskačného modulu, na druhej strane tiež rieši vytvorenie, správu, prípadne ukončenie obslužného vlákna spracovávaných paketov. Stručne si popíšeme nosné metódy tejto triedy:

- `start` - Verejná metóda, ktorá je ale čisto *virtuálnou*, a teda vlastnú implementáciu je potrebné definovať v konkrétnom obfuskačnom module. Metóda by mala zabezpečiť vytvorenie virtuálnej fronty paketov, pre ktorú definuje odpovedajúcu *callback funkciu*. Callback funkcia je spustená vždy, keď sa vo fronte objaví nový paket a slúži na jeho obsluhu. Z tohto dôvodu je potreba riešiť implementáciu tejto metódy a callback funkcie jednotlivo pre každý samostatný obfuskačný modul, pretože callback funkcie sa budú líšiť vzhľadom na inú formu obfuskácie.
- `stop` - Pomocná verejná metóda, riešiacia synchronizáciu obslužného vlákna pri nutnosti jeho ukončenia, čím sa ukončí činnosť celého obfuskačného modulu.
- `initQueue` - Pomocná metóda vykonávajúca prvotnú inicializáciu virtuálnej fronty paketov. Metódu je potrebné použiť pri implementácii metódy `start` v odvodenej triede pred registrovaním callback funkcie daného modulu. Metóda zabezpečí korektné vytvorenie a naviazanie fronty na systémové štruktúry.
- `createQueue` - Doplňujúca metóda k `initQueue`, ktorá vykonáva dokončenie inicializácie virtuálnej fronty paketov a jej finálne vytvorenie. Metódu je nutné použiť až po registrácii callback funkcie, pretože táto hodnota je pri vytváraní požadovaná. Metóda tiež vytvorí nové obslužné vlákno programu, ktoré bude spracúvať novo vytvorenú frontu.
- `startThread` - Metóda slúžiaca na spustenie nového vlákna v danom obfuskačnom module. Metóda zabezpečuje, aby mal v prípade potreby každý objekt triedy svoje vlastné obslužné vlákno.



Obrázok 3.3: Diagram tried obfuskačnej knižnice

- **handlePackets** - Metóda predstavujúca obslužnú funkciu paketu, keď vlastná obsluha je vykonávaná spomínaným novo vytvoreným vláknom. Vďaka použitiu volania nového vlákna pomocou metódy **startThread**, predstavuje táto funkcia implementáciu funkčnosti celého vlákna, teda jedná sa o spracovanie paketu, ktorý vyhovuje obfuskačným kritériám a nachádza sa vo virtuálnej fronte paketov.

Ďalšie metódy triedy `Obfuscator` majú prevažne pomocný charakter, keď metódy `printError` a `printWarning` zabezpečujú jednotný výpis chybových hlásení a varovaní pre danú triedu. Metóda `cleanIptables` zjednodušuje uvoľnenie používaných zdrojov modulu a pri ukončení jeho behu zabezpečí navrátenie systémových štruktúr hostujúceho operačného systému do pôvodného stavu.

3.3.3 Triedy *ProtocolObfuscator* a *PortScanObfuscator*

Triedy `protocolObfuscator` a `portScanObfuscator` predstavujú konkrétnu implementáciu obfuskačných modulov, keď prvá menovaná trieda implementuje maskovanie komunikácie zvoleným protokolom, druhá rieši maskovanie kompletného skenovania portov cieľovej stanice. Obe triedy vychádzajú z triedy `Obfuscator`, od ktorej dedia spoločné metódy a teda aj jej rozhranie.

Triedy sa špecializujú na implementáciu špecifických *callback funkcií*, ktoré sú vytvorené vzhľadom na požiadavky daného obfuskačného modulu. K vyvolaniu callback funkcie dôjde v prípade, ak metóda rodičovskej triedy `handlePackets` zachytí dátový paket v spravovanej virtuálnej fronte. Úlohou callback funkcie je analyzovať tento paket a na základe obfuskačných kritérií, buď zaslať paket do *vstupno-výstupného modulu*, ktorý zabezpečí zaslanie v transformovanej podobe do proxy modulu, prípadne ak paket nevyhovuje kritériám, je zaslaný v nezmenenej podobe na pôvodné miesto určenia.

Metódy sú spoločné pre obe popisované triedy, pretože triedy sú odvodené z nadradenej triedy `Obfuscator`, je zrejmé, že musia implementovať zdedenú metódu `start`, kde sa vykoná registrácia callback funkcie do virtuálnej fronty, pre daný typ riešenej obfuskačnej komunikácie. Špecifickou metódou je `iptablesInitComands`, ktorá slúži na počiatočnú inicializáciu a vytvorenie príkazov, ktoré budú používané pri odchyťovaní odchádzajúcej komunikácie. Pravidlá sú špecifické pre jednotlivé typy obfuskačnej komunikácie, a preto každý modul musí implementovať túto metódu vo vlastnej rézii.

3.3.4 Trieda *InputOutputForwarder*

Trieda `InputOutputForwarder` zastrešuje riešenie komunikácie s proxy modulom, keď vykonáva odosielanie paketov, ktorého súčasťou je aj ich zakódovanie do obfuskovanej podoby. Trieda na druhej strane tiež zabezpečuje prijímanie dát v smere od proxy modulu, ich spätné dekodovanie a spracovanie, ktoré zahŕňa vloženie paketu obsahujúceho dáta odpovede pre danú komunikáciu do hostiteľského systému tak, aby komunikujúca aplikácia nebola schopná poznať, že komunikácia bola obfuskovávaná.

Implementácia tejto triedy je významná z hľadiska zachovania *transparentnosti knižnice* voči sieťovej komunikácii, ktorú obfuskuje. Zároveň implementuje samotné obfuskačné metódy, ktorými sa transformujú zasielané dáta. V súčasnej verzii sú použité metódy založené na *maskovaní protokolu* (2.1.3) a *kryptografické metódy* (2.1.1) postavené na šifrovaní dát. Podrobnejšie sa na detaily komunikácie zameriame v kapitole 3.7.

Signifikantné metódy tejto triedy si stručne popíšeme:

- `syncWithProxy` - Metóda slúžiaca na vytvorenie spojenia s proxy modulom na najnižšej úrovni, zahŕňajúca prácu so sieťovými socketmi, autentizáciu proxy modulu a podobne. Na základe parametrov obfuskačnej komunikácie sa vytvorí buď *šifrované* alebo *nešifrované* spojenie, ktoré bude spravované touto triedou.
- `disconnect` - Metóda zabezpečujúca prerušenie komunikácie s proxy modulom, v prípade ak je potrebné ukončiť obfuskačný proces. Po zavolaní tejto metódy teda dôjde

ku korektnému ukončeniu spravovanej komunikácie, ktoré zahŕňa tiež výmenu synchronizačných správ medzi oboma stranami.

- **forwardPacketToOutput** - Nosná metóda triedy, ktorá slúži na zaslanie paketu proxy modulu, využíva sa najmä obfuskáčnymi modulmi pri spracovávaní paketu. Po predaní dát, ktoré boli zachytené a sú určené na obfuskáciu, metóda vykoná ich korektné zaslanie do proxy modulu. Samozrejmosťou je zamaskovanie týchto dát pomocou ďalších transformačných funkcií.
- **handleInputPackets** - Významná funkcia, ktorá implementuje spracúvanie prichádzajúcich paketov od proxy modulu. Jej úlohou je prijaté dáta rozložiť na časti predstavujúce jednotlivé pakety, pretože v prípade komunikácie s proxy modulom môže dochádzať k rôznej *fragmentácii* a *zjednocovaniu*. Metóda teda získava obfuskované dáta jednotlivých prenášaných paketov a posúva ich na ďalšie spracovanie metóde **procesInput**. Implementácia tejto obsluhy je riešená samostatným vláknom programu, aby bolo možné spracovávať prichádzajúce pakety nezávisle na odosielaných dátach.
- **procesInput** - Na základe parametrov obfuskáčného procesu dekoduje dáta paketu, vytvorí teda spätnú transformáciu dát, zrekonštruuje z nich nový paket, ktorý bude vložený na spracovanie operačnému systému ako štandardný prijatý paket zo sieťového rozhrania. Správne zostrojenie tohto paketu je pomerne dôležité na zachovanie transparentnosti práce knižnice v hostiteľskom systéme.
- **createObfuscatedData**, **decodeObfuscatedData** - Podporné metódy, využívané predošlými popísanými metódami, ktoré slúžia na vytvorenie obfuskovanej podoby z dát, prípadne ich pôvodnej podoby z obfuskovaných dát. Metóda využíva údaje o parametroch obfuskácie na vytvorenie transformácie, ktorá spĺňa zadané požiadavky. Podrobnejšie sa na vykonávanie samotnej transformácie dát zameriame v odpovedajúcej kapitole 3.7.
- **intToHexString**, **hexStringToInt** - Predstavujú pomocné metódy, ktoré prevádzajú dáta z číselnej reprezentácie na reprezentáciu reťazcom a späť. Metódy sú používané v prípade vykonávania *nešifrovanej* obfuskácie, keď je v niektorých prípadoch potrebné prenášať paketové dáta v otvorenej textovej podobe.
- **decodeHashData** - Podporná metóda zabezpečujúca dekodovanie prijatých autentizačných dát zaslaných od proxy modulu do podoby, aby mohli byť porovnané s požadovanou hodnotou, ktorá je vyžadovaná na korektnú autentizáciu.
- **initSSL**, **cleanSSL** - Metódy slúžiace na jednotnú inicializáciu a uvoľnenie zdrojov, ktoré sú nevyhnutné na ustanovenie *šifrovanej* komunikácie medzi oboma stranami. K inicializácii spojenia šifrovane dôjde iba na základe *explicitného* nastavenia v konfiguračnom súbore, inak komunikácia prebieha v otvorenej podobe.

Ďalšie metódy tejto triedy majú podporný charakter, keď **startThread** vykonáva obdobnú funkciu ako rovnomenná metóda v triede **Obfuscator**. Zvyšné metódy implementované v tejto triede potom slúžia iba na zjednotenie výpisu chybových hlásení a varovaní, tak ako bolo spomenuté aj pri predchádzajúcich moduloch.

3.3.5 Ďalšie triedy obfuskačnej knižnice

Implementácia zvyšných tried nepredstavuje až takú významnú súčasť vytváranej knižnice, pretože slúžia v prípade `SettingsData` na jednoduchšie predávanie parametrov obfuskačie medzi jednotlivými modulmi, ktoré tieto dáta vyžadujú k svojej činnosti. Výhodou samostatnej triedy uchovávajúcej tieto dáta je, že v programe stačí vytvorenie jedinej inštancie v riadiacom module, a potom je odkaz na tento objekt predávaný zvyšným modulom.

Druhú podpornú triedu tvorí implementácia `PacketParser`, ktorá sa stará o spracovávanie dátového paketu. Trieda poskytuje vo svojom rozhraní množstvo metód slúžiacich na získanie väčšiny možných dát z hlavičiek a dátovej časti paketu. Trieda taktiež kontroluje validitu paketu, tak aby nedošlo k spracovávaniu prijatého poškodeného paketu. Implementácia tejto triedy je využívaná vo všetkých moduloch, ktoré pracujú priamo so sieťovými dátami. Výhodou je, že ostatné moduly nemusia riešiť analýzu všetkých hlavičiek v spracovávanom pakete, ale všetky potrebné údaje môžu získať s využitím tejto triedy.

V tejto sekcii sme popísali základnú štruktúru obfuskačnej knižnice, keď boli definované princípy práce jednotlivých modulov a vzájomnej interakcie medzi nimi. Na podrobnosti riešenia význačných častí sa podrobnejšie zameriame v odpovedajúcich kapitolách v ďalšom texte. Táto kapitola mala poskytnúť iba potrebný základ na rozvinutie riešenej problematiky, z tohto dôvodu sa nezameriavala podrobne na všetky detaily, ktoré je možné dohľadať podrobnejšie v projektovej dokumentácii, dostupnej na priloženom CD nosiči.

3.4 Štruktúra proxy modulu

Proxy modul, ktorý bude pri obfuskačii distribuovaný v cieľovej sieti, je dôležitým prvkom navrhovaného systému. Hlavnou úlohou proxy modulu je prijatie obfuskovanej komunikácie, jej dekodovanie, rekonštrukcia pôvodných paketov a ich ďalšia distribúcia v modifikovanej forme do vnútornej siete. Sekundárnou úlohou je odchytať pakety obsahujúce odpoveď na predošlú komunikáciu, rozoslanú v danej sieti, jej následnú transformáciu do obfuskovanej podoby a zaslanie späť druhej strane využívajúcej knižnicu.

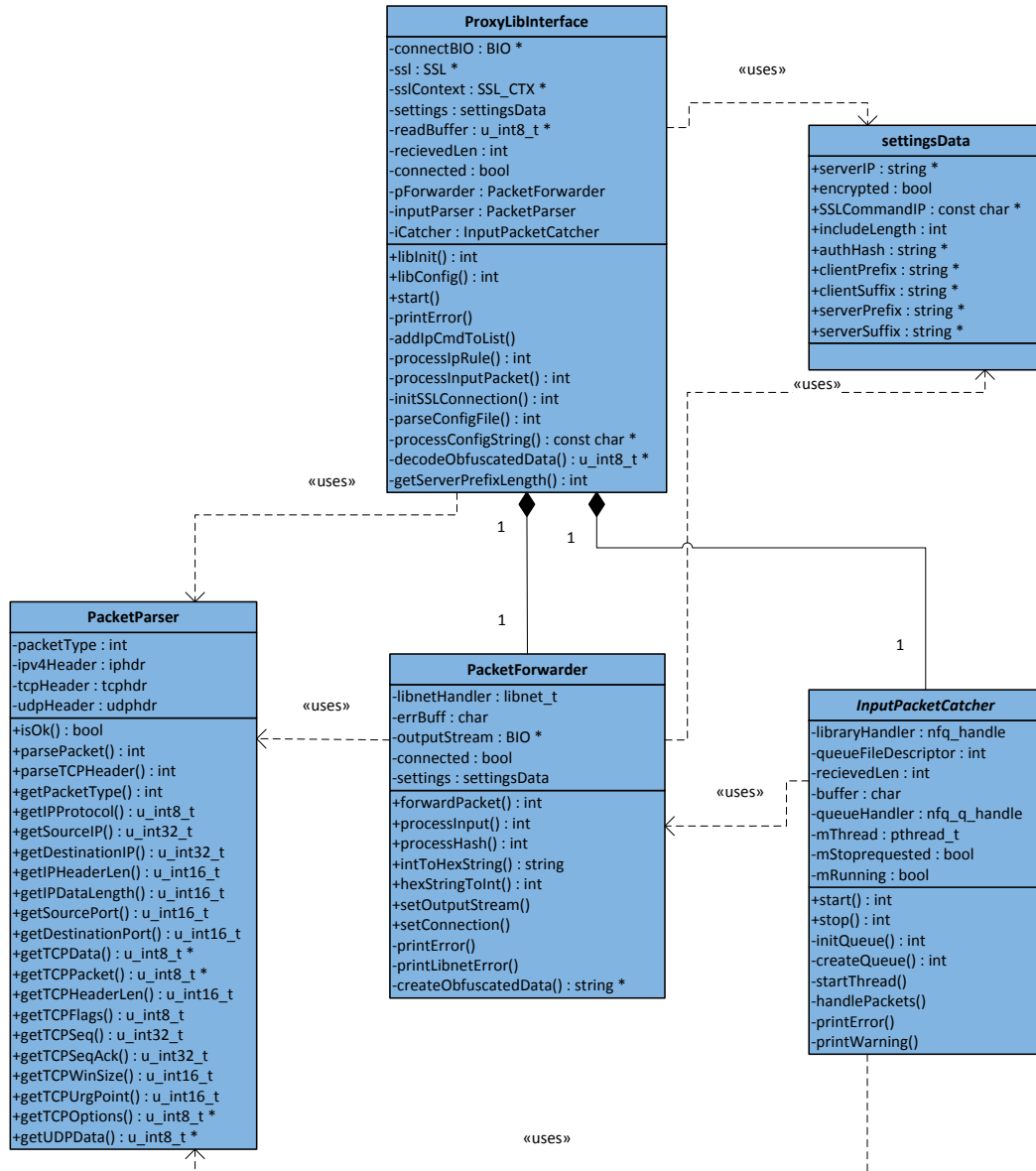
Dôraz pri návrhu tejto komponenty je kladený na jej univerzálnosť, a teda schopnosť pracovať s obfuskoványm tokom bez ohľadu na to, aký obfuskačný modul je zvolený na strane užívateľa s knižnicou. Abstraktný model proxy modulu, popísaný diagramom tried, môžeme vidieť na obrázku 3.4. Pomocné triedy, ako `PacketParser` a `SettingsData`, sú vo väčšej miere podobné, či už do štruktúry alebo funkcionality, ako ich implementácia vo vlastnej knižnici (3.3.5), z tohto dôvodu sa zameriame v ďalších podkapitolách iba stručne na popis unikátnych, dosiaľ nepopísaných tried proxy modulu.

3.4.1 Trieda *ProxyLibInterface*

Funkcionalita proxy modulu je implementovaná ako knižnica, ktorú je možné využiť v riadiacom programe, pričom trieda `ProxyLibInterface` tvorí jej jediné rozhranie. Trieda je zameraná na implementáciu riadenia, teda zastrešuje správu a komunikáciu medzi jednotlivými podriadenými komponentmi systému. Trieda taktiež rieši inicializáciu, vytvorenie, autentizáciu a synchronizáciu s užívateľským programom využívajúcim knižnicu.

Jednotlivé metódy si stručne popíšeme:

- **libInit** - Zabezpečuje počiatočnú inicializáciu proxy modulu, keď sa vykonajú rutinné operácie slúžiace na podporu správnej funkcie knižnice - napríklad nastavenie obsluhy signálov a podobne.
- **libConfig** - Slúži na konfiguráciu proxy modulu, ktorá prebieha pomocou *konfiguračného súboru*, popísaného podrobnejšie v samostatnej kapitole 3.8. Konfiguračný



Obrázok 3.4: Diagram tried proxy modulu

súbor slúži najmä na získanie informácií o nastavení obfuskačnej knižnice, pretože bez znalosti potrebných vlastností obfuskačie by nebolo možné dekódovať prichádzajúci dátový tok.

- **start** - Posledná z verejných metód rozhrania tejto triedy, po ktorej zavolaní sa spustí beh daného programu. Základom je nadviazanie korektnej komunikácie, doplnenej o autentizáciu s druhou komunikujúcou stranou a následné čakanie na prichádzajúce dáta, keď metóda zabezpečuje aj čiastočnú obsluhu týchto prichádzajúcich dát. Prichádzajúce dáta obsahujúce jednotlivé pakety môžu byť rôzne *pospájané* alebo *fragmentované*, preto je potrebné zabezpečiť ich správne rozdelenie na samostatné pakety. Obsluha paketu je následne predaná ďalším modulom.
- **processInputPacket** - Nosná metóda triedy, umožňujúca dokončiť obsluhu prichádzajúceho paketu, ktorý bol predspracovaný už v metóde **start**. Metóda vykoná dekódovanie paketu z obfuskovanej podoby a na základe údajov v pakete sa vytvoria nové pravidlá na odchyťvanie paketov s prípadnou odpoveďou. V záverečnej fáze sa paket predá na odoslanie do modulu riadiaceho komunikáciu vo vnútornej sieti, konkrétne implementovaného v triede **PacketForwarder**.
- **addIpCmdToList**, **processIpRule** - Pomocné metódy riešiace vytváranie a spracovanie nových pravidiel, slúžiacich na odchyťvanie komunikácie, ktorá obsahuje odpovede na pôvodnú, distribuovanú komunikáciu. Pravidlá sú vytvárané *dynamicky*, za behu, na základe charakteristiky prichádzajúcej komunikácie od užívateľa.
- **parseConfigFile**, **processConfigString** - Doplnkové metódy, slúžiace na jednoduchšie spracovanie konfiguračného súboru, keď zabezpečia extrakciu nastavení, ktoré sú potrebné na korektnú komunikáciu medzi oboma participujúcimi stranami.
- **decodeObfuscatedData**, **getServerPrefixLength** - Podporné metódy poskytujúce funkcionality potrebnú na dekódovanie obfuskovaného paketu, prípadne zistenie dĺžky hlavičky prichádzajúceho paketu.

Ostatné metódy triedy **ProxyLibInterface** nepredstavujú nosnú funkcionality tejto triedy a sú využívané najmä na jednoduchší výpis chybových hlásení modulu alebo, v prípade metódy **initSSLConnection**, na inicializáciu zabezpečeného spojenia, ak je požadované vytvoriť šifrovaný komunikačný kanál medzi komunikujúcimi entitami.

3.4.2 Trieda *PacketForwarder*

Implementácia triedy **PacketForwarder** zastrešuje riadenie odchádzajúcej komunikácie vo všetkých smeroch z proxy modulu, čo zahŕňa odosielanie paketov do vnútornej siete a na druhej strane tiež odosielanie dát späť užívateľovi. Z tohto dôvodu vzniká nutnosť implementovať obsluhu sieťových soketov pomocou samostatných vláken programu.

Stručne si popíšeme nosné metódy modulu:

- **forwardPacket** - Metóda slúžiaca na zaslanie paketu prijatého od užívateľa obfuskačnej knižnice do vnútornej siete. Pri odosielaní dôjde k *podvrhnutiu* IP adresy, ktorá sa nahradí za IP adresu proxy modulu tak, aby bol proxy modul neskôr schopný odchyť prípadnú odpoveď.

- `processInput` - Poskytuje opačnú funkcionálnosť k predošlej popísanej metóde, keď umožňuje zaslanie odchytenej odpovede z vnútornej siete, ktorá je určená pre užívateľa. Pri odosielaní sú dáta samozrejme transformované do obfuskovanej podoby.
- `processHash` - Pomocná metóda slúžiaca na odoslanie autentizačných informácií z proxy modulu užívateľovi. Autentizačné dáta sa od paketových čiastočne odlišujú, preto je nutná samostatná implementácia, ktorá je využívaná pri procese zostavovania komunikácie.
- `intToHexString`, `hexStringToInt` - Pomocné metódy, ktoré prevádzajú dáta z číselnej reprezentácie na reprezentáciu reťazcom a späť. Funkcionálnosť je obdobná s rovnomenými metódami popísanými v kapitole 3.3.4.
- `createObfuscatedData` - Metóda vykonávajúca transformáciu odosielaných dát smerom k užívateľovi, riadiaca sa parametrami zvolenej obfuskácie. Používa sa najmä pri vykonávaní nosnej funkcie `processInput`, na vykonanie potrebnej modifikácie dát, ktoré sa budú odosielať.

Ďalšie metódy, tak ako v predošlých prípadoch, vykonávajú doplnkové funkcie, ktoré slúžia na zvýšenie prehľadnosti implementovanej triedy, teda napríklad na jednotný výpis chybových hlásení, prípadne ďalšie metódy slúžiace na definovanie aktuálnych parametrov komunikácie, ktorými môže nadradený modul špecifikovať charakteristiky spojenia.

3.4.3 Trieda *InputPacketCatcher*

Poslednú popisovanú triedu predstavuje trieda `InputPacketCatcher`, ktorá zabezpečuje odchytyvanie paketov s odpoveďou na predtým rozoslané dáta. Úlohou triedy je zabezpečiť funkčnosť tak, aby nemuselo dochádzať k dynamickému otváraniu portov na hostiteľskom stroji proxy modulu, pretože nadväzovanie väčšieho množstva TCP spojení by mohlo, pri dátovom toku smerovaného na veľa portov (napríklad skenovanie portov), značne vyťažiť proxy modul. Z tohto dôvodu je použitý prístup, pri ktorom sú pakety odchytené ešte pred samotným spracovaním hostiteľským *operačným systémom*, a teda aj keď nie je port na danom stroji otvorený, systém *nebude* zasielať ICMP odpoveď o nedostupnom porte, keďže paket zostane pred ním utajený.

Štruktúra triedy je postavená na modeli podobnom už popisovanej triede `Obfuscator`, keď sú implementované metódy na spustenie a ukončenie práce a podporné metódy na správu obslužného vlákna programu. Základ odchytyvania paketov je založený na vytvorení *virtuálnej fronty*, do ktorej budú presmerované prichádzajúce pakety, spadajúce pod kritéria vytvorené pri dekódovaní a zasielaní pôvodného paketu prijatého v riadiacom module `ProxyLibInterface`. Pakety v tejto fronte sú následne spracovávané v *callback funkcii*, ktorá je volaná samostatne pre každý prichádzajúci paket. Dáta z paketu sú pomocou komunikačného modulu `PacketForwarder` odoslané v transformovanej podobe späť užívateľovi, kým samotný odchytený paket je *zahodený*, aby nedošlo k jeho spracovaniu *operačným systémom*.

Tento prístup bol zvolený hlavne z dôvodu efektívneho spracovania, ktoré je schopné zvládnuť spracovávať aj náročnejšie dátové toky, bez prílišného dopadu na rýchlosť prenosu medzi koncovými uzlami. Taktiež výhodou je, že na hostiteľskom počítači nie je možné odhaliť žiadne otvorené porty, čo umožňuje v prípade potreby lepšie zamaskovanie modulu na hostiteľskom počítači.

V tejto kapitole sme sa zamerali na stručný popis implementovaných modulov, ktoré umožňujú zamaskovanie prebiehajúcej komunikácie, prípadne zamaskovanie nejakého útoku v sieti. Ako vyplýva z popisu, dáta sú prenášané transformovane hlavne pri prechode na rozhraní vonkajšej a vnútornej siete, kde sa predpokladá prítomnosť IDS systému, následne vo vnútornej sieti dochádza k distribúcii dát v pôvodnej forme. Tento prístup je možné použiť, pretože IDS systém *nekontroluje* pakety vo svojej vnútornej, dôveryhodnej sieťovej doméne.

3.5 Nástroje použité pri implementácii

Na vlastné riešenie implementácie knižnice a proxy modulu bol použitý jazyk C++ s tým, že okrem štandardných knižníc boli použité aj niektoré neštandardné nástroje. Hlavným dôvodom ich použitia je čiastočné zjednodušenie prístupu k niektorým nízkoúrovňovým súčastiam hostiteľského operačného systému, pretože, ako bolo spomenuté, určité moduly musia pracovať priamo na úrovni operačného systému (napríklad práca s virtuálnymi frontami paketov, prípadne injektovanie paketov priamo na sieťové rozhranie). Jednotlivé použité knižnice si stručne popíšeme v nasledujúcich podsekciiach.

3.5.1 Knižnica na prácu s paketmi - *Libnet*

Knižnica *Libnet*¹ je multiplatformové API [26], implementované v jazyku C, ktoré poskytuje rozhranie na vyššej úrovni k nízkoúrovňovým sieťovým funkciám. Jedná sa o pomerne robustný nástroj, využívaný pri pokročilejšej práci so sieťovými dátami. Pre svoje možnosti plnej kontroly nad jednotlivými sieťovými paketmi, jednoduchému rozhraniu na ich vytváranie a injektovanie, patrí táto knižnica k obľúbeným nástrojom pri vývoji sieťových aplikácii zameraných na bezpečnosť, prípadne na druhej strane na vytváranie nových exploitov. Výhodou použitia tejto knižnice je jej rozšírenosť na rôzne operačné systémy, čo by malo zabezpečiť väčšiu prenositeľnosť výslednej aplikácie.

Pre potreby implementácie boli použité iba niektoré základné funkcie z tejto knižnice, najmä metódy zabezpečujúce vytváranie, zasielanie a injektovanie paketov priamo na rozhranie sieťovej karty, konkrétne sú využité v triedach, ktoré riešia rekonštrukciu paketov z obfuskovanej podoby a ich ďalšie zasielanie zo systému, prípadne v opačnom smere do systému. Konkrétne je teda knižnica *Libnet* použitá hlavne v triede `InputOutputForwarder` v architektúre knižnice a obdobnej triede `PacketForwarder` u proxy modulu.

V oboch prípadoch slúžia prostriedky knižnice *Libnet* na jednoduchšie vytváranie paketov, pretože v množstve prípadov je potrebné na základe získaných dát vytvoriť nový paket, prípadne v ňom pozmeniť niektoré údaje. Z toho vyplýva mnohokrát potreba výpočtu nového *kontrolného súčtu*, ktorý knižnica zabezpečí. V neposlednom rade je pomocou knižnice riešené aj odosielanie niektorých paketov, alebo, na druhej strane, opätovné vloženie pozmeneného paketu do hostiteľského operačného systému. Na tieto účely sa používa takzvaný RAW prístup, keď dochádza k obídeniu TCP/IP zapuzdrenia, čo nám teda umožní pracovať priamo na *linkovej vrstve* daného hostiteľského systému.

Uvedenú funkcionalitu by bolo možné jednoducho implementovať aj pomocou štandardných BSD soкетов, ale v prípade ich použitia by bolo riešenie orientované priamo na jeden daný operačný systém, pretože v implementácii pre rôzne distribúcie dochádza k drobným odlišnostiam. Tieto rozdiely knižnica *Libnet* vyrovnáva, a tým umožňuje vytváranie prenositeľnejších aplikácii.

¹<http://libnet.sourceforge.net/>

3.5.2 Nástroje jadra systému Linux - *netfilter* a *iptables*

Netfilter [16] predstavuje framework zameraný na filtrovanie paketov, ktorý je priamo súčasťou jadra operačného systému *Linux* už od verzie 2.4.x (približne od roku 2001). Nad rozhraním tohto frameworku je postavené aj známe riešenie užívateľského firewall-u *iptables*, ktoré vytvára jednoduché rozhranie pre prístup k pravidlám filtrovania paketov v jadre operačného systému. Na základe rôznych typov pravidiel filtrovania je možné efektívne riadiť dátové toky, smerované cez daný systém.

Ako rozšírenie čistej implementácie paketového filtra v jadre, bola použitá knižnica `libnetfilter_queue`², ktorá poskytuje užívateľské rozhranie na prácu s virtuálnymi frontami, do ktorých sú zoradované pakety spracovávané paketovým filtrom jadra systému. Knižnica teda umožňuje čiastočne presunúť toto spracovanie zo *systémového* (jadrového) režimu do *užívateľského* režimu. Použitie tohto prístupu umožňuje vytvorenie bezpečnejšej a prenositeľnejšej aplikácie, ako v prípade kompletnej vlastnej implementácie obsluhy paketov v systémovej režime operačného systému. Túto knižnicu je možné použiť vo všetkých jadrách operačného systému *Linux* od verzie 2.6.x a vyššej (všetky verzie vydávané približne od roku 2005), ktoré sú v súčasnosti majoritne rozšírené, a tým nijako neobmedzujú použiteľnosť výslednej implementácie.

Funkcionalita frameworku *netfilter* je postavená na zabezpečení registrácie užívateľskej callback funkcie do *kernel modulov*, ktoré sú potom volané pri spracovávaní jednotlivých paketov, pri splnení určitých zadaných podmienok. V našom prípade si najskôr vytvoríme pomocné virtuálne fronty v jadre systému, pre ktoré zaregistrujeme vytvorené callback funkcie, ktoré budú volané pre každý paket v danej fronte a na základe zvoleného typu obfuskácie dôjde k potrebnej transformácii a ďalšiemu spracovaniu tohto paketu. Pakety do týchto virtuálnych front presmerujeme pomocou pravidiel, ktoré sa dynamicky vkladajú do užívateľského firewallu *iptables*. Týmto sa zabezpečí, že kontrola nad obfuskovaným tokom sa môže dynamicky meniť na základe aktuálnej komunikácie.

Výhodou tohto riešenia je, že k spracovávaní paketov dochádza už na *linkovej vrstve*, kde dôjde k odchyteniu a analýze paketovým filtrom, ešte pred spracovaním vyšších vrstiev pomocou obslužných rutín jadra systému. Využitie tohto prístupu nám napríklad umožní modifikáciu prichádzajúcich obfuskovaných paketov tak, aby sa zabezpečila transparentnosť voči operačnému systému a teda došlo k dekódovaniu a spätnej transformácii ešte pred samotným spracovaním.

Callback funkciu, ako bolo spomenuté v kapitole 3.3.2, musí implementovať každá trieda realizujúca obfuskáčny modul, funkcia je následne zaregistrovaná na obsluhu virtuálnej fronty, do ktorej budú smerované odpovedajúce pakety pre daný typ obfuskácie. V prípade proxy modulu sa používa tento prístup v prípade odchyťovania odpovedí z vnútornej siete, keď sa pred odoslaním pôvodného paketu vytvorí a zavedie odpovedajúce pravidlo, ktoré presmeruje prípadnú odpoveď na daný paket do virtuálnej fronty. Týmto taktiež zabezpečíme, že na strane proxy modulu nie je potrebné otvárať nové komunikačné porty, čím sa čiastočne zabezpečí aj vyššia schopnosť maskovania proxy modulu vo vnútornej sieti.

3.5.3 Šifrovacia knižnica *OpenSSL*

Poslednou používanou knižnicou je *OpenSSL* [5], ktorá ponúka sadu nástrojov na jednoduchšiu prácu pri komunikácii bezpečnými protokolmi SSL (*Secure Socket Layer*) a TLS (*Transport Layer Security*) [27]. Jedná sa o kryptografické protokoly, ktoré sú situované

²http://www.netfilter.org/projects/libnetfilter_queue/index.html

do samostatnej vrstvy, ktorú vkladajú do prenášaných dát pred aplikačnú vrstvu, ktorá sa týmto krokom zabezpečí šifrovaním. Oba protokoly pracujú na rovnakom princípe, keďže protokol TLS je iba novšou verziou svojho predchodcu SSL s malým množstvom zmien. V našom prípade sa použitý protokol zvolí *automaticky* na základe podpory hostiteľských operačných systémov u oboch komunikujúcich entít. Okrem spomenutých funkcií ponúka tento nástroj aj súbor rôznych kryptografických algoritmov a nástrojov na prácu s certifikátmi podľa štandardu *X.509*, ktoré sú pri šifrovaní komunikácie tiež čiastočne použité.

Knižnica je využívaná v prvom rade pri riešení komunikácie medzi užívateľským programom a proxy modulom, ktorá môže prebiehať či už v šifrovanej alebo nešifrovanej podobe. *OpenSSL* ponúka jednoduché rozhranie, ktoré zapuzdruje prácu so sieťovými soketmi (pomocou BIO rozhrania - *Buffered Input/Output*) tak, že je možné jednoducho pridať aj podporu šifrovania správ, prípadne komunikovať nešifrovane. Režim komunikácie je zvolený na základe užívateľovej konfigurácie, keď je potrebné, na základe charakteru vykonávanej obfuskácie, zvoliť v akej forme bude komunikácia prenášaná.

Nástroje *OpenSSL* sú tiež použité na zjednodušenie vygenerovania *certifikátu s verejným kľúčom*, ktorý bude pri komunikácii použitý. Certifikát je potrebné vygenerovať na strane užívateľského programu, ku ktorému sa proxy modul bude snažiť pripojiť. Keďže kľúč certifikátu bude využívaný iba na šifrovanie, pričom autentizácia proxy modulu bude riešená vo vlastnej rézii v kapitole 3.8, je možné podpísať vytváraný certifikát už pri jeho vydávaní, preto hovoríme o takzvanom *self-signed* certifikáte, ktorý však pre naše potreby dostatočne.

V tejto kapitole sme si stručne popísali nástroje, ktoré boli využité pri implementácii vlastného riešenia obfuskačnej knižnice, pričom okrem týchto spomenutých nástrojov boli použité iba štandardné súčasti jazyka C++. Popis jednotlivých nástrojov bol uvedený v stručnej teoretickej rovine, pretože detaily fungovania daných knižníc sú značne rozsiahle a nad rámec tejto práce.

3.6 Odchyťavanie paketov dátových tokov

Zabezpečenie odchyťavania paketov na nižšej systémovej úrovni je dôležitou súčasťou vytvoreného nástroja. Správna selekcia a obsluha paketov, odchytených z *vstupno-výstupných front* hostiteľského systému, ktoré spadajú do určitého komunikačného toku, je základom vytvorenia knižnice, pracujúcej *transparentne* voči vyšším vrstvám daného operačného systému.

Princíp spočíva v tom, že na základe konfigurácie modulov knižnice, dochádza k *dynamickému* pridávaniu a odoberaniu pravidiel pre systémový *firewall iptables*, ktorý bol podrobnejšie popísaný v predošlej kapitole 3.5.2. Implementácia knižnice musí byť na základe týchto požiadaviek schopná analyzovať aktuálny používaný dátový tok užívateľom, pričom na základe charakteru jeho komunikácie generuje a pridáva nové pravidlá, prípadne pridané pravidlá zo systému odoberá.

Hlavnou úlohou, ktorú sa týmto prístupom snažíme dosiahnuť je, aby sa do virtuálnych front, ktoré sú kontrolované callback funkciami obfuskačných modulov dostalo čo najmenej paketov, ktoré nespádajú do obuskačných kritérií. Toto je dôležité najmä z hľadiska výkonnosti, pretože zbytočné presmerovanie neodpovedajúcich paketov zaťažuje, či už samotný operačný systém, ale najmä obfuskačnú knižnicu, ktorá by následne do komunikácie mohla zavádzať vyššie latencie. Pri generovaní a vkladaní pravidiel je z tohto dôvodu nutné zabezpečiť, aby boli pravidlá čo najviac konkrétne, a teda došlo k odchyteniu čo najmenej množiny, do obfuskácie nespádajúcich paketov.

3.6.1 Práca s pravidlami firewallu

Pridávanie a odoberanie pravidiel v knižnici prebieha pomocou štandardného rozhrania nástroja *iptables*, ktoré poskytuje samotný hostiteľský systém. Tento prístup sa používa u oboch implementovaných častí, keď u samotnej knižnice dochádza k presmerovaniu odchádzajúcich paketov, na druhej strane, proxy modul používa odchytyvanie prichádzajúcich paketov. Mechanizmus pri prichádzajúcich paketoch zabezpečí, že paket bude presmerovaný ešte pred spracovaním jadrom systému, a tým zabránime reakcii hostiteľského systému na prichádzajúci paket, pretože proxy modul pracuje bez otvárania portov pre komunikáciu po vnútornej sieti. Ak by teda nedošlo k presmerovaniu, systém by začal zasielať ICMP správy o nedostupnosti portu. Presmerovaný paket sa potom vôbec nedostane do systému a bude obslužený modulom knižnice.

Príklad jednoduchého pravidla môžeme vidieť v nasledujúcom výpise:

```
iptables -A INPUT -p tcp --source 192.168.0.2
          --sport 57481 -j NFQUEUE --queue-num 0
```

kde je špecifikovaný základný príkaz, ktorý pridáva pravidlo orientované na prichádzajúce pakety. Aby bol paket presmerovaný, musí komunikovať TCP protokolom, kde je špecifikovaná zadaná IP adresa a zdrojový port, pričom zvyšná časť príkazu s definovanou hodnotou *NFQUEUE* určuje, že paket bude presmerovaný do *virtuálnej fronty* riadenej jadrom knižnice *netfilter*. Na úspešné pridanie pravidla je ešte nutné definovať číslo virtuálnej fronty, do ktorej bude paket zaradený, keď táto hodnota je určená na základe smeru a typu komunikačného toku.

Odoberanie paketov prebieha obdobným spôsobom, keď je potrebné presne špecifikovať pravidlo, ktoré sa bude z aktuálneho zoznamu používaných pravidiel odoberať. Na tieto účely je potrebné v moduloch uchovávať aktuálne databázy vložených pravidiel pre jednotlivé fronty, aby bolo možné pravidlo jednoznačne identifikovať v prípade, že pre ďalší beh obfuskácie už nie je potrebné. Databáza pravidiel musí byť uchovávaná aj pre prípad ukončenia behu knižnice, keď je vhodné, vložené pravidlá odobrať, aby sme hostiteľský operačný systém navrátili do stavu, v ktorom bol pred spustením knižnice.

3.7 Realizácia obfuskovanej komunikácie

V nasledujúcej kapitole sa podrobnejšie zameriame na riešenie spôsobu komunikácie medzi užívateľským programom a proxy modulom, ktorú je potrebné vykonávať v obfuskovanej podobe. Návrh vlastného riešenia kládol dôraz na jednoduchú rozšíriteľnosť knižnice, z tohto dôvodu bolo jednoducho možné experimentovať s rôznymi prístupmi transformácie prenášanej komunikácie medzi participujúcimi entitami. Pri vývoji výslednej aplikácie boli implementované dva základné prístupy riešenia, keď počiatočný jednoduchší princíp bol neskôr nahradený za efektívnejšie a robustnejšie riešenie. Jednotlivé etapy vývoja si podrobnejšie priblížime v nasledujúcich podkapitolách.

3.7.1 Počiatočný prístup

Ako prvý prístup riešenia zadaného obfuskáčného problému bola snaha zamaskovať prebiehajúcu komunikáciu vo forme zašifrovaných UDP paketov. Toto navrhnuté riešenie implementovalo jednoduché maskovanie prenášaných dát tak, aby sa pri ich analýze javili na strane IDS ako bežne prenášané pakety s videom.

Pri tomto riešení vystupoval ako *server* samotný proxy modul, u ktorého bolo potrebné otvoriť načúvajúci UDP port, na ktorý sa zasielali z užívateľského programu transformované pakety. Proxy modul po dekodovaní dát, ich distribúcií vo vnútornej sieti a odchytení odpovede od cieľovej stanice, znova transformoval dáta do UDP paketu, ktoré boli zaslané späť užívateľskému programu. V tomto prípade sa teda vykonávalo len jednoduché šifrovanie a zapuzdrowanie komunikácie do UDP datagramov.

Výsledky získané z testovania tohto prístupu boli napriek jednoduchému riešeniu *úspešné*, keď IDS pre rôzne typy komunikácie nevygenerovalo žiadne varovania ani alarmy. Podrobnosti o spôsoboch testovania sú popísané v odpovedajúcej kapitole 4. Úspešnosť tohto riešenia však závisela čisto na šifrovaní, pretože IDS nemalo žiadnu možnosť pri svojej analýze preskúmať dátovú časť paketu a v prípade UDP datagramu pravdepodobne ani ku kontrole nedochádzalo, pretože pravidlá nie sú na takýto typ komunikácie vytvárané.

Tento prístup k riešeniu komunikácie má na druhej strane aj nevýhody, ktoré eskalovali k nevyužitíu tohto riešenia a vytvoreniu sofistikovanejšieho prístupu komunikácie medzi modulmi. Hlavnou nevýhodou je, že komunikácia bola nadväzovaná z vonkajšej siete, teda pakety boli zasielané priamo na otvorený port proxy modulu. V prípade reálneho nasadenia by s vysokou pravdepodobnosťou *stavový firewall* takéto pakety do vnútornej siete ani neprepustil a celá komunikácia by bola *neúspešná*. Ďalšou nevýhodou je samotná potreba otvoriť načúvací port vo vnútornej sieti, pretože to by viedlo k rýchlemu odhaleniu podozrivého správania, pretože takéto akcie nie sú bežné. Posledným nedostatkom tohto riešenia je aj, že UDP pakety sú zasielané v oboch smeroch, čo by sa pri podrobnejšej analýze mohlo javiť tiež ako anomália, keď zvyčajne je video smerované iba smerom od serveru do vnútra siete užívateľovi.

Na základe týchto získaných znalostí bola následne vytvorená pokročilejšia metóda, ktorá sa viac zameriava, okrem samotnej obfuskácie, aj na zamaskovanie tejto obfuskovanej komunikácie v sieťovom toku, aby sa ani pri podrobnejšej analýze, nejavil daný tok ako potenciálna anomália. Implementovaný prístup si podrobnejšie popíšeme v samostatnej nasledujúcej kapitole.

3.7.2 Výsledné riešenie

Finálne riešenie problému sa snaží oproti predošlému prístupu obfuskovať prebiehajúcu komunikáciu sofistikovanejším spôsobom, keď sa prenášané dáta maskujú do HTTP, prípadne pri potrebe šifrovania do HTTPS protokolu. Tieto protokoly boli zvolené hlavne z dôvodu, že sa jedná o bežné a veľmi rozšírené komunikačné protokoly vo väčšine sietí, a tým sa sťažuje potenciálna pokročilejšia analýza obfuskovaných dát a tiež by nemalo dôjsť k prípadnému zablokovaniu protokolu v sieti, v prípade vzniku podozrenia na nejakú sieťovú anomáliu.

Oproti predošlému prípadu došlo k zmene rolí komunikujúcich entít tak, aby komunikácia čo najviac odpovedala skutočnosti, teda užívateľská časť mimo chránenej siete vystupuje ako *webový server* a proxy modul sa naň snaží pripojiť ako *bežný klient*. Tento prístup by mal zabezpečiť, že zasielaná komunikácia nebude zablokovaná ani v prípade, že chránená sieť bude zabezpečená pomocou sieťového firewallu, pretože komunikácia odchádzajúca na HTTP prípadne HTTPS porty je štandardne povolená a stavový filter tiež zabezpečí aby bola doručená aj odpoveď na zaslanú požiadavku. Takýmto riešením sa teda vo väčšine prípadov zabezpečí garancia, že komunikácia nemôže byť zablokovaná inými súčasťami siete, ktoré sa starajú o bezpečnosť, pretože sa bude pri analýze javiť ako *štandardná komunikácia* napríklad pri prezeraní internetových stránok.

Komunikáciu môžeme stručne popísať tak, že po nadviazaní spojenia medzi jednotlivými komunikujúcimi stranami a autentizácii proxy modulu voči serveru, sú pôvodné prenášané dáta zakódované do paketov, ktoré obsahujú korektné HTTP hlavičky tak, aby výmena správ mala charakter *dotaz - odpoveď*, čo je pre HTTP protokol hlavný spôsob komunikácie. Ukážku zachytenej komunikácie programom *Wireshark*³ (pre prehľadnosť zjednodušená do zobraziteľnej podoby) je možné vidieť na obrázku 3.5. V tomto prípade sú požiadavky zasielané z proxy modulu na server označené *červenou* farbou a odpovede servera značené *zelené*. Komunikácia prebieha priamo, bez synchronizačných správ, pretože počiatočná autentizácia proxy a synchronizácia je prenesená v prvej GET požiadavke zaslanej na server, na ktorú už server odpovedá vlastnými prenášanými dátami.

Dáta prenášané zo serveru späť na proxy, sú transformované do textovej podoby tak, aby mohli byť vložené do tela štandardnej HTTP správy. Z uvedeného obrázka vyplýva, že aj v prípade skúmania odchytených paketov priamo na rozhraní sieťovej karty, sa javí komunikácia ako korektná HTTP prevádzka. Spôsob zasielania správ možno modifikovať konfiguráciou knižnice, teda namiesto metódy GET, keď sa prenášaná informácia vkladá do hlavičky správy, môžeme využiť, na obfuskáciu *efektívnejšiu*, metódu POST. Potom bude na prenos používané telo HTTP paketu v oboch smeroch komunikácie. Metóda GET bola popísaná z dôvodu, aby bol viditeľný spôsob prenosu dát v otvorenej forme, ktorý je možný, či už v hlavičke alebo tele paketu. Príklady konfigurácie pre metódy GET aj POST je možné vidieť v prílohe A.

```
GET
9e004500003c000040004006b966c0a80002c0a800030017854c974ef52fc74fb577a01216a06d070000020405b40402080a0013ab240
000091201030307 HTTP/1.1
HTTP/1.1 200 OK (text/plain)

GET
a600451000405e19400040065b39c0a80002c0a800030017854c974ef530c74fb5778018002e7ac500000101080a0013ab4b00000914f
ffd18fffd20fffd23fffd27 HTTP/1.1
HTTP/1.1 200 OK (text/plain)

GET
8e00451000345e1a400040065b44c0a80002c0a800030017854c974ef53cc74fb5928010002eb1f300000101080a0013ab4d00000914
HTTP/1.1
HTTP/1.1 200 OK (text/plain)
```

Obrázok 3.5: Príklad výmeny správ v nešifrovanej podobe - HTTP

Pre potreby robustnejšej obfuskácie bola implementovaná aj doplňujúca metóda kombinujúca maskovanie protokolu so šifrovaním dát, teda na komunikáciu sa využíva šifrovaný variant HTTP protokolu - HTTPS, keď sú dáta paketov šifrované pomocou SSL, prípadne TLS, ktoré sú popísané v kapitole 3.5.3. Táto metóda zabezpečí nemožnosť analýzy dátovej časti paketu, a tým ku kompletnému utajeniu prebiehajúcej komunikácie. Príklad nadviazania komunikácie a výmenu dátových HTTPS správ s použitím TLS, zachytenej obdobne ako v predošlom prípade programom *Wireshark*, je možné vidieť na obrázku 3.6, pričom správy vyznačené *červenou* farbou sú zasielané z proxy modulu na server, na druhej strane *zelené* predstavujú dáta zasielané v opačnom smere. Prvé dve výmeny správ zabezpečujú inicializáciu komunikácie, keď sú vykonané potrebné rutiny, ako zaslanie *serverového certifikátu*, výmenu šifrovacích kľúčov a nastavenie metód šifrovania. Po vykonaní týchto operácií môže začať vlastná komunikácia, ktorá prebieha v šifrovanej podobe, ktorá sa navonok javí aj pre program *Wireshark* ako zabezpečená HTTP komunikácia.

³<http://www.wireshark.org/>



Obrázok 3.6: Príklad výmeny správ v šifrovanej podobe - HTTPS

3.8 Konfigurácia a autentizácia modulov

Konfigurácia oboch modulov prebieha pomocou *konfiguračného súboru*, ktorý musí byť predaný konfiguračným metódam ešte pred štartom samotného modulu. Pomocou údajov, obsiahnutých v tomto súbore, sa nakonfigurujú jednotlivé súčasti oboch komunikujúcich modulov, z tohto dôvodu je dôležité, aby na oboch stranách figuroval rovnaký konfiguračný súbor, pretože inak nedôjde k vzájomnej synchronizácii a obfuskácia sa nebude môcť vykonávať. Kompletný príklad vzorových konfiguračných súborov pre rôzne formy obfuskácie je demonštrovaný v prílohe A.

Jednotlivé konfiguračné príkazy, možné hodnoty a ich význam si stručne popíšeme:

- **server_ip** - Definuje IP adresu, na ktorú sa bude proxy modul snažiť pripojiť pri nadväzovaní spojenia. V príkaze je teda potrebné definovať adresu počítača, na ktorom beží užívateľský program využívajúci knižnicu a v prípade, že k používaniu dochádza mimo privátnych a virtuálnych sietí, musí byť táto adresa *verejná*.
- **server_mode** - Určuje akým spôsobom bude komunikácia obfuskovaná, teda ako bolo popísané v kapitole 3.7.2, či komunikácia bude maskovaná ako HTTP alebo HTTPS, teda v otvorenej podobe alebo šifrovane. Konfigurácia sa vykonáva nastavením hodnoty tohto príkazu, konkrétne buď **plain**, prípadne **encrypted**. Na základe zvolenia jednej z týchto hodnôt, sa automaticky nakonfigurujú oba moduly a komunikácia prebieha buď na porte *80* pre otvorený režim alebo na porte *443* v šifrovanom režime.
- **destination_ip** - Pomocou tohto príkazu je možné definovať IP adresu, prípadne masku celej podsiete, ktorá predstavuje cieľový bod vo vnútornej chránenej sieti. Na základe tejto hodnoty bude čiastočne vykonávané odchytyvanie a transformovanie odchádzajúcich paketov z hostiteľského systému pre užívateľský program, používajúci knižnicu. Hodnota sa taktiež ako doplňujúci údaj používa aj v proxy module, pri vykonávaní ďalšieho smerovania vo vnútornej sieti.
- **protocol_modul**, **scan_modul** - Umožňujú definovať, aký obfuskáčny modul sa bude používať, konkrétne, či sa obfuskácia vykonáva iba pre definovaný protokol, alebo je potrebné obfuskovať kompletne skenovanie portov. V prvom prípade sa ako parametre

očakávajú čísla portov, ktoré budú zahrnuté do obfuskačného procesu. Súčasne môže byť v konfiguračnom súbore definovaná iba jedna z týchto variant, v opačnom prípade bude súbor považovaný za nekorektný. Hodnoty definované v týchto príkazoch sú taktiež používané pri dynamickom vytváraní pravidiel odchyťovania pre odchádzajúce pakety.

- `client_prefix`, `client_suffix` - Dávajú možnosť užívateľovi knižnice dodefinovať hlavičky, ktoré budú používané pri vytváraní HTTP paketov. V tomto prípade je možné špecifikovať, v akom tvare bude zasielaná komunikácia z proxy modulu na server. Pomocou týchto hodnôt sa dá prispôsobiť obfuskovaná komunikácia v otvorenom režime na základe charakteristiky siete, kde bude obfuskácia nasadená.
- `server_prefix`, `server_suffix` - Obdobne, ako v predošlom prípade, umožňuje definovať hlavičky, ktoré budú použité pri vytváraní paketov zasielaných zo serveru na proxy modul. V oboch prípadoch je možné využiť *escape sekvencie*, ktoré budú vo výsledku nahradené. Príkladom môže byť použitie `\1`, ktoré sa vo výsledných hlavičkách dynamicky nahrádza za aktuálnu dĺžku paketu.

Uvedené príkazy konfiguračného súboru ponúkajú užívateľovi knižnice kompletne riadenie behu oboch modulov a samotnej obfuskácie. Tento prístup bol zvolený hlavne z dôvodu, aby bolo možné všetky nastavenia vykonať na jednom mieste, a tým zjednodušiť vlastné použitie knižnice, pretože v prípade riešenia zadávaním parametrov napríklad z príkazového riadku programu by sa o parsovanie parametrov musel starať užívateľ.

S konfiguračným súborom čiastočne súvisí aj *autentizácia* proxy modulu voči užívateľskému serveru, pretože autentizačný reťazec je vytváraný na základe údajov v konfiguračnom súbore. Týmto prístupom sa zabezpečí, že server bude komunikovať iba s odpovedajúcim proxy modulom a iný klient sa k serveru nepripojí. Zároveň tým dosiahneme, že server aj proxy modul budú obsahovať *zhodný konfiguračný súbor*, čo je pre úspešnú obfuskáciu nevyhnutné.

Autentizácia prebieha, stručne povedané, zaslaním autentizačného reťazca v obfuskovanej podobe z proxy modulu na server, ktorý porovná dekodovanú hodnotu so svojou a v prípade úspešného porovnania sa považuje komunikácia za autentizovanú, v opačnom prípade, ak hodnoty nesúhlasia, komunikácia sa ukončí.

Autentizačný reťazec je vytváraný na základe *významných príkazov* a ich parametrov v konfiguračnom súbore, ktoré sú vstupom bloku obsahujúceho hashovaciu funkciu *SHA-1*. Výstupom bloku je textový reťazec, ktorý je získaný prekódovaním *160* bajtového výstupu *SHA-1* do odpovedajúcej textovej podoby. K tomuto výpočtu dochádza samozrejme u oboch komunikujúcich entít, čím sa zabezpečí, že hodnota sa bude zhodovať iba v prípade *zhodnej konfigurácie* oboch modulov.

V tejto kapitole sme si stručne priblížili spôsoby konfigurácie obfuskačnej knižnice a ukázali tiež spôsob využitia týchto dát pri autentizácii proxy modulu. Autentizácia je pomerne dôležitá z toho dôvodu, že fiktívny webový server by mohol byť zahltený bežnými požiadavkami, zasielanými na otvorené porty, ktoré sa javia ako webová služba. Takéto zahlcovanie a blokovanie by potom mohlo byť jednoducho využívané pri snahe zamedziť vykonávaniu samotnej obfuskácie, pričom zavedenie popísanej autentizácie by tomu malo aspoň čiastočne zabrániť.

Kapitola 4

Experimenty s knižnicou a dosiahnuté výsledky

Nasledujúca kapitola sa zaoberá popisom testovania navrhutej a implementovanej knižnice vo virtuálnom prostredí, kde je možné jednoducho simulovať rôzne situácie, ktoré by pri reálnom nasadení mohli nastať. Úlohou testovania je zistiť, do akej miery je knižnica schopná zamedziť detekcii komunikácie IDS systémom, na druhej strane je tiež nutné demonštrovať jej schopnosť *transparentne prenášať* obfuskovanú komunikáciu, aby sa na hosťateľskom operačnom systéme dala bez problémov používať.

Za účelom testovania bola teda vytvorená *virtuálna sieť*, do ktorej bolo možné nasadiť stanice s rôznymi operačnými systémami. Samotné testovanie môžeme rozdeliť na dve samostatné etapy, keď prvá zahŕňa testovanie na *legitímnej komunikácii*, pri ktorej sa otestuje úspešnosť prenosu a distribúcie paketov v rámci vnútornej siete, dynamické odchyťovanie paketov a transformácia zasielaných paketov. Úspechom pri tomto type testovania je korektné nadviazanie komunikácie medzi danými komunikujúcimi entitami, pričom nedôjde v inštalovanom IDS systéme k detekcii používaného obfuskovaného protokolu.

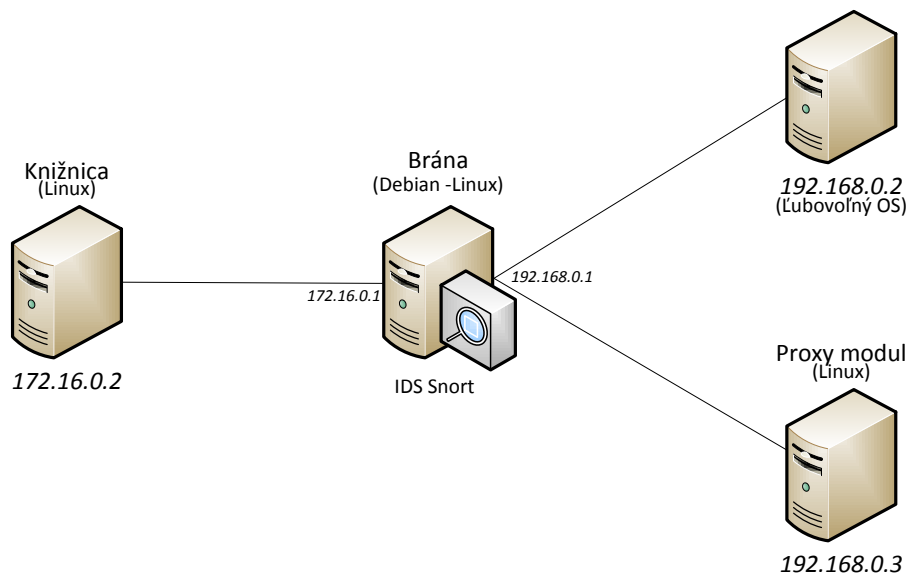
Druhá etapa testovania je postavená na zasielaní *kompromitujúcich dát*, teda zvyčajne exploitov, ktoré útočia na nejakú zraniteľnosť v systéme a mnohokrát k svojej činnosti využívajú a prenášajú *shellkód*. V tomto prípade je najdôležitejším ukazovateľom úspešného testu, ak IDS neodhalí daný útok a nevygeneruje ani varovanie o prenose *shellkódu*.

V nasledujúcich kapitolách sa podrobnejšie zameriame na popis vytvorenej virtuálnej siete, na ktorú budú potom jednotlivé testy aplikované. Následne budú uvedené podrobnosti jednotlivých testov, ich priebeh a výsledky.

4.1 Testovacia virtuálna sieť

Pre potreby vlastného testovania bola vytvorená virtuálna sieť, zložená zo *štyroch* samostatných počítačov, ktoré boli rozdelené do dvoch rozdielnych *podsietí*. Jedna podsieť predstavovala vnútornú sieť, chránenú pomocou IDS systému, pričom druhá podsieť predstavovala okolitý *nedôveryhodný priestor*. Schému zapojenia a nastavenia IP adries u jednotlivých virtuálnych strojov môžeme vidieť na obrázku 4.1.

Nosným prvkom tohto zapojenia je *sieťová brána*, ktorá zabezpečuje prepojenie spomenutých dvoch podsietí tak, aby mohla medzi nimi prebiehať komunikácia. Dôležitým prvkom brány je aj inštalovaný IDS systém *Snort*, ktorý kontroluje všetku prebiehajúcu komunikáciu. Brána bola v našom prípade postavená na operačnom systéme *Debian* v aktuálnej



Obrázok 4.1: Štruktúra zapojenia testovacej siete

verzii 6.0, pričom obsahuje tri na sebe nezávislé sieťové rozhrania. Rozhrania, do ktorých sú pripojené počítače z vnútornej siete, sú v bráne *softwarovo premostené*, čím je umožnená ich vzájomná komunikácia. Na druhej strane komunikácia s počítačom vo vonkajšej sieti je riešená pomocou *prekladu adres*, keď v prípade potreby zasielania mimo vnútornú sieť sa vnútorná adresa z podsiete 192.168.0.0/24 preloží na adresu vonkajšieho rozhrania z podsiete 172.16.0.0/16. Týmto zapojením dosiahneme, že všetka komunikácia vo virtuálnej sieti musí byť spracovaná pomocou brány.

Súčasťou brány je, ako bolo spomenuté, aj IDS *Snort* v aktuálnej verzii 2.9.4, ktorý má nakonfigurovanú ako domácu sieť 192.168.0.0/24 a všetko mimo tejto siete je považované za *nedôveryhodné*. Pri konfigurácii boli taktiež povolené všetky dostupné moduly *preprocesorov*, ktoré vykonávajú pokročilejšie spracovanie špecifikovaných dátových tokov. Súbor pravidiel bol vždy pri štarte automaticky aktualizovaný na poslednú dostupnú verziu pomocou nástroja *pulledpork*¹ a na jednoduchšiu správu vygenerovaných hlásení boli udalosti ukladané do databázy. Databáza bola ďalej spracovávaná nástrojom *BASE* (Basic Analysis and Security Engine) [12], ktorý umožňuje jednoduché triedenie a správu vygenerovaných hlásení.

Virtuálne stroje, slúžiace ako hostiteľské pre užívateľskú časť aj proxy modul, musia byť postavené na systéme *Linux* s minimálnou verziiu jadra 2.6 a vyššou, v našom prípade boli použité distribúcie založené na systéme *Ubuntu*, na ktoré boli okrem samotných modulov doinštalované aj používané knižnice, popísané v kapitole 3.5. Posledným prvkom siete je cieľový počítač s adresou 192.168.0.2, ktorý bude *koncovou destináciou* testovacej komunikácie, prípadne sieťových útokov. Operačný systém a jeho konfigurácia bude obmieňaná na základe charakteristiky testu (potreba vytvorenia zraniteľných miest alebo inštalácia sieťových služieb), pričom bude spresnená pri popise konkrétnych experimentov.

¹<https://code.google.com/p/pulledpork/>

4.2 Experimenty s legitímnou komunikáciou

Pri počítačnom testovaní na legitímnej komunikácii bolo potrebné zvoliť protokoly tak, aby sme boli schopní otestovať možnosti úspešného zostavenia komunikácie a v druhom rade zistiť, či prebieha úspešne aj jej zamaskovanie. Z tohto dôvodu boli pre počítačové testy zvolené protokoly, slúžiace na vzdialené pripojenie, konkrétne *telnet* a *ssh*.

Ako cieľový systém inštalovaný na stroji s IP adresou *192.168.0.2* bol v tomto prípade použitý operačný systém *CentOS* vo verzii *6.3*, na ktorom boli tieto testovacie služby spustené, teda aj zodpovedajúce porty boli *otvorené*, konkrétne pre *telnet* port číslo *23* a pre *ssh* port *22*. Jednotlivé testy a ich výsledky si stručne popíšeme.

4.2.1 Protokoly *telnet* a *ssh*

Komunikácia protokolom *telnet* prebieha v otvorenej podobe, preto je jeho použitie na vzdialený prístup neodporúčané, na druhej strane, pre naše potreby je testovanie s týmto protokolom výhodné, pretože IDS Snort obsahuje priamo preprocesor, ktorý vykonáva kontrolu nad zasielanými správami. IDS z tohto dôvodu vygeneruje varovanie pri nedôveryhodnej výmene správ týmto protokolom, napríklad ako podozrivý sa javí aj chybný pokus o prihlásenie. Ako úspešné v tomto prípade budeme považovať výsledok, keď sa podarí vytvoriť úspešné spojenie a súčasne, aj v prípade zadania chybného mena a hesla, nedôjde k vygenerovaniu hlásenia IDS systémom.

Ako referenčnú hodnotu bolo potrebné získať odozvu, ktorá vznikne *bez použitia* obfuskáčnej knižnice. Ako reakcia na pozitívne porovnanie so vzorom v prípade pokusu o chybné prihlásenie, dôjde ku vygenerovaniu nasledujúceho hlásenia:

| < Signatúra > | < Zdrojová adresa > | < Cieľová adresa > |
|----------------------|---------------------|--------------------|
| GPL TELNET Bad Login | 192.168.0.2:23 | 172.16.0.2:34164 |

Z výstupu je zrejmé, že IDS je pri analýze schopné kontrolovať zasielané správy v *telnet* spojení a porovnávaním uložených vzorov z databázy pravidiel odhaliť prípadné nežiaduce správy, ktoré v prípade výskytu loguje v podobe vygenerovaného alarmu.

Pri použití obfuskáčnej knižnice je nutné nastavenie, ktoré zahŕňa transformáciu *telnet* protokolu, teda parameter `protocol_modul` z konfiguračného súboru musí obsahovať minimálne definovanú hodnotu *23*. Po spustení procesu konfigurovanej obfuskácie je možné bez problémov vytvoriť spojenie s cieľovou stanicou a aj v prípade chybného pokusu o prihlásenie IDS nehlásilo *žiadne varovania*. Testovanie prebiehalo, či už pre komunikáciu v otvorenej podobe, maskovanú v HTTP protokole, alebo šifrovanú komunikáciu HTTPS protokolom, pričom v oboch prípadoch bolo *úspešné*.

Obdobným spôsobom prebiehalo aj testovanie s *ssh* protokolom, ktorý na rozdiel od *telnet*-u pracuje so šifrovaným bezpečným kanálom. Z tohto dôvodu IDS nemôže priamo kontrolovať obsah dátových paketov, a tým ani generovať varovania na základe ich obsahu. Cieľom tohto testovania bola teda snaha demonštrovať, že knižnica je schopná obfuskovať aj komunikáciu zložitejším protokolom. Oproti protokolu *telnet*, ktorý pracuje jednoducho štýlom dotaz - odpoveď vo forme klasických textových správ, pri inicializácii *ssh* kanála musí na druhej strane dôjsť k výmene šifrovacích kľúčov a tiež sa vykonávajú kontroly, či nedošlo k nejakej modifikácii zasielaných paketov.

V oboch prípadoch obfuskácie sa podarilo *úspešne nadviazať komunikáciu*, ktorá nebola nijako poznačená obfuskáciou a komunikujúcemu klientovi sa javila ako *transparentná*. Na strane IDS nedošlo k detekcii žiadnych sieťových anomálií a komunikácia nebola označená žiadnymi hláseniami. Na základe toho môžeme test považovať za *úspešný*.

4.2.2 Skenovanie portov

Operácia skenovania portov sa nachádza na hranici *legitímnej komunikácie*, pretože často vedie k ďalším pokusom o útok na cieľovú stanicu, na druhej strane sa však pri jej vykonávaní nezasielajú žiadne škodlivé dáta. Z dôvodu zvýšenia bezpečnosti systémov, IDS systémy potenciálne skenovanie portov zaznamenávajú a konkrétne *Snort* obsahuje na pokročilejšiu analýzu samostatný preprocesor. Pri skenovaní portov zasiela zdrojový systém na cieľovú stanicu množstvo požiadaviek na jednotlivé porty a na základe odpovedí sa snaží zistiť, ktorý z portov je otvorený, prípadne, aká služba na ňom beží. Z hľadiska testovania obfuskanej knižnice je najdôležitejšou úlohou zamedziť vygenerovaniu varovaní, ktoré sa môžu objaviť z dôvodu detekcie podozrivej komunikácie. Dôležitou vlastnosťou úspešnej obfuskácie je tiež získanie zhodného výsledku skenovania ako v prípade behu bez obfuskácie.

Testovanie bolo vykonávané pomocou programu *nmap*, ktorý umožňuje vykonávanie rôznych typov skenovania portov. V prvom kroku boli vytvorené referenčné hodnoty tak, že sme vykonali skenovanie príkazom `nmap 192.168.0.2`, ktorý vykoná preskenovanie najčastejšie využívaných portov. IDS na túto operáciu reagovalo nasledujúcimi hláseniami:

```
< Signatúra >
                                < Zdrojová adresa >           < Cieľová adresa >
portscan: TCP Portscan
                                172.16.0.2                 192.168.0.2

ET POLICY Suspicious inbound to mySQL port 3306
                                172.16.0.2:57558           192.168.0.2:3306

ET POLICY Suspicious inbound to MSSQL port 1433
                                172.16.0.2:45127           192.168.0.2:1433

ET POLICY Suspicious inbound to PostgreSQL port 5432
                                172.16.0.2:55873           192.168.0.2:5432

ET SCAN Potential VNC Scan 5900-5920
                                172.16.0.2:57306           192.168.0.2:5904

ET SCAN Potential VNC Scan 5800-5820
                                172.16.0.2:38582           192.168.0.2:5802
```

Ako môžeme z výpisu vidieť, *Snort* bol schopný odhaliť anomálie na niektorých portoch, ale zároveň označiť celú vykonávanú komunikáciu ako *skenovanie portov*. Jednotlivé hlásenia sú vygenerované rôznymi modulmi, čo naznačuje, že vykonávaná analýza prichádzajúcej komunikácie je vykonávaná podrobne s použitím *viacerých preprocesorov*.

Pri testovaní tohto typu obfuskácie sa overí funkčnosť druhého implementovaného modulu, to znamená, že v konfiguračnom súbore musí byť aktívna druhá položka oproti predošlému prípadu, konkrétne príkaz `scan_modul`. Vykonanie obfuskácie, so správnou nakonfigurovanou a spustenou obfuskacnou knižnicou, zabezpečilo získanie *zhodných výsledkov skenovania* ako v prípade behu bez obfuskácie, pričom pri oboch typoch transformácie odosielaných dát neodhalil *Snort* žiadne podozrivé pakety, a teda nedošlo k vygenerovaniu *žiadnych hlásení*. Z tohto dôvodu môžeme tento test považovať za *úspešný*.

4.3 Experimenty s exploitačnými dátami

Druhá fáza testovania prebiehala, oproti predošlým prípadom, s dátami, ktoré sa snažili vykonať nejaké nekalé činnosti na cieľovom systéme, teda najčastejšie získať *vzdialený prístup* k cieľovému počítaču. Pri takýchto útokoch sa najčastejšie využíva zraniteľnosť v službe, ktorá je prístupná na vzdialenom počítači, pričom sa útočí na zraniteľné miesta spôsobené zvyčajne chybami pri práci s pamäťou v programe, teda napríklad pretečenie zásobníka (*stack overflow*) alebo pretečenie pamäti (*buffer overflow*) [7]. Útoky zneužívajú tieto zraniteľnosti tým spôsobom, že zabezpečia vykonanie svojho vlastného kódu (*arbitrary code*) na vzdialenom počítači, ktorý zabezpečí uskutočnenie jadra útoku, zvyčajne eskaláciu prístupových práv a vytvorenie vzdialenej TCP relácie na kompromitovanom stroji. S vykonaním úzko súvisí distribúcia exploitačného kódu do vzdialeného počítača, ktorý sa najčastejšie zasiela v podobe *shellkódu*.

Na druhej strane sa na detekciu shellkódu orientujú aj samotné IDS systémy, ktoré sa snažia odhaliť dátové pakety obsahujúce takýto kompromitovaný *payload*. Úlohou obfuskanej knižnice nie je v tomto prípade snaha o úplné zamaskovanie komunikácie, ale zabránenie, aby nebolo *jadro útoku* detekované. Samotné útoky boli pri testovaní vykonávané vo väčšine prípadov pomocou nástroja *metasploit*², slúžiaceho na pokročilé penetračné testovanie systémov, ktorý obsahuje rozsiahlu databázu známych exploitov, ktoré je možno nakonfigurovať podľa vlastných požiadaviek daných vlastnosťami cieľového systému. V nasledujúcich sekciách si podrobnejšie popíšeme niektoré útoky, spôsob testovania a výsledky, získané pri obfuskácii s knižnicou.

4.3.1 Útoky na webové služby

Dôležitým aspektom pri vývoji knižnice bola jej univerzálnosť, a z dôvodu, že na prenos obfuskovaných dát sa využívajú protokoly HTTP a HTTPS, bolo potrebné otestovať schopnosť knižnice zamaskovať aj komunikáciu, ktorá je v originálnej forme prenášaná týmito protokolmi, a teda dochádza k transformácii HTTP do obfuskovaného HTTP, obdobne aj v prípade šifrovanej varianty HTTPS. Testovanie by teda malo v hlavnej miere overiť schopnosť knižnice odchytať pakety na hostiteľskom počítači tak, aby boli zachytávané iba pakety s pôvodnými prenášanými dátami a odosielané, knižnicou transformované pakety, zostali ďalej systémom *nepovšimnuté*. Ak by dochádzalo k odchyťovaniu a obfuskovaniu už transformovaných paketov, vznikol by *cyklus*, pri ktorom by sa paket rekurzívne transformoval a nikdy by nedošlo k jeho odoslaniu. Na demonštráciu boli vybrané dva rôzne útoky, zameriavajúce sa na exploitáciu na HTTP porte 80, respektíve na HTTPS porte 443. Podrobnejšie si jednotlivé testy popíšeme.

Ako štandardný webový server bežiaci na porte 80 bola zvolená jednoduchá, odľahčená implementácia webového servera *BadBlue* v poslednej vydanej verzii 2.72b, ktorá trpí zraniteľnosťou založenou na *pretečení zásobníka*, popísaná v CVE-2007-6377 [19]. Útočník môže využiť túto zraniteľnosť vo svoj prospech, pretože server nesprávne kontroluje *URI* adresu a pri zaslaní špeciálne zostavenej HTTP požiadavky, môže dôjsť k porušeniu integrity a pretečeniu, ktoré vedie až k možnosti vykonania vlastného kódu útočníka. Pri vykonávaní útoku sa zasiela všeobecne známy *shellkód*, ktorý zabezpečí otvorenie reverzného TCP spojenia späť na počítač útočníka, čím sa zabezpečí vzdialený prístup s administrátorskými právami na cieľovom počítači. Implementácia webovej služby bola inštalovaná na serveri

²<http://www.metasploit.com/>

bežiacom v operačnom systéme *Windows 2000*, ktorý bol pripojený vo virtuálnej sieti na štandardnú IP adresu *192.168.0.2*, vyplývajúcu z architektúry testovacej siete.

Prvým krokom vykonávaného testu je získanie referenčného výstupu IDS, *ak nebola* pri komunikácii využitá obfuskačná knižnica. Výstup obsahuje nasledujúce varovania:

| < Signatúra > | < Zdrojová adresa > | < Cieľová adresa > |
|--|---------------------|--------------------|
| ET SHELLCODE Rothenburg Shellcode | 172.16.0.2:39159 | 192.168.0.2:80 |
| INDICATOR-SHELLCODE x86 OS agnostic fnstenv geteip dword xor decoder | 172.16.0.2:39159 | 192.168.0.2:80 |
| http_inspect: LONG HEADER | 172.16.0.2:39159 | 192.168.0.2:80 |

Prvé dve varovania sú spôsobené detekciou prenášaného shellkódu, ktorý IDS systém pri analýze dokázal odhaliť a zároveň aj klasifikovať, pretože sa jedná o rozšírenú variantu určenú na vytvorenie reverzného spojenia pre systémy *Windows*. Posledné varovanie predstavuje upozornenie na prenos príliš dlhej URI adresy, snažiacej sa spôsobiť pretečenie zásobníka na cieľovom systéme. Po prenesení tejto kompromitovanej komunikácie došlo k vytvoreniu vzdialenej relácie s administrátorskými právami, a teda ku kompletnému ovládnutiu cieľového systému.

Záverečná fáza testovania zahŕňa prevedenie popísaného útoku so spustenou obfuskačiou pre oba režimy prenosu transformovaných dát, teda v šifrovanej aj nešifrovanej podobe. V oboch prípadoch došlo k *prelomeniu cieľového počítača*, vytvoreniu vzdialeného sedenia a zároveň IDS nevygeneroval žiadne varovania, preto môžeme test zameraný na obfuskačiu HTTP komunikácie považovať za *úspešný*.

Druhý útok v tejto práci orientovaný na webové služby je zameraný na exploitáciu servera *apache*, ktorý je rozšírený doplnkom `mod_ssl`, teda kompromitovaná komunikácia prebieha na šifrovanom porte *443*. Zraniteľnosť servera vychádza z možného *pretečenia pamäti* v inštalovanom doplnku, keď niektoré staršie verzie *apache* používajú rozšírenie `mod_ssl` vo verzii nižšej ako *2.8.7*, ktorá obsahuje chybu v práci s vyrovnávacou pamäťou, keď sa nesprávne ukladajú dáta aktuálnych spravovaných vzdialených relácií. Jedná sa v praxi o pomerne ťažko exploitovateľnú zraniteľnosť, ale na demonštráciu funkčnosti knižnice dostačuje. Podrobnosti o útoku a možnosti využitia zraniteľnosti boli čerpané z odpovedajúceho CVE-2002-0082 [18].

Útok vychádza zo znalosti, že `mod_ssl` nesprávne inicializuje pamäť a pri uložení SSL relácie môže dôjsť k pretečeniu pamäti. Pri útoku je nutné pokúsiť sa o zväčšenie dát reprezentujúcich reláciu, čo sa dá dosiahnuť tak, že sa použije extrémne veľký certifikát klienta, pričom musí byť splnené, že na strane servera dochádza k overovaniu certifikátu a ten musí byť podpísaný certifikačnou autoritou, ktorej web server dôveruje. Z uvedeného popisu vyplýva, že útok je pomerne ťažko realizovateľný pri reálnom nasadení, ale pri vlastných experimentoch boli uvedené podmienky na úspešné prelomenie splnené, s využitím servera s operačným systémom *Linux*, na ktorom bol nainštalovaný *apache* vo verzii *1.3.20* s doplnkom `mod_ssl` verzie *2.8.4*.

Útok bol vykonávaný pomocou verejne dostupného exploitu [8], ktorý bol modifikovaný na správnu funkčnosť vo využívanej virtuálnej sieti. Použitý exploit využíva uvedené zraniteľnosti a po získaní prístupu s obmedzenými právami k serveru, dopĺňa činnosť o distribúciu známeho lokálneho exploitu `ptrace_kmod.c` [23], ktorý umožní získanie administrátorských práv a kompletné ovládnutie cieľového operačného systému. Ako je z popisu útoku zrejmé,

celá komunikácia prebieha v šifrovanej podobe, a preto aj pri testovaní bez využitia obfuskacej knižnice, IDS nie je schopné odhaliť prenášané exploitačné dáta a nevygeneruje žiadne hlásenia. Test v tomto prípade nie je zameraný priamo na zamaskovanie komunikácie, ale má overiť možnosť transparentného prenosu šifrovaného útoku či už v otvorenej HTTP podobe alebo opätovným zašifrovaním do HTTPS podoby. Útok bol pri praktických testoch, s využitím obfuskacej knižnice, pri využití oboch typov prenosu *úspešný* a vždy sa podarilo vytvoriť vzdialenú reláciu s administrátorským prístupom k cieľovému počítaču.

Úspešnosť popísaných testov pramení najmä zo spôsobu implementácie spracovávania odchytených paketov, keď sa transformácia do obfuskovanej podoby a samotné odoslanie realizuje na *systémovej úrovni*, čím sa zabezpečí transparentnosť voči hostiteľskému systému, a teda nedôjde k zacykleniu ani v prípade realizácie obfuskacie protokolov, ktoré sú zároveň používané na zamaskovanie komunikácie v sieti. Výsledky týchto testov by mali potvrdiť schopnosti knižnice univerzálne maskovať komunikáciu bežnými protokolmi v sieťovom prostredí.

4.3.2 Zraniteľnosť Microsoft DCOM-RPC

Nasledujúci popísaný experiment sa orientuje na útok [29] zameriavajúci sa na zraniteľnosť, ktorá sa nachádza v riešení volania vzdialených procedúr (RPC) systému *Microsoft Windows*. RPC umožňuje volanie procedúr medzi procesmi, ktoré môžu byť situované na rôznych systémoch. RPC využíva pri svojej činnosti takzvané DCOM rozhranie, ktoré však obsahuje zraniteľnosť spôsobenú chybnou prácou s pamäťou, keď môže dôjsť k jej pretečeniu. Konkrétne sa táto chyba nachádza v systémoch *Windows XP, 2000 a 2003 Server*. Útočník môže v tomto prípade exploitovať túto zraniteľnosť a získať vzdialený prístup k napadnutému počítaču. Jedná sa o pomerne známu zraniteľnosť, pri ktorej dochádza k distribúcii shellkódu vytvárajúceho reverzné TCP spojenie. Zraniteľnosť svojho času využíval aj známy a rozšírený internetový červ *Blaster*, pričom súhrnné informácie, odkiaľ čiastočne čerpala aj táto práca sú uvedené v CVE-2003-0352 [20].

Na otestovanie knižnice bol nasadený, ako cieľový počítač s IP adresou *192.168.0.2* vo virtuálnej sieti, stroj s operačným systémom *Windows 2000*, na ktorom bola povolená RPC služba, bežiacia na porte *135*. Samotný útok na túto službu bol vykonaný pomocou nástroja *metasploit*, keď po uskutočnení príslušného útoku došlo k vytvoreniu vzdialenej relácie na cieľovom počítači, ktorá umožňovala vzdialený prístup s administrátorskými právami na napadnutom počítači, pričom pri teste bez použitia obfuskacej knižnice IDS systém *Snort* vygeneroval nasledujúce hlásenia:

```

< Signatúra >
                < Zdrojová adresa >                < Cieľová adresa >
ET SHELLCODE Rothenburg Shellcode
                172.16.0.2:54134                    192.168.0.2:135

INDICATOR-SHELLCODE x86 OS agnostic fnstenv geteip dword xor decoder
                172.16.0.2:54134                    192.168.0.2:135

OS-WINDOWS DCERPC NCACN-IP-TCP
IActivation remoteactivation overflow attempt
                172.16.0.2:54134                    192.168.0.2:135

```

Z uvedeného výpisu je zrejmé, že *Snort* nemá problém s odhalením takéhoto rozšíreného útoku a dokonca okrem detekcie pokusu o využitie zraniteľnosti spôsobenej pretečením pamäti a odhalenia samotného shellkódu, dôjde aj k jeho klasifikácii.

Testovanie útokov, s využitím obfuskačnej knižnice, ktorá musela byť nakonfigurovaná na obfuskáciu komunikácie na porte s číslom *135*, bolo *úspešné* aj pri tomto útoku, keď IDS nevygeneroval ani jeden z uvedených alarmov, pričom na druhej strane bol útok stále úspešný a došlo k vytvoreniu vzdialenej relácie na cieľovom počítači. Zhodný priebeh útoku bol zaznamenaný v oboch prístupoch k obfuskácii, pretože aj v prípade komunikácie v otvorenej podobe je shellkód transformovaný do HTTP paketu v textovej forme, čo neumožní IDS systému jeho detekciu.

4.3.3 Útok na službu *samba*

Posledný podrobne popísaný útok je zameraný na exploitáciu služby *samba*, ktorá sa používa na zdieľanie súborov medzi operačnými systémami Windows a Linux. Jedná sa o pomerne rozšírenú aplikáciu využívanú v mnohých sieťach, kde je potrebné zabezpečiť zdieľanie súborov medzi rôznymi operačnými systémami. Úlohou tohto testu je potvrdiť univerzálnosť navrhnutého riešenia pracovať, na rozdiel od predošlého prípadu zameriavajúceho sa na Windows platformu, aj ak cieľový stroj beží na inom operačnom systéme.

Útok je aj v tomto prípade postavený na pretečení pamäti vo funkcii `call_trans2open`, z čoho je odvodený aj názov exploitu, ktorý je popísaný v CVE-2003-0201 [21]. Zraniteľnosť je spôsobená tým, že dokonca anonymný, neautentizovaný užívateľ má možnosť zaslať a uložiť dáta do staticky alokovaného miesta v pamäti, čo môže viesť k pretečeniu a poškodeniu citlivých miest v pamäti. Úspešné vykonanie tohto útoku má za následok vykonanie kódu útočníka s právami, s ktorými je spustený samotný *samba* proces na cieľovom počítači. Uvedená zraniteľnosť sa nachádza v starších verziách od *2.2.0* po *2.2.8* a existuje množstvo dostupných exploitov, ktoré túto chybu využívajú. V našom prípade bol použitý exploit nachádzajúci sa v nástroji *metasploit*, ktorý ako payload opäť zasiela shellkód vytvárajúci reverzné TCP spojenie s počítačom útočníka. Priebeh samotného útoku je realizovaný hrubou silou (*brute-force*), pretože je potrebné odhadnúť návratovú adresu, kde bude škodlivý shellkód uložený. Návratová adresa čiastočne závisí na platforme, na ktorej služba beží ale presne sa odhadnúť nedá. Útok hrubou silou na druhej strane lepšie preverí schopnosti knižnice maskovať danú komunikáciu.

Testovanie prebiehalo voči operačnému systému *Linux*, na ktorom bola nainštalovaná a nakonfigurovaná sieťová služba *samba* vo verzii *2.2.1a*. Referenčný výstup hlásení IDS pri vykonávaní útoku na danú službu bol získaný priamo exploitáciou služby bez použitia obfuskačnej knižnice.

Získaný výpis hlásení potvrdzujúci schopnosť IDS zaznamenať kompromitovanú komunikáciu v sieti je nasledovný:

| < Signatúra > | < Zdrojová adresa > | < Cieľová adresa > |
|--|---------------------|--------------------|
| GPL NETBIOS SMB trans2open buffer overflow attempt | 172.16.0.2:33539 | 192.168.0.2:139 |
| GPL NETBIOS SMB IPC\$ share access | 172.16.0.2:33539 | 192.168.0.2:139 |

Dvojice týchto hlásení boli v originálnom výstupe opakované viackrát, na základe počtu pokusov, ktoré musel exploit pri útoku hrubou silou vykonať, kým sa mu podarilo preložiť cieľovú službu a získať vzdialený prístup s administrátorskými právami na cieľovom počítači. IDS pri tomto útoku dokáže odhaliť, že útočník sa snaží využiť známu zraniteľnosť *trans2open*, pričom toto hlásenie je doplnené o upozornenie, že pripojenie na *samba* server prichádza z počítača mimo chránenej siete.

Realizácia útokov s využitím knižnice nakonfigurovanej na obfuskáciu komunikácie na porte *135*, na ktorom služba *samba* štandardne beží, prebehla *úspešne* aj v tomto prípade pre oba typy prenášanej obfuskovanej komunikácie. IDS systém teda nevygeneroval žiadne varovania ani pri testovaní útoku hrubou silou, keď exploit potreboval zvyčajne približne desať pokusov na korektné dokončenie. U IDS tiež nedošlo k vygenerovaniu žiadneho hlásenia ani v prípade použitia HTTP prenosu, na ktorého analýzu *Snort* používa aj samostatný preprocesor. Na základe týchto výsledkov môžeme tvrdiť, že aj táto komunikácia je za týchto podmienok IDS systémom nedetekovateľná.

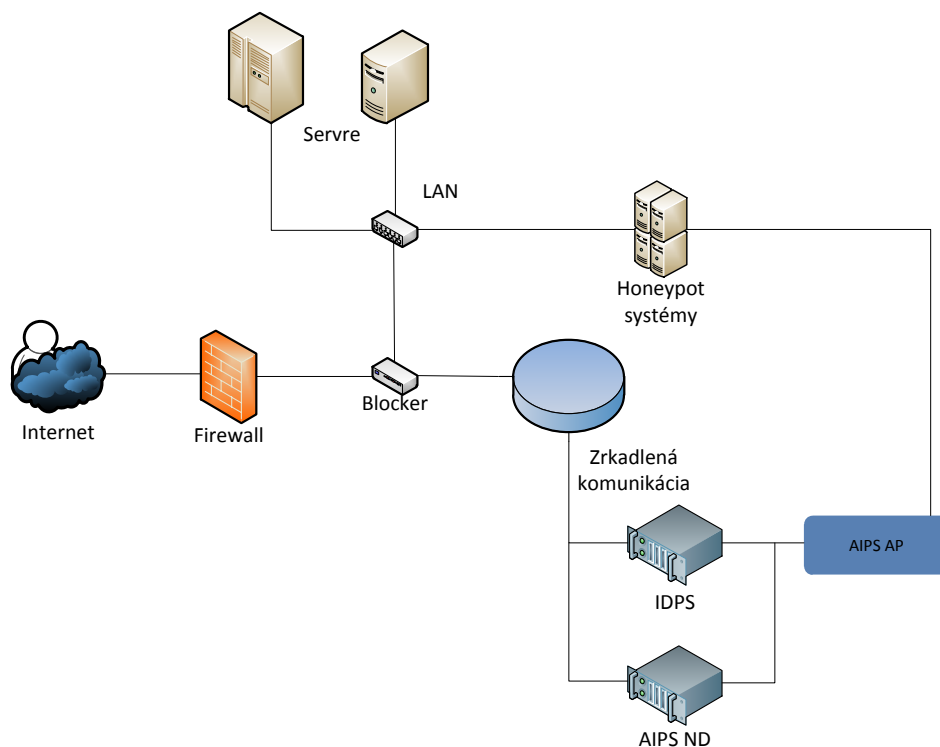
4.4 AIPS systém

Správna funkcia a efektivita navrhnutého implementovaného nástroja mala byť následne otestovaná v rámci projektu AIPS, ktorý je vyvíjaný na FIT VUT v Brne výskumnou skupinou *Buslab*. Projekt AIPS využíva pri svojej činnosti *Honeypot* systémy, ktoré generujú *behaviorálne pravidlá* (signatúry) pre moduly, ktoré slúžia na detekciu útokov. Hlavnou úlohou tohto systému je zvýšiť schopnosť detekcie nových, dosiaľ neznámych a *zero-day* útokov, pri ktorých bežné rozpoznávanie vzorov zlyháva. Architektúra sieťovej detekčnej časti AIPS systému je znázornená na nasledujúcom obrázku číslo 4.2.

Zo zobrazenej schémy vidíme, že systém sa delí na dve spolupracujúce časti, keď základom je AIPS Network Detector (ND), pracujúci ako sieťová sonda, schopná odhaľovať pokusy o útoky, pričom využíva databázu znalostí, ktorú spracováva AIPS Attack Processor (AP). Tieto prvky sú ďalej doplnené o Intrusion Detection and Prevention System (IDPS), zabezpečujúci detekciu, prípadne doplnenú o vykonanie preventívneho zásahu v reálnom čase, ktorý využíva vlastnú databázu vzorov. Druhou, čiastočne oddelenou časťou, sú *Honeypot* systémy, ktoré majú za úlohu získavať znalosti o odhalených útokoch. Extrahované dáta sú ďalej poslané AIPS AP, riadiacemu ďalšiu analýzu a vyhodnotenie získaných dát. Hlavným zdrojom dát, vstupujúcich do systému, je tiež duplikovaná, zrkadlená komunikácia, ktorá je predspracovaná rozdelením na jednotlivé sieťové toky, slúžiace na vyextrahovanie behaviorálnych pravidiel pre detekčný systém.

Princíp funkcie spočíva v tom, že *Honeypot* systémy sa snažia nalákať útočníka, aby zaútočil na niektorú z inštalovaných dostupných zraniteľných služieb, ktoré na *Honeypot* systéme bežia, pričom tieto systémy obsahujú mechanizmy, ktoré umožňujú odhaliť, že došlo k využitiu zraniteľného miesta, napríklad *pretečeniu pamäti*, ak program zapíše mimo pridelenú pamäť. Informácie o útoku sú následne zaslané do AIPS AP, ktorý na základe týchto znalostí doplnených o extrahované dáta zo sieťovej komunikácie, vytvorí popis správania daného útoku. Výhodou tohto riešenia, na rozdiel od doposiaľ používaného prístupu, založeného na definovaní legitímneho správania sa systému, popísaného v kapitole 1.2.1, je to, že nemusíme vytvárať model korektného správania systému, ale uchovávané sú naučené znalosti, ktoré predstavujú vzory správania sa jednotlivých útokov. Podrobnosti o architektúre a princípe fungovania systému boli čerpané najmä z [3] a [2].

Z pohľadu tejto práce je najpodstatnejšou časťou IDPS, realizujúci detekciu nelegálnej komunikácie v reálnom čase. V konkrétnej implementácii AIPS systému je ako nosný de-



Obrázok 4.2: Architektúra sieťovej časti AIPS systému [3]

tekčný mechanizmus použitý IDS *Snort* [2], ktorý bol používaný aj pri vlastnom testovaní vo virtuálnej sieti. Z tohto dôvodu neboli vykonané testy priamo voči AIPS, pretože by výsledky detekcie odpovedali už realizovaným testom, testovanie by bolo teda redundantné. Na druhej strane, ak by bol obfuskovaný útok pomocou knižnice útočníkom vedený na Honey-pot systém, útok by putoval z proxy modulu po vnútornej sieti k cieľovému počítaču už v pôvodnej dekódovanej podobe, pretože inak by nebolo možné útok úspešne vykonať. Na Honey-pot systéme by bol preto útok samozrejme odhalený, pretože detekcia je v tomto prípade postavená primárne na pokročilých mechanizmoch *kontroly prístupu do pamäte* a podobne. V tomto aspekte neexistuje reálna možnosť zamaskovania útoku obfuskáciou sieťovej komunikácie voči Honey-pot systému tak, aby bol útok stále úspešne dokončený.

Obfuskačná knižnica pri testovaní voči Honey-pot systému môže byť využitá maximálne na sťaženie vygenerovania behaviorálneho pravidla, pretože komunikácia spôsobujúca nelegálnu operáciu na cieľovom systéme, je z vonkajšej do vnútornej siete prenášaná v maskovanej podobe a v prípade použitia šifrovania môže dôjsť k problémom pri vytváraní vzorov správania pre odchytené útoky. Modifikáciu útokov je tiež možné vykonať modifikovaním hlavičky, čo sťažuje porovnanie so vzormi správania. Nevýhodou ale v tomto prípade je, že útok sa v rámci Honey-potu javí ako vedený z vnútornej siete, čo môže viesť k rýchlemu odhaleniu proxy modulu distribuovaného v chránenej sieti. Uvedené teórie sú však čiastočne nad rámec tejto práce, pretože knižnica sa zameriava na vyhnutie sa detekcii primárne voči IDS systémom.

4.5 Zhrnutie získaných výsledkov

Táto kapitola sa zameriavala na podrobný popis metodiky a výsledkov testovania vytvorenej obfuskačnej knižnice, ktorá sa zameriava na vyhnutie sa detekcii IDS systému. Metodika a výber jednotlivých testov sa snaží poukázať na univerzálnosť a variabilitu knižnice, ktorá umožňuje obfuskovať či už legitímnu komunikáciu, ale aj komunikáciu obsahujúcu dáta, ktoré majú za úlohu vykonať nelegálnu akciu na cieľovom počítači. Testy boli teda zvolené so snahou *maximálnej diverzity* prenášanej komunikácie tak, aby princípy obfuskácie boli odlišné, čo bolo podporené aj využitím oboch súčasne rozšírených platforiem pri testovaní, konkrétne *Windows* a *Linux*, ktoré boli použité na cieľových stanicách pre rôzne útoky.

Pri vytváraní a testovaní systému bolo samozrejme vykonaných množstvo iných útokov, z ktorých boli do tejto práce vybrané tie najkomplexnejšie, pretože väčšina ďalších testov mala podobný priebeh. Pre dodatočné vyladenie aplikácie s cieľom maximálneho maskovania protokolu bola komunikácia analyzovaná aj programom *Wireshark*, keď bolo potrebné upraviť detaily komunikujúcich protokolov tak, aby sa prebiehajúca komunikácia javila ako bežná výmena dát medzi *webovým serverom* a *klientom*. Tieto úpravy boli vykonávané najmä pomocou konfiguračného súboru, konkrétne správnym použitím hlavičiek protokolu s dynamickým vkladaním údajov, na základe vlastností prenášaných dát. Záznamy komunikácie pre jednotlivé testy, uložené vo formáte *pcap* z programu *Wireshark* sa nachádzajú na priloženom CD, ktorého obsah je zosumarizovaný v prílohe B.

Úspešnosť obfuskácie voči IDS implementácii *Snort* pramení najmä z toho, že ten pri svojej detekcii využíva v hlavnej miere iba detekciu založenú na porovnávaní vzorov, pričom transformácia vykonaná knižnicou pozmení paket v takej miere, že aj v prípade prenosu v otvorenej HTTP podobe, paket vzoru neodpovedá. V prípade zašifrovania obfuskovaných dát tiež IDS nemá prakticky možnosť paket so vzorom ani porovnať. Na detekciu týchto spôsobov obfuskácie by mohli byť teoreticky úspešnejšie pokročilejšie behaviorálne analyzátory, aj keď maskovanie sa za komunikáciu prenášajúcu webový obsah, by mohlo byť efektívne aj v týchto prípadoch.

Záver

Podstatou tejto práce bolo, na základe znalostí získaných o princípoch detekcie IDS systémov a elementárnych metódach realizácie obfuskácie v súčasnosti, navrhnúť a vytvoriť knižnicu schopnú maskovať komunikáciu pred detekčným systémom. Z analýzy požiadaviek na vytváraný obfuskačný nástroj bol ako adekvátny zvolený systém, ktorý je rozdelený na dva samostatné moduly, komunikujúce medzi sebou pomocou vlastného protokolu tak, aby prenášané dáta nebolo možné treťou stranou jednoducho analyzovať. Na maskovanie boli preto zvolené v sieťach pomerne frekventované, užívateľmi hojne využívané protokoly HTTP a HTTPS, ktoré zároveň poskytujú dostatočnú mieru diverzity na vloženie obfuskovaných dát, pričom prenášané pakety budú stále navonok vyzeráť ako štandardná komunikácia s webovým serverom.

Jadro práce tvorí popis vlastného návrhu architektúry knižnice a realizácie jej významných častí. Nosnou požiadavkou návrhu knižnice bola jej univerzálnosť, v zmysle schopnosti pracovať v rôznych sieťach, rozširiteľnosť o ďalšie možnosti obfuskácie a v neposlednom rade o jednoduchosť jej použitia užívateľom. Vlastná realizácia bola preto riešená zapuzdrením fundamentálnych algoritmov, realizujúcich transformáciu dát a samotnú komunikáciu, ktoré sú prístupné iba pomocou jednoduchého programového rozhrania. Centralizovane je vykonávané aj nastavenie spôsobu realizácie obfuskácie, keď užívateľ definuje parametre v jednom konfiguračnom súbore.

Záverečná časť práce sa zaoberá testovaním vytvorenej knižnice, keď so snahou poukázať na univerzálnosť riešenia, boli vykonané rôzne experimenty na rozdielnych platformách a sieťových topológiách. Testovanie bolo rozdelené na dve kategórie, keď prebiehalo nad legitímnou a na druhej strane škodlivou komunikáciou, pričom v oboch prípadoch boli testy voči IDS *Snort úspešné* a nedochádzalo ku generovaniu žiadnych hlásení. Na základe týchto údajov môžeme realizáciu knižnice považovať za efektívnu vzhľadom na bežné nasadzované IDS systémy, avšak na druhej strane detekcii obfuskovanej komunikácie pokročilejšími nástrojmi ako sú Honeypot systémy nemožno zabrániť, ale maximálne sťažiť spätnú analýzu komunikácie, či už šifrovaním alebo modifikáciou hlavičiek prenášaných paketov.

Na úspešnú prácu s knižnicou je potrebné užívateľom implementovať dva samostatné programy, keď jeden sa bude tváriť ako webový server, pričom druhý modul musí byť distribuovaný do vzdialenej siete, odkiaľ sa komunikácia s fiktívnym webovým serverom bude inicializovať. Zavedenie modulu do vnútornej chránenej siete nie je predmetom tejto práce, pretože sa jedná o značne špecifickú úlohu, výrazne závislú na štruktúre danej siete.

Výstupom práce je experimentálna knižnica, realizujúca maskovanie komunikačných protokolov v sieti, ktorá môže byť použitá pri ďalšom vývoji detekčných algoritmov a nových prístupov odhaľovania nežiaducej komunikácie. Na druhej strane existuje možné využitie aj pri penetračnom testovaní nasadzovaných systémov, keď by mohli byť pomocou metód ponúkaných knižnicou overené schopnosti detekčného systému. Ako potenciálny spôsob použitia možno uviesť aj zneužitie útočníkom, ktorému by mohlo byť umožnené prepašovanie kompromitujúcich dát do chránenej siete.

Možnosti ďalšieho rozšírenia knižnice

Návrh knižnice poskytuje podmienky na jej ďalšie rozšírenie, keď je možné jednoduché pridanie ďalších obfuskačných modulov. Nové moduly môžu implementovať odlišné princípy analýzy odchyťovaných paketov, a tiež je možné transformovať realizáciu komunikácie medzi oboma komunikujúcimi entitami. Súčasne implementované moduly sú orientované na analýzu odchyťovanej komunikácie na úrovni *transportnej vrstvy* ISO/OSI modelu, keď užívateľ pri konfigurácii musí zadať čísla portov, ktoré budú podliehať obfuskácii.

Ako možné rozšírenie sa javí rozšírenie o analýzu aj na vyšších vrstvách sieťového modelu. Príkladom by mohla byť implementácia pokročilého modulu, zameraného na zložitejší protokol *aplikačnej vrstvy*. V tomto prípade by bolo potrebné kompletne analyzovať možné stavy, v ktorých sa komunikácia môže nachádzať a na základe charakteru daného protokolu dynamicky modifikovať filtrovanie odchyťovaných paketov. Ako zložitejší protokol môžeme považovať FTP komunikáciu, keď za behu komunikácie, na základe zvoleného režimu serveru, aktívny alebo pasívny, sa inicializuje prenos dátovým kanálom, pričom po pôvodnom kanáli dochádza iba k prenosu riadiacich informácií a príkazov. Na správnu funkcionálnosť by v týchto prípadoch bola potrebná pokročilá *stavová analýza* pre daný protokol.

Pokročilejším rozšírením knižnice sa javí aj pridanie ďalšieho protokolu, ktorý bude slúžiť na prenos obfuskovanej komunikácie medzi modulmi. Doplnkový protokol môže byť zvolený na základe analýzy siete, kde má byť obfuskačný systém nasadený, teda ak je napríklad blokovaná komunikácia protokolmi na prenos hypertextových informácií, užívateľ môže predefinovať riešenie obfuskácie na iný protokol. Dodefinovaný protokol by mal v tomto prípade byť v danej sieti frekventovaný, aby nedošlo aj k jeho zablokovaniu.

Literatura

- [1] ANDERSSON, S., CLARK, A. J. a MOHAY, G. M. Detecting network-based obfuscated code injection attacks using sandboxing. In *AusCERT Asia Pacific Information Technology Security Conference: Refereed R&D Stream*. 2005. S. 13 – 25. Dostupné na: <http://eprints.qut.edu.au/21168/1/c21168.pdf>.
- [2] BARABAS, M., DROZD, M. a HANÁČEK, P. Behavioral signature generation using shadow honeypot. *World Academy of Science, Engineering and Technology*. 2012, roč. 2012, č. 65. S. 829–833. ISSN 2010-376X.
- [3] BARABAS, M., HOMOLIAK, I., DROZD, M. et al. Automated Malware Detection Based on Novel Network Behavioral Signatures. *International Journal of Engineering and Technology*. 2013, roč. 5, č. 2. S. 249–253. ISSN 1793-8236.
- [4] BEALE, J. et al. *Snort 2.1 Intrusion Detection, Second Edition*. 2. vyd. Rockland, MA, USA: Syngress Publishing, 2004. ISBN 1-931836-04-3.
- [5] COX, M. J., ENGELSCHALL, R. S., HENSON, S. et al. *OpenSSL: The Open Source toolkit for SSL/TLS* [online]. 2013, [rev. 2013-04-25] [cit. 26. dubna 2013]. Dostupné na: <http://www.openssl.org/>.
- [6] DEBAR, H., DACIER, M. a WESPI, A. Towards a taxonomy of intrusion-detection systems. *Computer Networks*. 1999, roč. 31, č. 8. S. 805 – 822. Dostupné na: <http://www.sciencedirect.com/science/article/pii/S1389128698000176>. ISSN 1389-1286.
- [7] ERICKSON, J. *Hacking - umění exploitace*. 2. vyd. Brno, CZ: ZONER software, s.r.o., 2009. 544 s. ISBN 978-80-7413-022-9.
- [8] EXPLOITS DATABASE BY OFFENSIVE SECURITY. *Apache OpenSSL Remote Exploit (Multiple Targets)* [online]. [rev. 2003-04-04] [cit. 25. dubna 2013]. Dostupné na: <http://www.exploit-db.com/exploits/764/>.
- [9] HESSLER, A., KAKUMARU, T., PERREY, H. et al. Data obfuscation with network coding. *Computer Communications*. 2012, roč. 35, č. 1. S. 48 – 61. Dostupné na: <http://www.sciencedirect.com/science/article/pii/S0140366410004779>. ISSN 0140-3664.
- [10] HJELMVIK, E. a JOHN, W. *Breaking and Improving Protocol Obfuscation*. Sweden: Chalmers University of Technology and Göteborg University, 2010. Technical report. Dostupné na: http://www.cse.chalmers.se/~johnwolf/publications/hjelmvik_breaking.pdf.

- [11] HYNEK, J. *Genetické algoritmy a genetické programování*. 1. vyd. Praha, CZ: Grada Publishing a.s., 2008. 182 s. ISBN 978-80-247-2695-3.
- [12] JOHNSON, K. et al. *Basic Analysis and Security Engine, (BASE) project* [online]. [rev. 2009-05-28] [cit. 29. dubna 2013]. Dostupné na: [<http://base.secureideas.net/>](http://base.secureideas.net/).
- [13] KAYACIK, H., ZINCIR-HEYWOOD, A., HEYWOOD, M. et al. Generating mimicry attacks using genetic programming: A benchmarking study. In *Computational Intelligence in Cyber Security, 2009. CICS '09. IEEE Symposium*. Duben 2009. S. 136 –143. Dostupné na: [<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4925101>](http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4925101).
- [14] KAYACIK, H., ZINCIR-HEYWOOD, A. a HEYWOOD, M. Automatically Evading IDS Using GP Authored Attacks. In *Computational Intelligence in Security and Defense Applications, 2007. CISDA 2007. IEEE Symposium*. Duben 2007. Dostupné na: [<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4219095>](http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4219095).
- [15] LAROCHE, P., ZINCIR-HEYWOOD, A. a HEYWOOD, M. I. Using code bloat to obfuscate evolved network traffic. In *Proceedings of the 2010 international conference on Applications of Evolutionary Computation - Volume Part II*. Berlin, Heidelberg: Springer-Verlag, 2010. S. 101 – 110. Dostupné na: [<http://dx.doi.org/10.1007/978-3-642-12242-2_11>](http://dx.doi.org/10.1007/978-3-642-12242-2_11). ISBN 3-642-12241-8, 978-3-642-12241-5.
- [16] MCHARDY, P., KADLECSIK, J., AYUSO, P. N. et al. *The netfilter.org project* [online]. 2013, [rev. 2013-04-29] [cit. 30. dubna 2013]. Dostupné na: [<http://www.netfilter.org/>](http://www.netfilter.org/).
- [17] MOHAJERI MOGHADDAM, H., LI, B., DERAKHSHANI, M. et al. SkypeMorph: protocol obfuscation for Tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2012. S. 97–108. Dostupné na: [<http://doi.acm.org/10.1145/2382196.2382210>](http://doi.acm.org/10.1145/2382196.2382210). ISBN 978-1-4503-1651-4.
- [18] NATIONAL VULNERABILITY DATABASE. *Apache mod_ssl/Apache-SSL Buffer Overflow Vulnerability* [online]. [rev. 2008-09-10] [cit. 27. dubna 2013]. Dostupné na: [<http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2002-0082>](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2002-0082).
- [19] NATIONAL VULNERABILITY DATABASE. *BadBlue 2.72b PassThru Buffer Overflow* [online]. [rev. 2011-08-03] [cit. 30. dubna 2013]. Dostupné na: [<http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-6377>](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-6377).
- [20] NATIONAL VULNERABILITY DATABASE. *Microsoft RPC DCOM Interface Overflow* [online]. [rev. 2008-09-10] [cit. 20. dubna 2013]. Dostupné na: [<http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2003-0352>](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2003-0352).
- [21] NATIONAL VULNERABILITY DATABASE. *Samba 'call_trans2open' Remote Buffer Overflow Vulnerability* [online]. [rev. 2008-09-10] [cit. 28. dubna 2013]. Dostupné na: [<http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2003-201>](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2003-201).
- [22] NURILOV, S., SHANMUGASUNDARAM, K. a MEMON, N. Protocol Masking to Evade Network Surveillance. In *Information Sciences and Systems, 2006 40th Annual*

- Conference on*. Princeton, NJ, USA: IEEE Xplore, 2006. S. 1467–1472. Dostupné na: <<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4068037>>. ISBN 1-4244-0349-9.
- [23] PURCZYNSKI, W. *Linux Kernel 2.2.x - 2.4.x ptrace/kmod Local Root Exploit* [online]. [rev. 2003-03-30] [cit. 29. dubna 2013]. Exploits Database by Offensive Security. Dostupné na: <<http://www.exploit-db.com/exploits/3/>>.
- [24] RIBONI, D., VILLANI, A., VITALI, D. et al. Obfuscation of sensitive data in network flows. In *INFOCOM, 2012 Proceedings IEEE*. Březen 2012. S. 2372–2380. ISSN 0743-166X.
- [25] ROELKER, D. *HTTP IDS evasions revisited* [online]. 2004, rev. 21. srpen 2004 [cit. 2012-12-21]. Dostupné na: <<http://defcon.org/images/defcon-11/dc-11-presentations/dc-11-Roelker/dc-11-roelker-paper.pdf>>.
- [26] SCHIFFMAN, M. D. *Libnet 101, Part 1: The Primer*. Waltham, MA, USA: Guardent, Inc., 2000. 10 s. White Paper. Dostupné na: <<http://packetfactory.openwall.net/papers/Libnet-primer/libnet-primer.pdf>>.
- [27] THOMAS, S. A. *SSL & TLS Essentials: Securing the Web*. 1. vyd. New York, NY, USA: John Wiley & Sons Inc., 2000. ISBN 0-471-38354-6.
- [28] VERWOERD, T. a HUNT, R. Intrusion detection techniques and approaches. *Computer Communications*. Zář 2002, roč. 25, č. 15. S. 1356–1365. Dostupné na: <<http://www.sciencedirect.com/science/article/pii/S0140366402000373>>. ISSN 0140-3664.
- [29] VIGNA, G., ROBERTSON, W. a BALZAROTTI, D. Testing Network-based Intrusion Detection Signatures Using Mutant Exploits. In *Proceedings of the 11th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2004. S. 21–30. Dostupné na: <<http://doi.acm.org/10.1145/1030083.1030088>>. ISBN 1-58113-961-6.
- [30] WAGNER, D. a SOTO, P. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2002. S. 255–264. Dostupné na: <<http://doi.acm.org/10.1145/586110.586145>>. ISBN 1-58113-612-9.

Příloha A

Príklady konfiguračných súborov

HTTP metóda GET

Príklad konfiguračného súboru definuje nastavenie modulov, pri ktorom má počítač hostujúci obfuskačnú knižnicu, teda server IP adresu *172.16.0.2*, pričom samotná komunikácia s proxy modulom bude prebiehať v otvorenej podobe protokolom HTTP. Na zamaskovanie správ z proxy modulu v smere na server budú využité HTTP požiadavky GET. Tento prípad je pre pokročilejšie metódy komunikácie nie príliš vhodný, pretože sa môže stať, že hlavička správy bude v niektorých prípadoch označená za príliš dlhú.

```
# Verejná IP adresa užívateľského modulu.
server_ip = 172.16.0.2

# Definícia režimu komunikácie medzi užívateľom a proxy
# Možnosti : encrypted/plain
server_mode = plain

# Výber používaného modulu a jeho nastavenie
protocol_modul = 22,23
# scan_modul = true
destination_ip = 192.168.0.2

client_prefix = "GET "
client_suffix = " HTTP/1.1\r\nHost:172.16.0.2\r\n\r\n"

server_prefix = "HTTP/1.1 200 OK\r\n
                Content-Type: text/plain\r\n
                Content-Length: \1\r\n\r\n"
server_suffix = ""
```

HTTP metóda POST

Nasledujúci konfiguračný súbor, na rozdiel od predošlého variantu využíva pri svojej komunikácii HTTP požiadavky typu POST, keď môžu byť dáta uložené do tela paketu a nie do hlavičky. Tento prístup umožní zasielanie väčšieho objemu dát. Na dosiahnutie správneho zamaskovania je potrebné použiť *escape sekvenciu* \l, ktorá zabezpečí dynamické vkladanie veľkosti dát do hlavičky paketu.

```
# Verejna IP adresa serveru s~kniznicou.
server_ip = 172.16.0.2

# Rezim komunikacie medzi serverom a proxy - encrypted/plain
server_mode = plain

# Pouzity modul a jeho nastavenie
protocol_modul = 443
# scan_modul = true
destination_ip = 192.168.0.2

client_prefix = "POST index.html HTTP/1.1\r\n
                Host: 172.16.0.2\r\n
                Content-Type: text/plain\r\n
                Content-Length: \l\r\n\r\n"

client_suffix = ""

server_prefix = "HTTP/1.1 200 OK\r\n
                Content-Type: text/plain\r\n
                Content-Length: \l\r\n\r\n"

server_suffix = ""
```

Příloha B

Obsah CD

Zložky:

- **doc** – projektová dokumentácia, vygenerovaná zo zdrojového kódu použitím nástroja *Doxygen*.
- **examples** – vzorová implementácia riadiacich programov pre jednotlivé moduly. Okrem definície prekladu súborom *Makefile* obsahujú implementácie aj vzorový konfiguračný súbor *settings.conf*.
- **extLibs** – zdrojové kódy externých knižníc vo verziách, s ktorými prebiehalo testovanie. Knižnice sú potrebné na úspešný preklad a spustenie modulov. Inštalácia závisí najmä na cieľovej *Linuxovej* distribúcii.
- **logs** – záznamy komunikácie pri útokoch popísaných v kapitole 4.3. Súbory sú uložené v štandardnom *pcap* formáte, určené pre program *Wireshark*. Pre každý útok sú vytvorené tri záznamy, jeden pre komunikáciu priamo s cieľom a dva pre obe formy transformácie obfuskačnou knižnicou.
- **src** – zdrojové kódy knižnice v jazyku C++, obsahujúce *Makefile* na úspešný preklad do formy *statickej knižnice*, ktorú možno prilinkovať k vytváranému programu.
- **tex** – zdrojové kódy textovej časti práce vytvorené pre typografický systém *L^AT_EX*.

Súbory:

- **Readme** – súbor popisujúci základné použitie a inštaláciu knižnice.