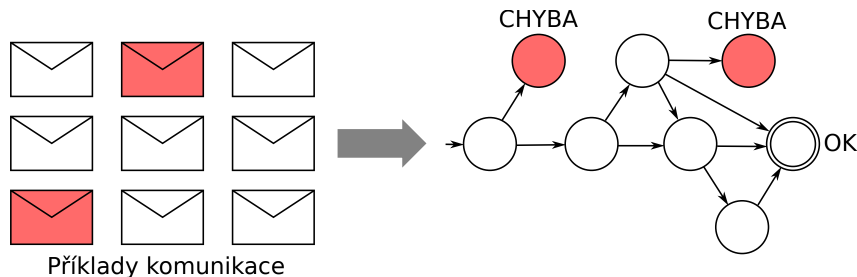
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2018
http://excel.fit.vutbr.cz

Poloautomatická diagnostika síťových protokolů

Ondřej Svoboda*



Abstrakt
Při komunikaci v počítačových sítích se používá velké množství protokolů. Pro administrátora je obtížné, aby znal do detailu všechny protokoly a uměl rozeznat všechny typy chyb, které mohou nastat. Tento článek se zabývá poloautomatickou diagnostikou síťových protokolů. Byl vytvořen nástroj, který umožní uživateli diagnostikovat neznámou komunikaci a nalézt v ní případné chyby. Aplikace s asistencí uživatele vytvoří popis protokolu ze vzorků korektní i nekorektní komunikace a poté tento popis využije při kontrole neznámé komunikace. Uživatel tak není nucen dlouze studovat normu příslušného protokolu. Nástroj je implementovaný v jazyce Python a má jak konzolovou, tak grafickou podobu, přičemž funkčně jsou totožné.

Klíčová slova: diagnostika protokolů — analýza paketů — dynamické programování — konečný automat

Přiložené materiály: *N/A*

*xsvobo0k@stud.fit.vutbr.cz, *Fakulta informačních technologií, Vysoké učení technické v Brně*

1. Úvod

V dnešní době se při komunikaci přes počítačové sítě používá celá řada protokolů. Některé protokoly jsou velice složité a pochopení jejich fungování může být zdlouhavá záležitost. Pro správce sítě může být obtížné přijít na podstatu neznámého problému v síti. Tyto problémy mohou být způsobeny chybnou konfigurací sítě, chybnou akcí uživatele anebo chybou fyzického média. V důsledku toho pak není například možná komunikace mezi dvěma uzly v síti, pakety vůbec nedorazí do svého cíle anebo je přístup uživatele odmítnut.

Pokud se tedy nějaká chyba při komunikaci přes síť vyskytne, musí se správce sítě pokusit odhalit její příčinu. Často mu nezbude nic jiného, než nastudovat normu k příslušnému protokolu, prohlížet logovací soubory anebo použít chybový výpis aplikace. Vzhledem k množství a složitosti protokolů, které se v sítích používají, může být diagnostika sítí velmi komplikovaná záležitost. K diagnostice se většinou používá specializovaný software, který zobrazí komunikaci jako posloupnost paketů a případně umožní další analýzu těchto dat.

Cílem navrhované aplikace je poskytnout uživateli poloautomatický nástroj k diagnostice síťových protokolů. To znamená nástroj, kterým lze provádět diagnostiku síťových protokolů bez širší znalosti o těchto protokolech. Tento nástroj by měl být schopný se naučit správné i chybové chování protokolu ze vzorků komunikace a poté umět určit, zda je neznámá komunikace obsahující tento protokol správná, anebo nesprávná. Pokud komunikace obsahuje chybu, měl by také umět určit jakou. Nástroj se zaměřuje na chyby vyšších síťových vrstev, jako např. špatně zadané

heslo apod. Chyby nízkých vrstev, jako např. bitové chyby, nástroj neřeší. Diagnostika je poloautomatická, protože vyžaduje od uživatele zadat jména políček identifikujících zprávy a dále při trénování chyb zadat popis chyby.

2. Stávající řešení

Aplikací částečně řešící podobný problém je Wireshark. Tento program slouží k zachycení komunikace, analýze komunikace a dokáže i detekovat některé chyby. Většinou se ale jedná o chyby protokolů nižších vrstev, jako např. chybný handshake u TCP protokolu, apod. Chyby jako například špatné heslo u POP3 příkazu PASS Wireshark nedetekuje. V takovém případě se musíme spolehnout pouze na informaci o chybě, kterou zaslal samotný protokol. Naše aplikace by měla být schopná detekovat a popsat chyby, na které byla naučena za předpokladu, že uživatel dodá při učení dané chyby i její popis. Dále by měla být schopna detekovat chyby při výskytu neznámého příkazu.

Dalším podobným řešením je nástroj ReverX [1]. Tento nástroj také vytváří popis protokolu ve formě konečného automatu a zabývá se jeho minimalizací. Neřeší ovšem obohacení popisu protokolu o chybné stavy.

Vytvářením popisu z příkladů se zabývají i v článcích [2, 3]. Pomocí genetického programování vytvářejí regulární výrazy pro textové řetězce. Vzhledem k použití genetického programování je toto řešení nedeterministické.

3. Návrh řešení

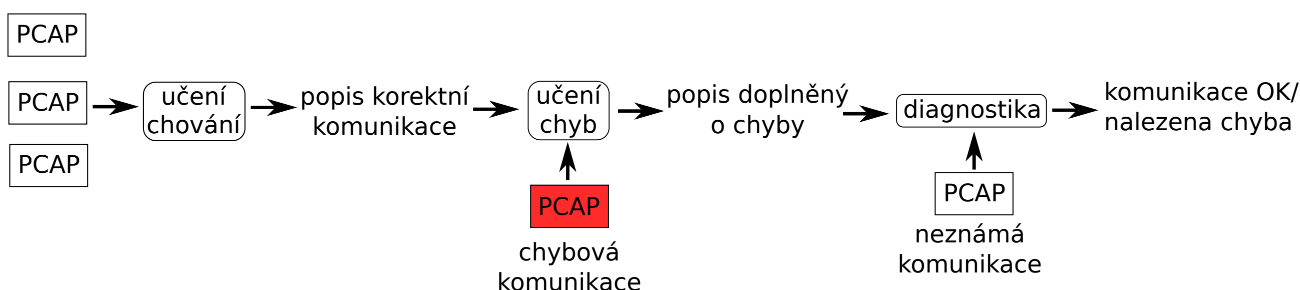
Pro diagnostiku protokolu je nutné nejprve vytvořit jeho popis, který bude obsahovat jak korektní, tak nekorektní komunikaci. Popisem protokolu rozumíme strukturu, ve které budou určitým způsobem reprezentovány možné korektní a nekorektní komunikace protokolu. Korektní komunikací rozumíme takovou, která na rozdíl od nekorektní neobsahuje žádný příkaz znamenající chybu a její příkazy jsou ve správném pořadí.

Popis fungování nástroje je vidět na obrázku 1. Nejprve natrénujeme komunikaci korektní a to pomocí sady pcap souborů obsahujících komunikaci v daném protokolu. Poté do popisu protokolu přidáme stavy reprezentující chybné zprávy v komunikaci a to opět pomocí pcap souborů, které tentokrát obsahují chybu. Tento popis poté můžeme použít pro kontrolu neznámé komunikace a to tak, že procházíme naučeným popisem protokolu podle zpráv neznámé komunikace. Pokud narazíme na stav reprezentující chybu, tak jsme v neznámé komunikaci tuto chybu nalezli. Nástroj ve svém výstupu tedy buď označí komunikaci za korektní, anebo vypíše chybu, kterou nalezl.

Vyvíjený nástroj je tedy založen na zprávách, které si v protokolu vyměňují komunikující subjekty. Pokud nástrojem TShark převedeme pcap soubor do formátu JSON, můžeme tyto zprávy identifikovat pomocí pojmenovaného políčka. Políčkem zde rozumíme klíč ve dvojici klíč-hodnota v JSON souboru. Jména těchto políček nejsou specifikována v normě protokolu, ale vycházejí z nástroje TShark. Například pro žádost u protokolu POP3 jmenuje *pop.request.command*. Jména těchto políček musí zadat uživatel a to zvlášť pro žádosti a odpovědi protokolu.

V našem případě tedy chápeme protokol pouze jako posloupnost zpráv. Předpokládáme, že komunikace probíhá stylem dotaz-odpověď, což nám umožní sjednotit dotaz a jeho odpověď do jednoho stavu. To provádíme proto, že mnohdy se odpovědi v protokolu opakují a tyto odpovědi bychom poté u některých metod sloučili do jednoho stavu, což by značně snížilo přesnost popisu protokolu. Vstupem jednotlivých metod tvorby popisu protokolu je tedy posloupnost dvojic dotaz-odpověď, kde dvojice jsou řazeny dle času jejich výskytu v komunikaci.

Jiné parametry u zpráv neřešíme a spoléháme se na to, že pro popis protokolu nejsou podstatné. Například u protokolu POP3 neřešíme u příkazu „USER" konkrétní přihlašovací jméno.



Obrázek 1. Ukázka architektury a použití nástroje.

3.1 Popis chování protokolu

Popisem protokolu může být například konečný automat, regulární výraz anebo gramatika. Z RFC daného protokolu lze většinou vyvodit ideální podobu tohoto popisu. Například u protokolu POP3 [4] jsou v RFC přímo definovány stavy, kterými protokol prochází, a dále zprávy použitelné u jednotlivých stavů. Takového popisu bychom nejraději dosáhli ve výstupu z naší aplikace.

Při tvorbě popisu se střetávají dva protiklady. Prvním je fakt, že se ze vzorků komunikace snažíme vytvořit popis co možná nejvíce obecný, neboli aby popis akceptoval i takovou korektní posloupnost příkazů, na kterou nebyl trénován. Tím ovšem vzniká problém, že můžeme vytvořit popis, který sice bude dostatečně obecný, ale bude označovat i nekorektní komunikaci za korektní. Popis by měl být také co nejvíce přesný, neboli měl by akceptovat pouze korektní komunikaci. To může mít ovšem za následek to, že popis bude akceptovat pouze ty posloupnosti zpráv, na kterých byl trénován.

3.2 Neznámé příkazy

Při používání tohoto nástroje se můžeme setkat s dvěma typy problematických příkazů. Na první typ můžeme narazit při trénování popisu protokolu a to jak korektní komunikace, tak chybové. Problémové jsou příkazy, které jsou při každé komunikaci jiné, jako například příkaz AUTH [5] u protokolu POP3. Po použití tohoto příkazu klient odesílá zakódovaný řetězec, který je při každé komunikaci jiný a v paketu se nachází ve stejném políčku jako běžné příkazy (po převedení pcapu nástrojem TShark do formátu JSON je to políčko *pop.command.request*, ve kterém bývají i ostatní příkazy klienta). Tyto řetězce tedy odfiltrujeme a to na základě jejich délky, protože bylo vysledováno, že jejich délka bývá většinou značně větší, než délka příkazů v protokolu.

Druhým typem jsou neznámé příkazy při kontrole neznámé komunikace. Tato situace nastává v momentě, kdy při kontrole narazíme na příkaz, který v současném stavu komunikace v popisu neznáme. Mohou nastat dvě možnosti: buď se jedná o příkaz, na který jsme kdekoliv jinde v komunikaci narazili a v takovém případě ho budeme považovat za chybu, protože takovou posloupnost příkazů nemáme natrénovanou, nebo může jít o příkaz zcela nový, na který jsme nikdy nenarazili. To se může lehce stát vzhledem k tomu, že většina protokolů má různá rozšíření, která přidávají nové příkazy.

Mohou nastat dvě situace, jak je možné vidět na obrázku 2. První situace je zachycena na druhém řádku, kdy byl neznámý příkaz (NOOP) vložen do jinak známé posloupnosti příkazů. Tento příkaz jednoduše vynecháme. Druhá možnost je zachycena na třetím řádku. Zde neznámý příkaz (NLIST) nahradil jinak známý příkaz v naučené komunikaci. V takovémto momentě se pokusíme neznámý příkaz přeskočit a pokračovat v kontrole dalšího příkazu a současně se posunout do dalšího možného stavu v popisu protokolu. Bylo rozhodnuto, že takto překročit můžeme pouze jeden příkaz, protože kdybychom překročili více neznámých příkazu za sebou, měli bychom již příliš mnoho možností, kam se v popisu protokolu posunout.

USER→PASS→LIST→QUIT
USER→PASS→NOOP→LIST→QUIT
USER→PASS→NLIST→QUIT

Obrázek 2. Možnosti výskytu neznámých příkazů.

3.3 Trénování chybové komunikace

Poté, co je popis protokolu natrénován na korektní komunikaci, je možné přidat do popisu komunikaci nekorektní, tzn. obsahující chyby. To se provádí s asistencí uživatele, který vždy při trénování chybové komunikace popíše chybu, která se v dané komunikaci nachází. Za chybu je poté označen takový přechod (dva po sobě jdoucí páry dotaz-odpověď), kde ze stavu odpovídajícímu prvnímu páru nemůžeme přejít do stavu odpovídajícímu druhému páru.

4. Metody tvorby popisu protokolu

Metody, pomocí kterých vytváříme popis protokolu, se liší hlavně v tom, jakým způsobem popis protokolu vytvářejí, a také v tom, jak výsledný popis vypadá. Vzhledem k tomu, že popisy vytvořené jednotlivými metodami se liší, je možné, že určitou chybu naleznou pouze některé metody a jiné ne. Kontrola neznámé komunikace je u všech metod podobná neboli čteme neznámou komunikaci po zprávách a zkoumáme, zda stejnou sekvencí zpráv můžeme projít i v popisu protokolu. Každá metoda vypisuje svůj výstup samostatně, protože se jejich výstupy mohou lišit. Tyto metody byly navrženy v rámci této práce.

Jedním z požadavků na metody bylo, aby byly zcela deterministické. To znamená, že pokud bude metoda trénována vícekrát na stejném vstupním datasetu, výsledný popis protokolu bude vždy stejný. To má za následek, že není možné u metod využít různé heuristické metody, jako například genetické programování [2, 3].

U některých protokolů začíná komunikace odpovědí, jako například u POP3, kdy server zašle na úvod odpověď „OK“. Tím pádem máme jiný počet dotazů

a odpovědí. To vyřešíme tak, že k této první odpovědi bez dotazu doplníme speciální hodnotou „NONE“, čímž srovnáme počet dotazů a odpovědí.

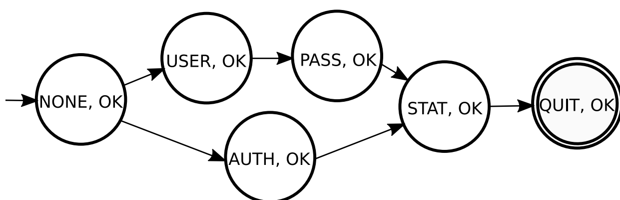
Popisy tvořeny všemi metodami podporují cykly a to buď přímo formou přechodu, anebo je to u příkazu v popisu explicitně zmíněno. Cyklus v modelu vznikne v momentě, kdy se stejný příkaz vyskytuje v trénované komunikaci více než dvakrát za sebou.

4.1 Metoda 1

První metoda funguje na principu, že pro každý nový příkaz, na který v komunikacích narazíme, vytvoříme v konečném automatu nový stav. Poté pro všechny sousední příkazy v trénované komunikaci přidáme přechod do automatu mezi stavy těchto sousedních příkazů. Výsledek jednoduchého trénování můžeme vidět na obrázku 3. Algoritmus 1 popisuje pseudokódem metodu. Vstupem je množina příkazů (commands), která obsahuje jednotlivé komunikace, neboli posloupnosti příkazů.

Algoritmus 1: Metoda 1

```
Input: (Commands)
Output: (FSM)
FSM.NewState(_start)
for i = 0 to len(commands) do
    previous = _start
    for j = 0 to len(commands[i]) do
        cmd = commands[i][j]
        if cmd not in FSM.AllStates then
            actual = FSM.NewState(cmd)
        else
            actual = FSM.GetState(cmd)
        end if
        if not FSM.Transition(previous, actual) then
            FSM.NewTransition(previous, actual)
        end if
        previous = actual
    end for
end for
```



Obrázek 3. Výsledek trénování první metody na protokolu POP3.

Výhodou této metody je to, že je poměrně obecná, tudíž dokáže akceptovat většinu korektní neznámé komunikace. Její nevýhodou je naopak to, že někdy označí i nekorektní komunikaci za korektní.

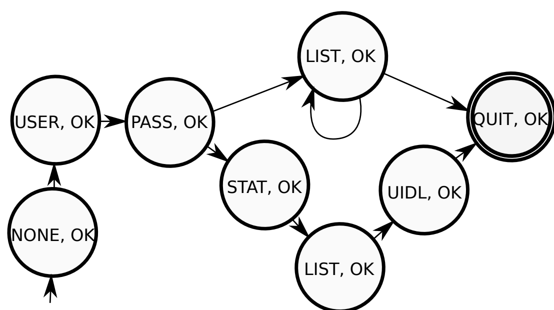
4.2 Metoda 2

Druhá metoda pracuje na principu slučování stejných podsekvencí příkazů v komunikacích. Při trénování nové komunikace si pokaždé vygenerujeme všechny možné průchody stávajícím modelem a poté hledáme nejdelší společnou podsekvenci mezi nově přidávanou komunikací a jednotlivými průchody modelem. Společné podsekvence poté sloučíme a přidáme stavy jenom pro příkazy nové komunikace, které se nepovedlo sloučit. Pro hledání nejdelších společných podsekvencí používáme algoritmus, který funguje na principu dynamického programování [6].

Algoritmus 2: Metoda 2

```
Input: (Commands)
Output: (FSM)
FSM.NewState(_start)
FSM.NewState(_end)
for i = 0 to len(commands) do
    cmds = commands[i]
    Add "_start" at beginning of cmds
    Add "_end" at end of cmds
    Remove sequences of same commands from
      cmds
    found = True
    while found do
        found = False
        Generate all possible paths through FSM
        Find the longest subsequence of cmds in one
          of the paths
        if subsequence found then
            found = True
            Remove subsequence from cmds
            Fix transitions in FSM
        end if
    end while
end for
```

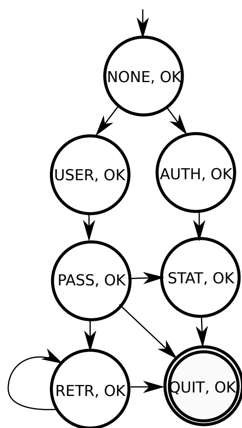
Popis vytvořený druhou metodou můžeme vidět na obrázku 4. Je méně obecný než popis vytvořený metodou první, na druhou stranu je zde ale menší šance, že tento model bude považovat nekorektní komunikaci za korektní, a to díky tomu, že se tato metoda snaží slučovat celé sekvence příkazů. Z tohoto důvodu například pro příkaz, který se může nacházet na začátku i konci komunikace, budou pravděpodobně vytvořeny dva stavy, což zvýší přesnost popisu.



Obrázek 4. Výsledek trénování druhé metody na protokolu POP3.

4.3 Metoda 3

Tato metoda je založena na předpokladu, že v korektních komunikacích se stejné příkazy budou vyskytovat přibližně ve stejný čas. Na rozdíl od předchozích metod, které přidávaly do popisu jednotlivé komunikace postupně, tato metoda bere všechny komunikace zároveň příkaz po příkazu. Výstupem této metody je popis, který má určité prvky konečného automatu a určité prvky stromu. Stejně jako v konečném automatu zde může mít uzel více než jednoho předka a stejně jako ve stromu jsou zde uzly rozděleny do úrovní. Platí také, že z uzlu můžeme přejít pouze do stejné úrovně nebo do úrovně nižší. Příklad takového popisu můžeme vidět na obrázku 5 a popis fungování metody v algoritmu 3.



Obrázek 5. Výsledek trénování třetí metody na protokolu POP3.

Tato metoda je také méně obecná než metoda první, nicméně díky omezením, které jsou dány vlastnostmi popisu protokolu, bude méně často považovat chybovou komunikaci za korektní. Problémové jsou zejména komunikace, které se výrazně liší od ostatních komunikací v trénovací množině, co se příkazů a místa jejich výskytu týká.

Algoritmus 3: Metoda 3

```
Input: (Communication sequences)
Output: (FSM)

FSM.NewState(_start)
max = max_length(Communication sequence)
for i = 0 to max do
    commands = Pick next sequence
    cmd = commands[i]
    pred = FSM.GetState(commands[i - 1])
    if Node n named as cmd on pred level then
        FSM.NewTransition(pred, n)
    else
        //We are iterating over leves below
          predecessor
        while Not at bottom do
            if Node n named as cmd exists on this
              level then
                FSM.NewTransition(pred, n)
            end if
        end while
    end if
    if We found nothing then
        //One level below pred
        FSM.NewState(cmd)
    end if
end for
```

5. Implementace a použití

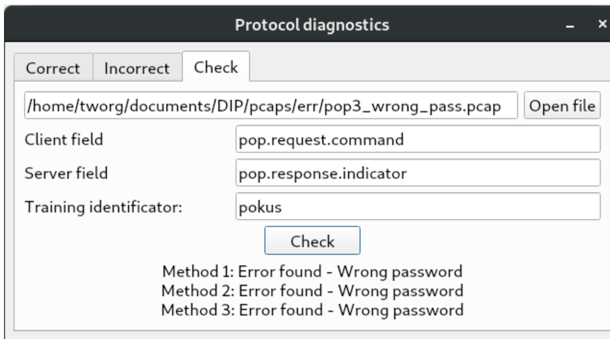
Vstupem aplikace je soubor nebo soubory ve formátu pcap, což je formát pro ukládání odchycené systémové komunikace. Tyto soubory mohou být vytvořeny například programem Wireshark, který komunikaci odchytí. Program předpokládá, že v jednom souboru bude zachycena pouze jedna kompletní komunikace, tzn. např. u protokolu POP3 posloupnost zpráv od úvodní autentizace až po příkaz „QUIT“.

Při trénování korektní komunikace uživatel zadává složku, ve které se nacházejí pcap soubory s těmito komunikacemi, zatímco při trénování nekorektní komunikace zadává pouze jeden soubor. Je to z toho důvodu, že při trénování chybné komunikace musí zadat i popis chyby, který se později zobrazí v momentě, kdy metoda nalezne tuto chybu v kontrolované komunikaci. Tento popis se také stane součástí modelu.

Program dokáže v každé komunikaci detekovat pouze jednu chybu. Je to dáno tím, že v momentě, kdy při trénování chybné komunikace narazíme na chybu, tak nemůžeme předpokládat, že zbytek komunikace je korektní.

5.1 Ukázka použití

Aplikace má dvě formy a to s grafickým uživatelským rozhraním a jako konzolová aplikace. Tyto dvě verze jsou funkčně zcela totožné, záleží pouze na uživateli, kterou zvolí. Ukázku GUI můžeme vidět na obrázku 7. Výstupy jednotlivých metod jsou zde zobrazeny zvlášť, protože jak již bylo řečeno, jejich výsledky se mohou lišit.



Obrázek 7. Ukázka grafického uživatelského rozhraní při kontrole neznámé komunikace a nalezení chyby v této komunikaci.

Natrénovaný popis vytvořený druhou metodou na 14 pcap souborech obsahujících POP3 komunikaci je vidět na obrázku 6. Korektní komunikace je v popisu znázorněna černou barvou. Červenou barvou je poté znázorněna nekorektní komunikace a jak si můžeme všimnout, pro trénování byly použity dvě nekorektní komunikace, jedna obsahující špatné heslo u příkazu „PASS“ a ve druhé klient použil příkaz „AUTH“, který server neznal. Chybové stavy jsou také stavy konečné a z tohoto důvodu můžeme detekovat maximálně jednu chybu v komunikaci, jak již bylo zmíněno dříve.

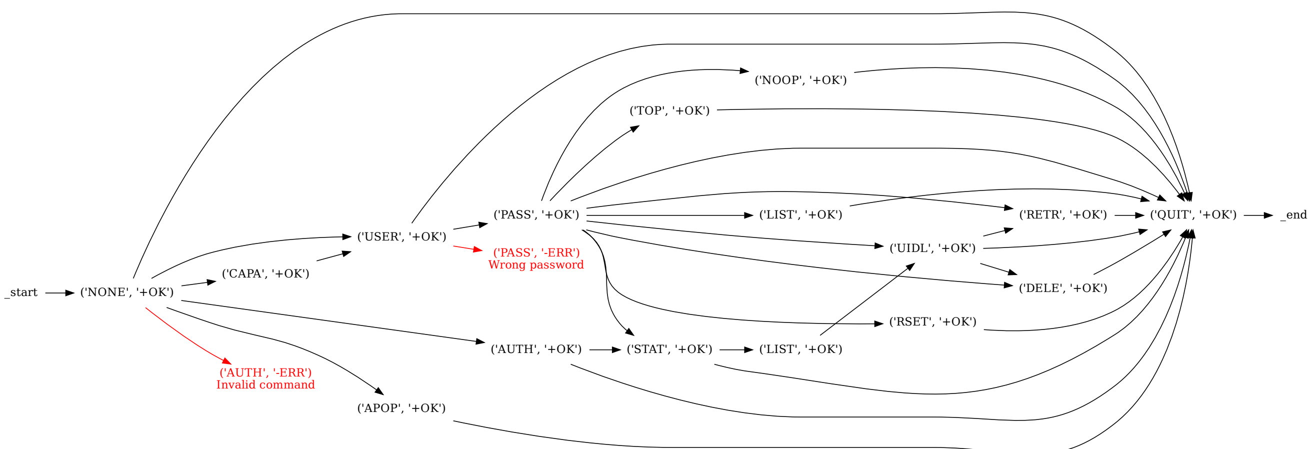
	Označeno za korektní	Označeno za nekorektní
Korektní komunikace	9	2
Nekorektní komunikace	1	2

Tabulka 1. Výsledky testování

6. Testování a experimentování

Nástroj byl důkladně testován na protokolu POP3 a částečně na protokolu SMTP. Metody jsou schopny správně vytvořit popis protokolu pro korektní i nekorektní komunikaci a dále zkontrolovat neznámou komunikaci i s neznámými stavy. Doba trénování korektní komunikace na 20 pcap souborech je cca 5.5 sekundy s tím, že asi 80 % času zabere převod souboru z formátu pcap na formát JSON programem TShark.

Experimentování s jednotlivými metodami probíhalo na protokolu POP3. K dispozici bylo 44 pcap souborů obsahujících korektní komunikaci a 11 pcap souborů obsahující nekorektní komunikaci. Obě dvě množiny byly rozděleny na trénovací a testovací podmnožiny. K trénování korektní komunikace bylo použito 33 souborů a 8 souborů nekorektní komunikace. K otestovaní funkčnosti bylo použito 11 souborů korektní komunikace a 3 soubory nekorektní komunikace. Výsledky můžeme vidět v tabulce 1. U korektní komunikace byly nesprávně za chybné označeny ty soubory, ve kterých se nacházel přechod, který nebyl zanesen v popisu. Tím pádem jsme narazili na neznámý příkaz v daném místě popisu. V případě nekorektní komunikace byl jeden soubor označen jako korektní a to z toho důvodu, že daná chyba nebyla natrénována a tím pádem byl tento chybový příkaz přeskočen.



Obrázek 6. Výsledek po trénování korektní a nekorektní komunikace

7. Závěr

Tato práce se zabývá poloautomatickou tvorbou popisu protokolů. Byly zde zobrazeny navržené podoby popisů a dále jejich vlastnosti se zaměřením na přesnost a obecnost. Dále zde bylo nastíněno řešení neznámých příkazů v komunikaci a to jak v trénovací, tak v kontrolní fázi.

V další části byly představeny tři metody tvorby popisu protokolu a jejich vlastnosti. Zabývali jsme se tím, co pro nás znamená, že metody musejí být deterministické, popsali jsme, jak probíhá kontrola neznámé komunikace a řešili jsme otázku párování dotazů a odpovědí v protokolu. U každé metody poté bylo nastíněno její fungovaní a popsán popis protokolu, který tato metoda vytváří.

Dále byl navržen a implementován nástroj, který umožňuje pomocí těchto tří metod natrénovat popis protokolu na množině pcap souborů obsahujících komunikaci v tomto protokolu a poté zkontrolovat neznámou komunikaci. S pomocí tohoto nástroje bylo provedeno testování metod, tzn. ověření, že metody jsou schopny vytvářet popisy protokolu tak, jak bylo navrženo. Také byly provedeny experimenty, kdy jsme zjišťovali, jaké chyby dokážeme odhalit a jaké nikoliv.

Další prací bude vyzkoušet nástroj a metody na dalších protokolech pro ověření funkčnosti. V plánu máme otestovat protokoly SMTP a ISAKMP.

Poděkování

Rád bych poděkoval vedoucímu své práce Ing. Martinovi Holkovičovi za jeho pomoc při tvorbě tohoto článku.

Literatura

[1] Paulo Verissimo Jo ao Antunes, Nuno Ferreira Neves. Reverx: Reverse engineering of protocols. Department of Informatics of the University of Lisbon, 2011.

[2] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. Inference of regular expressions for text extraction from examples. In *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, volume 28, pages 1217–1230, 05 2016.

[3] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. Learning text patterns using separate-and-conquer genetic programming. In *Genetic Programming EuroGP 2015*, pages 16–27. Springer International Publishing Switzerland, 04 2015.

[4] J. Myers. *Post Office Protocol - Version 3*. Network Working Group, 1996. [Online; navštíveno 23.3.2018].

[5] J. Myers. *POP3 AUTHentication command*. Network Working Group, 1994. [Online; navštíveno 6.1.2018].

[6] Dynamic programming — longest common substring, 2015. [online; navštíveno 14.1.2018].