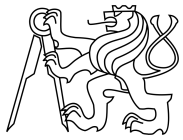
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Nástroj pro hodnocení bezpečnosti PPTP VPN
Student:	Bc. Ján Sebechlebský
Vedoucí:	Ing. Tomáš Zahradnický, Ph.D.
Studijní program:	Informatika
Studijní obor:	Systémové programování
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Seznamte se s protokolem PPTP, popište jeho základní principy. Navrhněte auditní nástroj pro testování bezpečnosti PPTP serveru tak, aby bylo jeho prostřednictvím možné získat maximum informací o testovaném serveru. Návrh koncipujte takovým způsobem, aby bylo možné z jím generovaných výpisů zrekonstruovat celý průběh testování v době jeho provádění. Implementujte navržený nástroj a ověřte ho na Vámi zvoleném PPTP serveru.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 25. října 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Teoretickej Informatiky

Diplomová práca

Nástroj na hodnotenie bezpečnosti PPTP VPN

Bc. Ján Sebechlebský

Vedúci práce: Ing. Tomáš Zahradnický, Ph.D.

9. mája 2017

Poďakovanie

Chcem poďakovať vedúcemu práce Ing. Tomášovi Zahradnickému, Ph.D. za venovaný čas, ochotu a cenné rady. Zároveň by som rád poďakoval mojej rodine a priateľom za podporu počas celého štúdia.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k užívaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo užívať. Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela (vrátane komerčného využitia). Toto oprávnenie je časovo, územne a množstevne neobmedzené.

V Prahe 9. mája 2017 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na túto prácu

Sebechlebský, Ján. *Nástroj na hodnotenie bezpečnosti PPTP VPN*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Táto práca popisuje princíp fungovania *VPN* protokolu *PPTP*. Podrobnejšie sa zaoberá autentizačnými metódami používanými pri tomto protokole. Zároveň popisuje návrh a implementáciu nástroja na bezpečnostný audit *PPTP* serverov v jazyku *Python* a rozšírenie projektu *Scapy* o podporu protokolov používaných pri *PPTP* spojeniach.

Klíčová slova PPTP, VPN, bezpečnosť, audit, scapy

Abstract

This work describes principles of *VPN* protocol *PPTP*. It focuses closely on authentication methods used with this protocol. Apart from that, the work deals with a design and implementation of the security auditing tool for *PPTP* servers implemented in *Python* programming language and addition of support for *PPTP* related protocols into the *Scapy* project.

Keywords PPTP, VPN, security, audit, scapy

Obsah

Úvod **1**

1 Point to Point Tunneling Protocol **3**
1.1 Riadiace spojenie PPTP 4
1.2 GRE 13
1.3 Point to Point Protocol 16
1.4 Autentizačné metódy PPTP/PPP 19

2 Analýza **31**
2.1 Existujúce nástroje 31
2.2 Možnosti implementácie *PPTP* 33
2.3 Scapy 34
2.4 Zhrnutie 40

3 Návrh **43**
3.1 Požiadavky 43
3.2 Rozšírenie Scapy o protokoly používané pri *PPTP* 44
3.3 Návrh nástroja PPTP-Auditor 45
3.4 Varovanie používateľa na základe výsledkov testov 48
3.5 Zhrnutie 49

4 Implementácia **51**
4.1 Implementácia chýbajúcich protokolov do Scapy 51
4.2 Implementácia nástroja PPTP-Auditor 53
4.3 Zhrnutie 60

5 Testovanie **61**
5.1 Testy pridaných protokolov v Scapy 61
5.2 Testovanie nástroja 62
5.3 Zhrnutie 66

Záver **67**

Literatúra **69**

A Zoznam použitých skratiek **73**

B Obsah priloženého CD **75**

Zoznam obrázkov

1.1 Sekvenčný diagram typického priebehu úspešného použitia *PPTP* . 5
1.2 Štruktúra hlavičky správ riadiaceho spojenia *PPTP* 6
1.3 Štruktúra inicializačných správ riadiaceho spojenia *PPTP* 7
1.4 Štruktúra správ používaných na inicializáciu *GRE*/*PPP* tunelu . . 10
1.5 Štruktúra správy *Set-Link-Info* 12
1.6 Štruktúra správ *Echo-Request* a *Echo-Reply* riadiaceho spojenia . . 12
1.7 Štruktúra správ *Call-Clear-Request* a *Call-Disconnect-Notify* riadiaceho spojenia 13
1.8 Štruktúra modifikovanej hlavičky protokolu *GRE* 14
1.9 Štruktúra správ *LCP* 17
1.10 Štruktúra správ *LCP-Option* 17
1.11 Štruktúra správ *LCP-Echo* 19
1.12 Štruktúra správ *LCP-Auth-Method-Option* 19
1.13 Štruktúra správ autentizačného protokolu *PAP* 20
1.14 Štruktúra správy *CHAP-Challenge*/*Response* 21
1.15 Štruktúra správ *EAP* 26
1.16 Sekvenčný diagram nadväzovania *TLS* spojenia 28
1.17 Štruktúra správ *EAP-TLS-Request*/*EAP-TLS-Response* 28

Zoznam tabuliek

1.1 Prehľad riadiacich správ 6
1.2 Kódy výsledku inicializácie riadiaceho spojenia 8
1.3 Všeobecné chybové kódy riadiaceho spojenia 9
1.4 Kódy výsledku správy *Call-Disconnect-Notify* 14
1.5 Prehľad správ LCP 17
1.6 Prehľad parametrov *LCP* 18

Úvod

V kontexte počítačovej bezpečnosti sa VPN (Virtuálne súkromné siete) používajú na viacero účelov:

- Vzdialený prístup do súkromnej siete.
- Ochrana pred odpočúvaním sieťových spojení.
- Maskovanie skutočnej identity alebo lokality používateľa.

V súčasnosti existuje veľké množstvo softwaru a služieb poskytujúcich VPN, či už ide o otvorené alebo proprietárne riešenia. Pri voľbe vhodného VPN riešenia však užívatelia často nehľadia len na bezpečnosť, ale zároveň aj na pohodlie a kompatibilitu.

Práve jednoduché sprevádzkovanie a podpora pre množstvo zariadení sú dôvodmi prečo je VPN stále pomerne často realizovaná pomocou Point-to-Point-Tunneling Protokolu, napriek mnohým bezpečnostným problémom, ktoré sprevádzajú jednotlivé implementácie i protokol samotný.

Ohliadnuc od toho, že *PPTP* používa na šifrovanie zraniteľný protokol *MPPE* [19][1], mnohé z autentizačných metód používaných s *PPTP* boli prelomené a ich zneužitie je triviálne [9][14]. Je preto dôležité aby v prípade keď existuje dôvod prečo musí byť použité *PPTP* konfigurácia servera zakázala použitie týchto nebezpečných metód.

Bohužiaľ neexistuje nástroj ktorý by umožnil pohodlne skontrolovať nastavenie vzdialeného *PPTP* serveru. Preto je pri bezpečnostnom audite potrebné skontrolovať nastavenia priamo na serveri (čo môže vyžadovať extra povolenie prístupu k VPN serveru) alebo zdĺhavé ručné testovanie jednotlivých autentizačných metód.

Cieľom tejto práce je vytvoriť nástroj, ktorý umožní jednoduché a rýchle získanie informácií o testovanom *PPTP* serveri.

[1] Microsoft® Point-to-Point encryption

Tento nástroj bude zároveň podporovať zaznamenávanie komunikácie prostredníctvom logovania vykonávaných akcií a záznamu sieťovej prevádzky medzi testovacou stanicou a cieľovým serverom. Tým poskytne dostatok informácií pre dodatočnú analýzu konfigurácie serveru v čase testu.

Práca je rozdelená do štyroch kapitol. Prvá kapitola sa venuje popisu sieťových protokolov ktoré sú využívané pri *PPTP VPN* spojeniach. Tie zahŕňajú protokol riadiaceho *TCP* spojenia, protokol *GRE* poskytujúci zapuzdrenie *PPP* komunikácie v *IP* sieti a protokol *PPP*. Ďalej táto kapitola podrobnejšie rozoberie jednotlivé autentizačné metódy *PPP* protokolu a rozširujúce autentizačné metódy protokolu *EAP*.

Druhá kapitola stručne popisuje dostupné nástroje na analýzu *PPTP*. Následne sa zaoberá výberom vhodnej technológie na implementáciu nástroja na testovanie zabezpečenia *PPTP* serveru. Stručne popisuje knižnicu Scapy ktorá umožňuje jednoduchú manipuláciu s paketmi na nízkej úrovni v jazyku Python.

Tretia kapitola začína popisom rozšírení ktoré je potrebné implementovať pre návrh nástroja *PPTP Auditor*. Následne rozoberá návrh samotného nástroja a popisuje fungovanie *Scapy* automatov, ktoré sú jeho kľúčovou súčasťou.

Štvrtá kapitola v úvode popisuje implementáciu zmien do knižnice *Scapy*. Následne popisuje implementáciu kľúčových častí nástroja *PPTP Auditor*v jazyku *Python*.

Posledná kapitola popisuje automatické testovanie pridaných *Scapy* vrstiev prostredníctvom nástroja *UTScapy* a testovanie nástroja oproti *PPTP* serveru *PoPToP* a PPTP serveru v operačnom systéme *Windows Server*.

Kapitola 1

Point to Point Tunneling Protocol

Protokol PPTP bol vyvíjaný organizáciou *PPTP Forum*, ktorej členmi boli organizácie Ascend Communications, U.S. Robotics, 3COM Corporation, Microsoft Corporation a ECI Telematics. Cieľom bolo získať protokol poskytujúci virtuálnu privátnu sieť medzi vzdialenými používateľmi a sieťovými servermi [3].

V čase, kedy tento protokol vznikal, bolo bežnou praxou používať na vzdialené pripojenia vytáčané analógové spojenie. V privátnej sieti ktorá umožňovala vzdialený prístup typicky existoval server vzdialeného prístupu[2], ktorý disponoval sériou modemov. Vzdialený užívateľ pri pokuse o pripojenie vytočil pomocou svojho modemu číslo modemu serveru vzdialeného prístupu. Následná autentizácia, riadenie spojenia i samotný prenos dát prebiehali za využitia protokolu *PPP*[3] [8].

Cieľom protokolu *PPTP* bolo umožniť "pretunelovať"protokol *PPP* prostredníctvom (verejnej) *IP* siete, tak aby nebolo nutné priame vytáčané pripojenie medzi užívateľom a serverom vzdialeného prístupu.

Dokument *RFC2637* [6] definuje protokol *PPTP* medzi dvoma účastníkmi - tzv. koncentrátorom *PPTP* prístupu[4] a *PPTP* sieťovým serverom[5].

Koncentrátorom *PPTP* prístupu sa typicky označovalo zariadenie disponujúce modemom, ktoré pomocou protokolu *PPTP* transparentne tunelovalo existujúce *PPP* spojenie na *PPTP* sieťový server pomocou *IP* siete. Protokol umožňuje nadväzovanie spojenia z obidvoch strán, v moderných implementáciách *PPTP* však spojenie nadväzuje výhradne klient v roli koncentrátora *PPTP* prístupu. Pre zjednodušenie terminológie bude v práci používaný ter-

[2]RAS - Remote Access Server
[3]Point to Point Protocol
[4]PAC - PPTP Access Concentrator
[5]PNS - PPTP Network server

mín *PPTP* klient v zmysle koncentrátoru *PPTP* prístupu a *PPTP* server v zmysle sieťového *PPTP* serveru.

Vďaka jednoduchému nasadeniu sa protokol *PPTP* stal veľmi obľúbeným pri implementácií *VPN* sietí. Aj napriek rozsiahlej kritike a množstvu bezpečnostných problémov je natívne podporovaný širokou škálou zariadení a operačných systémov.

Protokol sa skladá zo špecifikácie riadiaceho *TCP* spojenia a *GRE* zapuzdrenia. Riadiace spojenie slúži na vytvorenie a riadenie *PPP* tunela pomocou zapuzdrenia do modifikovaného protokolu *GRE*[6] [6].

Typický príklad priebehu *PPTP* VPN spojenia je možné vidieť na obrázku 1.1. Komunikácia začína nadviazaním *TCP* spojenia (*PPTP* server typicky očakáva spojenia na porte 1723) a inicializáciou riadiaceho spojenia. Po inicializácii riadiaceho spojenia klient požiada o vytvorenie *GRE* tunelu. Pri úspešnom potvrdení vytvorenia *GRE* tunelu obdrží číslo volania (*call-id*), ktoré identifikuje *GRE* tunel. Následne sa prostredníctvom vzniknutého tunelu pomocou protokolu *PPP LCP* dohodnú parametre *PPP* spojenia a autentizačný protokol. Odoslaním správy *Set-Link-Info* pomocou riadiaceho spojenia naberajú dohodnuté parametre spojenia platnosť a klient sa autentizuje pomocou zvoleného protokolu. Po úspešnej autentizácií sa vyjedná používané šifrovanie a kompresia prostredníctvom *PPP CCP* protokolu a prostredníctvom *PPP IPCP* parametre *IP* pripojenia vo vzdialenej sieti. Nasleduje prenos dát pomocou protokolov vyšších vrstiev zapuzdrených v *GRE/PPP* tuneli.

V nasledujúcich sekciách podrobnejšie popíšeme vybrané súčasti *PPTP*, využívané pri moderných scenároch využitia protokolu *PPTP*, ktoré nezahŕňajú vytáčané pripojenie.

1.1 Riadiace spojenie PPTP

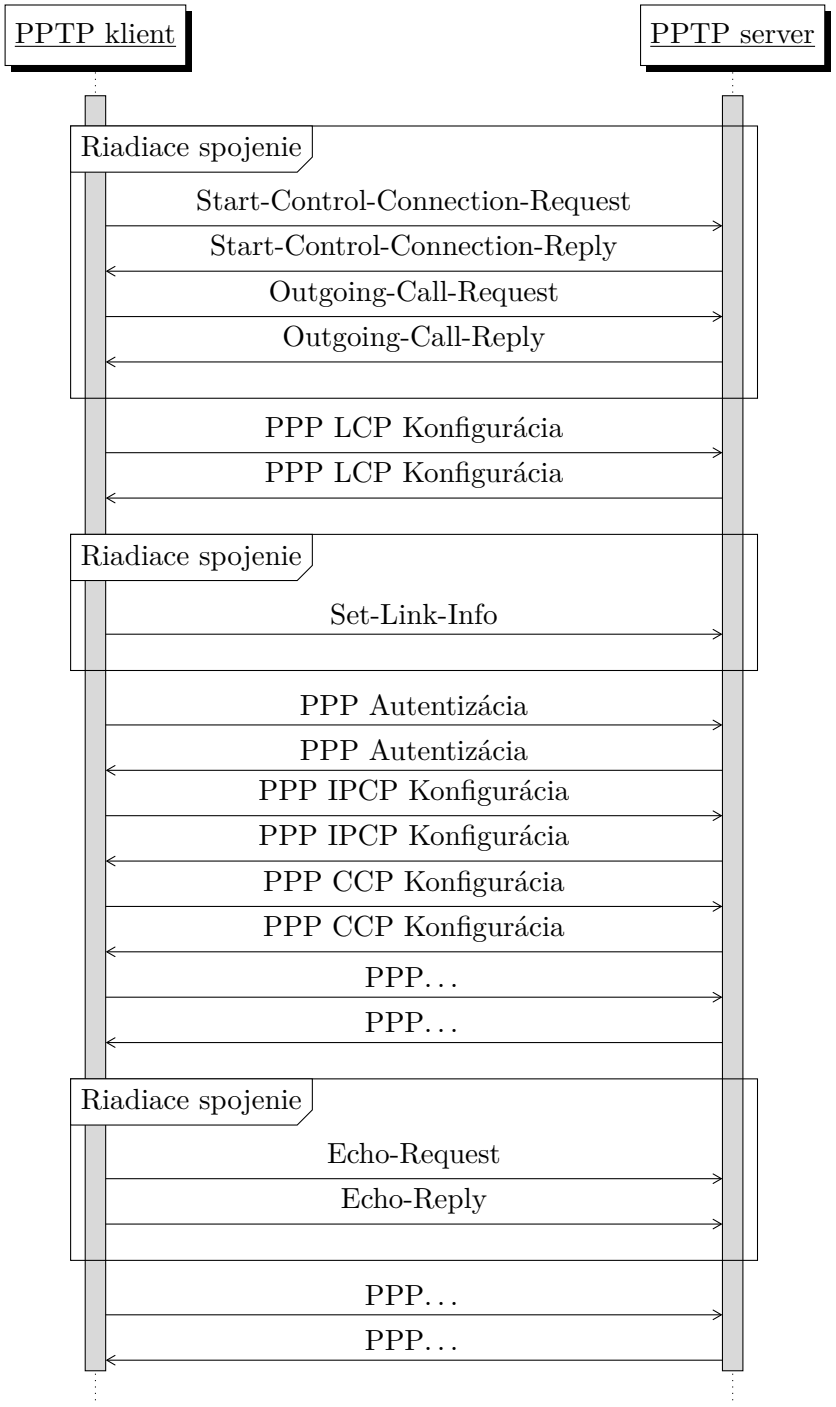
Prostredníctvom riadiaceho spojenia sa nadväzujú, riadia a rušia *PPP* sedenia prenášané prostredníctvom *GRE* tunelu, je preto potrebné ho vytvoriť ako prvé. Každý tunel musí mať priradené vlastné riadiace spojenie, pri vytváraní viacerých tunelov je teda potrebné pre každý nadviazať samostatné riadiace spojenie.

Dokument *RFC2637* [6] definuje že spojenie môže byť nadviazané ktoroukoľvek stranou. V súčasných implementáciách využívajúcich výhradne spojenie prostredníctvom *IP* siete však spojenie nadväzuje klient a server typicky očakáva klientské spojenia na porte 1723.

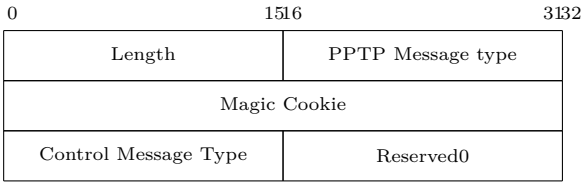
Každá správa riadiaceho spojenia musí začínať hlavičkou ktorej štruktúru je možné vidieť na obrázku 1.2. Význam jednotlivých častí hlavičky je nasledovný:

- **Length** - Dĺžka celej správy vrátane hlavičky.

[6] Generic routing encapsulation



Obr. 1.1: Sekvenčný diagram typického priebehu úspešného použitia *PPTP*

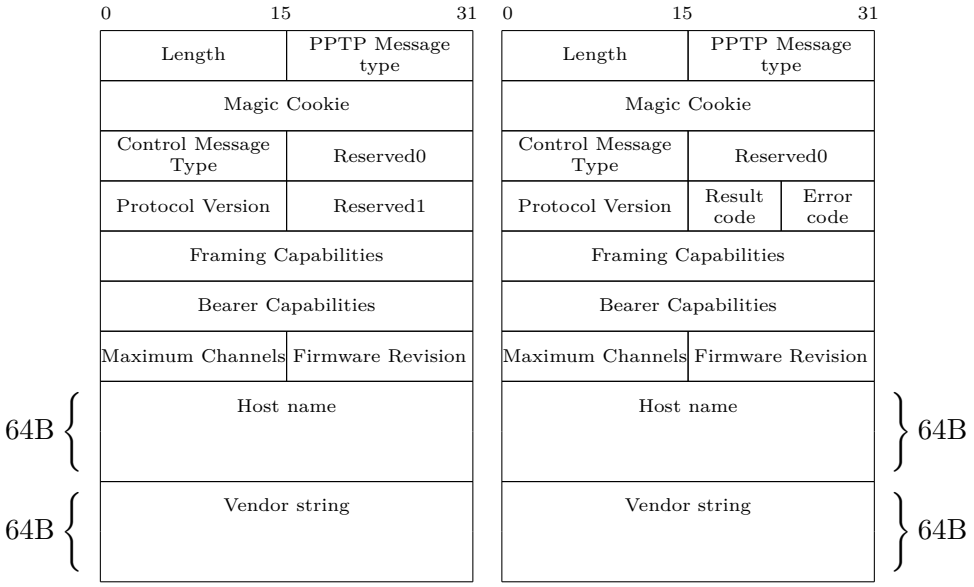


Obr. 1.2: Štruktúra hlavičky správ riadiaceho spojenia *PPTP*

Tabuľka 1.1: Prehľad správ riadiaceho *PPTP* spojenia

Riadiaca správa	Control Message Type
Start-Control-Connection-Request	1
Start-Control-Connection-Reply	2
Stop-Control-Connection-Request	3
Stop-Control-Connection-Reply	4
Echo-Request	5
Echo-Reply	6
Outgoing-Call-Request	7
Outgoing-Call-Reply	8
Incoming-Call-Request	9
Incoming-Call-Reply	10
Incoming-Call-Connected	11
Call-Clear-Request	12
Call-Disconnect-Notify	13
WAN-Error-Notify	14
Set-Link-Info	15

- **PPTP Message Type** - Typ PPTP správy. *RFC2637* [6] definuje 2 typy správ:
  - *Control Message*(1) - Riadiaca správa.
  - *Management Message*(2) - Správy tohoto typu neboli do súčasnosti definované a teda sa nepoužívajú.
- **Magic Cookie** - Konštantná hodnota 0x1a2b3c4d, slúži na overenie že ide o správu *PPTP* riadiaceho spojenia.
- **Control Message Type** - Typ riadiacej správy, jednotlivé typy správ s príslušnými hodnotami sú uvedené v tabuľke 1.1.
- **Reserved0** - Táto časť hlavičky bola rezervovaná na využitie novšími verziami protokolu. V súčastnosti neexistuje verzia protokolu, ktorá by túto časť hlavičky využívala a je vyžadované aby mala hodnotu 0.



(a) Start-Control-Connection-Request (b) Start-Control-Connection-Reply

Obr. 1.3: Štruktúra inicializačných správ riadiaceho spojenia *PPTP*

1.1.1 Inicializácia riadiaceho spojenia

Bezprostredne po nadviazaní *TCP* spojenia je potrebné riadiace spojenie inicializovať výmenou správ *Start-Control-Connection-Request* a *Start-Control-Connection-Reply*, štruktúru ktorých je možné vidieť na obrázkoch 1.3a a 1.3b. Správu *Start-Control-Connection-Request* odosiela strana ktorá nadviazala TCP spojenie. Táto správa obsahuje okrem hlavičky tiež nasledujúce informácie:

- **Protocol version** - Hodnota je tvorená 2 bajtami z ktorých vyšší označuje číslo verzie protokolu a nižší číslo revízie protokolu, ktorú si odosielateľ praje používať.
- **Reserved1** - Dva bajty, ktoré musia byť v nastavené na hodnotu 0.
- **Framing capabilities** - Jednotlivé bity určujú typy rámcov *HDLC*[7] podporovaných odosielateľom. Pri *PPTP* používanom len prostredníctvom *GRE* tunelu cez *IP* sieť je tento parameter irelevantný. V rámci *GRE* sa používajú rámce *HDLC* bez kontrolného súčtu a akéhokoľvek escapovania.
- **Bearer capabilities** - Jednotlivé bity určujú typ podporovaných kanálov. Ide o 32 bitové pole, kde je definovaný význam len bitov s hodnotami 1 (analógový nosič) a 2(digitálny nosič).

[7]High-Level Data Link Control

Tabuľka 1.2: Význam kódov výsledku inicializácie riadiaceho spojenia

Kód výsledku	Význam
1	Riadiace spojenie bolo úspešne inicializované.
2	Pri riadiacom spojení nastala chyba ktorej konkrétny typ je špecifikovaný hodnotou Error code.
3	Riadiace spojenie už bolo nadviazané.
4	Odosielateľ požiadavky nemá oprávnenie na nadväzovanie spojenia.
5	Vyžadovaná verzia protokolu nie je podporovaná.

- **Maximum channels** - Maximálny počet simultánnych *PPP* sedení, ktoré podporuje odosielateľ správy. Pokiaľ bolo spojenie nadväzované zo strany *PPTP* sieťového serveru, je vyžadované aby bolo toto číslo 0. Pokiaľ je nadväzované zo strany koncentrátora, ktorý disponuje sériou modemov, môže toto číslo zodpovedať počtu modemov. V prípade použitia *PPTP* výhradne v *IP* sieti, je táto hodnota záležitosťou implementácie klienta.

- **Firmware revision** - Verzia firmwaru (v súčastnosti častejšie verzia softwarového klienta) odosielateľa.

- **Host name** - Doménové meno odosielateľa, zarovnané bajtmi s hodnotou 0 na dĺžku 64 bajtov.

- **Vendor string** - Textový reťazec určený s popisom odosielateľa (výrobca, názov softwaru a pod.). Reťazec musí byť rovnako ako doménové meno zarovnaný na dĺžku 64 bajtov nulami.

Protistrana (*PPTP* server) odpovedá na prijatie správy *Start-Control-Connection-Request* takmer identickou správou *Start-Control-Connection-Reply*. Namiesto poľa dvoch vyhradených bajtov označených v požiadavke ako *Reserved1*, obsahuje odpoveď:

- **Result code** - kód výsledku, indikuje úspešnosť pokusu o inicializáciu riadiaceho spojenia. Kód výsledku je hodnota z rozsahu 1-5. Význam jednotivých hodnôt je uvedený v tabuľke 1.2:

- **Error code** - Označuje všeobecný chybový kód. V prípade že je hodnota kódu výsledku 2, určuje tento kód konkrétny typ chyby, ktorá nastala. Možné hodnoty sú uvedené v tabuľke 1.3. Dané chybové kódy sú používané všetkými správami PPTP riadiaceho spojenia, ktoré môžu vracať chybový kód.

Zo správ *Start-Control-Connection-Request* a *Start-Control-Connection-Reply* je možné vyčítať základné informácie o *PPTP* serveri. V niektorých

Tabuľka 1.3: Všeobecné chybové kódy riadiaceho spojenia

Chybový kód	Označenie chyby	Význam
0	None	Žiadna chyba.
1	Not-Connected	Riadiace spojenie nebolo inicializované.
2	Bad-Format	Odoslaná správa neobsahuje správnu hodnotu Magic cookie, alebo nesedí uvedená dĺžka správy.
3	Bad-Value	Niektoré z polí obsahuje hodnotu mimo povolený rozsah.
4	No-Resource	Nedostatok prostriedkov na obsluhu požiadavky.
5	Bad-Call-Id	Nesprávne číslo volania.
6	PAC-Error	Všeobecná chyba na strane koncentrátora PPTP spojení (PPTP klienta).

prípadoch ich je možné použiť na identifikáciu operačného systému pod ktorým server beží. Implementácia *PPTP* serveru Poptop[8] používaná na linuxových systémoch odosiela ako doménové meno vždy reťazec 'host' a ako vendor string reťazec 'linux'. Podľa toho je možné ju jednoducho identifikovať. Staršie implementácie v *Microsoft Windows*, odosielajú v poli *firmware_version* číslo verzie podľa ktorého je možné určiť verziu *Microsoft Windows*, na ktorom server beží [15].

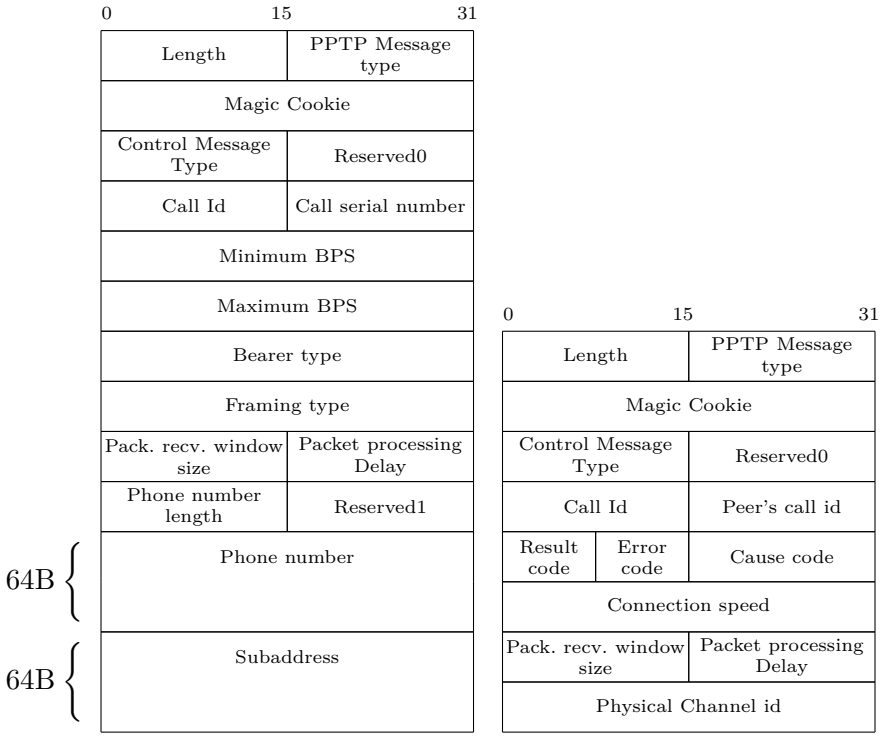
1.1.2 Inicializácia GRE/PPP tunelu

Po úspešnej inicializácií riadiaceho spojenia, pri scenári, ktorý nezahrňuje vytáčané pripojenie, odošle klient serveru správu *Outgoing-Call-Request*, prostredníctvom ktorej žiada o vytvorenie *GRE/PPP* tunelu.

Táto správa obsahuje nasledujúce informácie:

- **Call Id** - Identifikátor tunelu zvolený klientom, asociovaný s vytváraným sedením. Server tento identifikátor použije na označenie *GRE* paketov určených klientovi, ktoré patria k vytváranému spojeniu. Identifikátor musí byť unikátny pre dvojicu klient-server.
- **Call serial number** - Identifikátor spojenia pre účely logovania. Narozdiel od `Call id` je tento identifikátor zdieľaný klientom i serverom.
- **Minimum BPS** - Minimálna akceptovaná rýchlosť spojenia, uvedená v bitoch za sekundu.
- **Maximum BPS** - Maximálna akceptovaná rýchlosť spojenia, uvedená v bitoch za sekundu.
- **Bearer type** - Typ nosiča, povolené hodnoty sú analógový(1), digitálny(2), akýkoľvek(3).

[8] `http://poptop.sourceforge.net/`



(a) Outgoing-Call-Request (b) Outgoing-Call-Reply

Obr. 1.4: Štruktúra správ používaných na inicializáciu *GRE/PPP* tunelu

- **Framing type** - Typ HDLC rámcov, povolené hodnoty sú synchrónne(1), asynchrónne(2), akékoľvek(3).

- **Pack. recv. window size** - Verzia protokolu *GRE* využívaná v *PPTP* používa tzv. okienkový protokol. Toto pole udáva šírku okna, ktorú si klient praje používať a predstavuje maximálnu dĺžku sekvencie *GRE* paketov, ktoré budú odoslané serverom bez prijatého potvrdenia o prijatí.

- **Packet processing Delay** - Čas spracovania maximálneho množstva dát (závislého na veľkosti okna) v bufferi. Uvádza sa v desatinách sekundy, protokol nešpecifikuje ako sa táto hodnota určí. Moderné implementácie toto pole ignorujú.

- **Phone number length**, **Phone number**, **Subaddress** - Dĺžka telefónneho čísla a telefónne číslo. Správa Outgoing-Call-Request pri použití so serverom prostredníctvom vytáčaného pripojenia slúžila na vyžiadanie spojenia prostredníctvom modemu. Pri pripojení pomocou siete IP sú tieto polia nastavené na hodnotu 0.

Na požiadavku *Outgoing-Call-Request* odpovedá server správou *Outgoing-Call-Reply*, v ktorej zasiela:

- **Call Id** - Identifikátor tunelu zvolený serverom, asociovaný s vytváraným spojením. Klient tento identifikátor použije na označenie *GRE* paketov určených pre server, ktoré patria k vytváranému spojeniu. Rovnako ako klientský identifikátor, musí byť unikátny pre dvojicu klient-server.

- **Peer's call id** - Identifikátor tunelu protistrany (klienta), slúži na spárovanie požiadavky s odpoveďou.

- **Result code** - Kód výsledku. Nadobúda hodnotu 1 v prípade, že tunel bol úspešne inicializovaný. Pokiaľ nastala chyba nadobúda hodnotu 2. Chybové kódy 3-7 indikujú problémy s vytáčaním požadovaného čísla.

- **Error code** - V prípade že *Result code* je 2, obsahuje toto pole číslo chyby, tak ako je uvedené v tabuľke 1.3.

- **Cause code** - Dodatočné informácie o chybe vytáčania, v prípade *ISDN* pripojenia obsahuje kód *ISDN* chyby podľa štandardu *Q.931*.

- **Connection speed** - Rýchlosť spojenia podporovaná serverom v bitoch za sekundu.

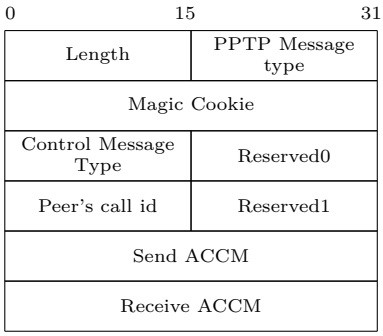
- **Pack. recv. window size** - Maximálna veľkosť okna, ktorá môže byť použitá klientom.

- **Packet processing delay** - Čas spracovania maximálneho množstva dát (závislého na veľkosti okna) v bufferi, uvedený v desatinách sekundy. Protokol nešpecifikuje ako sa táto hodnota určí. Moderné implementácie toto pole ignorujú.

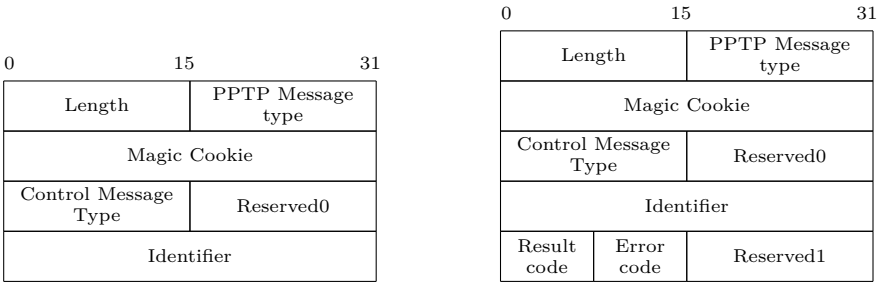
- **Physical channel id** - Protokolom nešpecifikované *ID* fyzického kanálu pre logovacie účely.

Po prijatí *Outgoing-Call-Request* s kódom výsledku 1 je možné začať komunikáciu pomocou *GRE* tunela označeného dohodnutými číslami spojenia. Parametre *PPP* spojenia sú následne dohodnuté prostredníctvom protokolu *PPP LCP* (Link Control Protocol), ktorý je popísaný neskôr. Začiatok platnosti konfigurácie je signalizovaný zaslaním správy *Set-Link-Info*(obrázok 1.5) prostredníctvom riadiaceho spojenia *PPTP* klientom *PPTP* serveru. Je však dobré spomenúť, že takáto obsluha správy *Set-Link-Info* nie je implementovaná napríklad v linuxovom PPTP serveri *Poptop* a klientovi *pptpclient*[9]. Tento fakt však pri použití *PPTP* na vytvorenie tunelu pomocou *IP* siete

[9]Zistené pomocou inšpekcie zdrojových kódov a analýzou sieťovej komunikácie



Obr. 1.5: Štruktúra správy *Set-Link-Info*



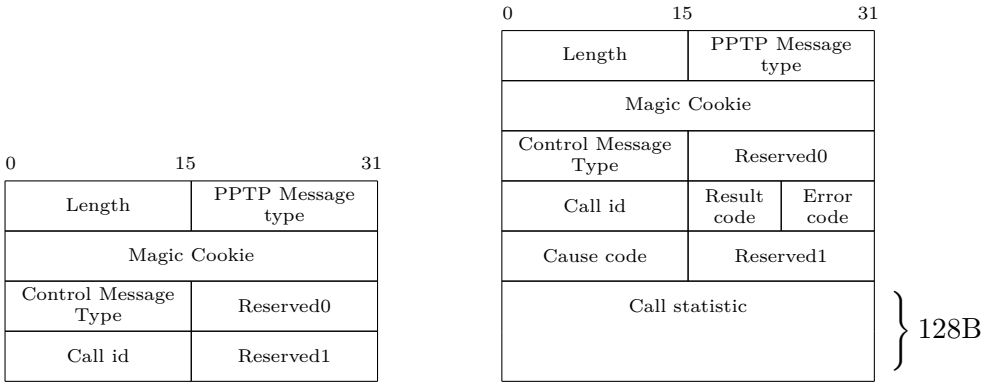
(a) Echo-Request (b) Echo-Reply

Obr. 1.6: Štruktúra správ *Echo-Request* a *Echo-Reply* riadiaceho spojenia

nepredstavuje problém, keďže hlavný význam správy *Set-Link-Info* bolo dohodnutie escapovania riadiacich znakov, ktoré pri prenose *GRE* tunelom nie je potrebné.

1.1.3 Udržiavanie riadiaceho spojenia

Dokument *RFC2637* [6] špecifikuje správy *Echo-Request* a *Echo-Reply*, ktorých štruktúru je možné vidieť na obrázku 1.6. Pokiaľ je riadiace spojenie inicializované (došlo k úspešnej výmene správ *Start-Control-Connection-Request* a *Start-Control-Connection-Reply)* a počas 60 sekúnd nebola prostredníctvom riadiaceho spojenia odoslaná žiadna správa, je potrebné odoslať protistrane spávu *Echo-Request*. Táto správa obsahuje okrem hlavičky iba identifikátor (Identifier), ktorý odosielateľ zvolí ľubovoľne. Ak počas nasledujúcich 60 sekúnd nie je prijatá správa *Echo-Reply*, spojenie sa ukončuje. Táto správa okrem identifikátoru ktorý sa musí zhodovať s identifikátorom požiadavky na ktorú odpovedá, obsahuje naviac hodnoty kódu výsledku (*Result code*) a kódu chyby (*Error code*). Tieto hodnoty reflektujú stav spojenia. Kód výsledku v správe *Echo-Reply* má hodnotu 1 v prípade, že žiadna chyba nenastala alebo 2 v opačnom prípade. Pokiaľ je hodnota kódu výsledku 2, obsahuje kód chyby nenulovú



(a) Call-Clear-Request (b) Call-Disconnect-Notify

Obr. 1.7: Štruktúra správ *Call-Clear-Request* a *Call-Disconnect-Notify* riadiaceho spojenia

hodnotu chyby podľa tabuľky 1.3.

1.1.4 Zrušenie GRE tunelu

Pri ukončení spojenia odosiela klient serveru správu *Call-Clear-Request* (obrázok 1.7a). Požiadavka obsahuje okrem hlavičky riadiacej správy identifikátor spojenia, zvolený klientom, keďže požiadavka na ukončenie môže byť odoslaná počas nadväzovania spojenia, pred prijatím správy *Outgoing-Call-Reply*.

Na požiadavku server odpovedá zaslaním *Call-Disconnect-Notify*. Dokument *RFC2637* [6] hovorí, že správa *Call-Disconnect-Notify* je zasielaná vždy pri ukončení spojenia, ktoré môže byť ukončené zo strany servera aj bez predchádzajúcej požiadavky. Linuxový *PPTP* server *PoPToP* toto správanie nerešpektuje, a po prijatí správy *Call-Clear-Request* ukončuje riadiace *TCP* spojenie. To zároveň znamená, že pre opätovné vytvorenie tunelu je potrebné nanovo naviazať a inicializovať riadiace spojenie.

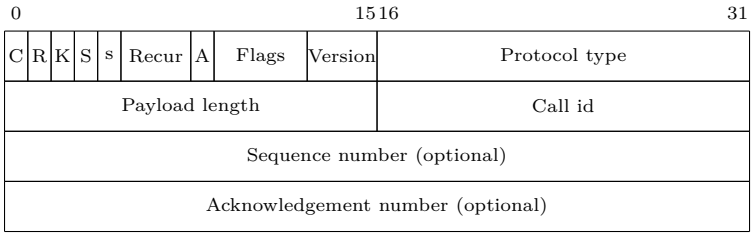
Správa *Call-Disconnect-Notify* obsahuje okrem hlavičky takisto identifikátor spojenia (*Call id*) zvolený klientom. Zároveň obsahuje kód výsledku (*Result code*), oznamujúci dôvod ukončenia spojenia (viď tabuľka 1.4). V prípade, že je kód výsledku 2, je typ chyby, ktorá nastala určený hodnotou *Error code* podľa tabuľky 1.3.

1.2 GRE

GRE (Generic Routing Encapsulation) je označenie pre tunelovací protokol vyvinutý spoločnosťou Cisco Systems, ktorého účelom je zapuzdrenie protokolov sieťovej vrstvy vo virtuálnom *PPP* spojení prenášanom cez *IP* sieť. Štandardný *GRE* protokol je popísaný v dokumentoch *RFC 2784* [4] a *RFC*

Tabuľka 1.4: Kódy výsledku správy *Call-Disconnect-Notify*

Kód výsledku	Význam
1	Strata spojenia (napríklad výpadok linky pri vytáčanom spojení)
2	Pri riadiacom spojení nastala chyba ktorej konkrétny typ je špecifikovaný hodnotou Error code
3	Ukončenie spojenia z administratívnych dôvodov (napríklad reštart servera)
4	Ukončenie spojenia v reakcii na požiadavku *Call-Clear-Request*



Obr. 1.8: Štruktúra modifikovanej hlavičky protokolu *GRE* používaná na zapuzdrenie *PPP* komunikácie do *IP* protokolu

2890 [2]. Protokol *PPTP* však používa modifinovanú verziu ktorá je definovaná v dokumente *RFC2637*[6]. Štruktúru *PPTP GRE* hlavičky je možné vidieť na obrázku 1.8. Význam jednotlivých častí je nasledovný:

- **C** - Bit indikuje prítomnosť kontrolného súčtu v *GRE* hlavičke, *PPTP GRE* hlavička kontrolný súčet nikdy neobsahuje a preto je vyžadované aby bol tento bit nastavený na hodnotu 0.
- **R** - Bit indikujúci prítomnosť informácie o smerovaní paketu. Rovnako ako v predošlom prípade, je táto funkcionalita nevyužívaná a vyžaduje sa, aby bol bit nastavený na hodnotu 0.
- **K** - Bit indikuje prítomnosť tzv. kľúča. V prípade *PPTP* je kľúč tvorený hodnotami **Payload length** a **Call id**. Tieto hodnoty musia byť prítomné v každej *PPTP GRE* hlavičke a teda tento bit musí byť nastavený na hodnotu 1.
- **S** - Bit indikujúci prítomnosť sekvenčného čísla (**Sequence number**). Pokiaľ hlavička obsahuje sekvenčné číslo, je nastavený na hodnotu 1, v opačnom prípade 0.
- **s** - Bit určujúci typ smerovacej informácie v prípade že je prítomná. Pri *PPTP* musí byť jeho hodnota vždy 0.

- **Recur** - 3 bitové číslo určujúce koľkokrát je možné paket zabaliť do ďalšej *GRE* vrstvy. Pri *PPTP* je vyžadované aby táto hodnota bola 0.
- **A** - Bit indikujúci prítomnosť čísla potvrdenia (**Acknowledgment number**).
- **Payload length** - Veľkosť zapuzdrených dát (bez GRE hlavičky) v bajtoch.
- **Call id** - Identifikátor tunelu, ktorý má prijímateľ správy asociovaný s týmto tunelom.
- **Sequence number** - Poradové číslo paketu. V prípade že odosielaný *GRE* paket obsahuje dáta (nejde iba o hlavičku s potvrdením prijatia), je prítomnosť sekvenčného čísla vyžadovaná. Prvý *GRE* paket odoslaný prostredníctvom tunelu by mal mať sekvenčné číslo 0, každý ďalší paket obsahujúci dáta sekvenčné číslo o 1 vyššie.
- **Acknowledgment number** - Číslo potvrdenia. Odoslaním *GRE* paketu s prítomným číslom potvrdenia odosielateľ potvrdzuje prijatie všetkých *GRE* paketov so sekvenčným číslom rovným alebo nižším ako je číslo potvrdenia.

Hlavným rozdielom oproti štandardnému protokolu *GRE* je zavedenie potvrdzovania prijatia paketov pomocou čísla potvrdenia v hlavičke. Účelom toho potvrdzovania však nie je znovuodoslanie stratených paketov a vytvorenie spoľahlivého potvrdzovaného spojenia (ako pri *TCP*), ale riadenie toku tunela a synchronizácia veľkosti používaných okien pri okienkovom protokole.

1.2.1 Okienkový protokol používaný v PPTP GRE

Pri nadväzovaní spojenia si klient a server vymenia maximálne veľkosti okien, ktoré môže protistrana používať pri odosielaní *GRE* paketov. Protokol tiež definuje spôsob, akým by mala každá zo strán dynamicky meniť veľkosť okna a vyhnúť sa tak zahlteniu preťaženej siete.

Na začiatku spojenia by mala každá zo strán začať s veľkosťou okna rovnou polovici maxima protistrany (ktoré pozná zo správ *Outgoing-Call-Request* a *Outgoing-Call-Reply*). Odosielanie paketov nie je prerušené, pokiaľ počet nepotvrdených paketov nepresiahne aktuálnu veľkosť okna. Vždy keď je prijatie kompletného okna potvrdené (nedojde k time-outu), zvýši odosielateľ aktuálnu veľkosť okna o 1, až pokiaľ nedosiahne maximum. V prípade, že dôjde k time-outu, je aktuálna veľkosť okna znížená na polovicu (so zaokrúhlením nahor).

V prípade time-outu protistrana nikdy opätovne nezasiela nepotvrdené pakety. Je možné, že pakety dorazia v nesprávnom poradí. Prijímateľ v tomto prípade potvrdí prijatie paketu, ktorý dorazil prvý (má vyššie sekvenčné číslo)

a pokladá paket s nižším sekvenčným číslom za stratený. V prípade, že tento paket neskôr dorazí, musí byť zahodený, keďže pakety mimo poradie môžu spôsobiť problémy v protokole *PPP*. Protokol *PPP* si totiž dokáže poradiť so stratenými paketmi, ale nie s paketmi, ktoré dorazia v opačnom poradí [23].

1.3 Point to Point Protocol

Point to Point Protocol je protokol dátovej vrstvy (druhej vrstvy *OSI/ISO* modelu) používaný na vytvorenie spojenia medzi dvoma bodmi (zariadeniami) [18]. Tento protokol je definovaný v dokumente RFC1661 [23].

Protokol *PPP* má tri hlavné komponenty:

- Metódu na zapuzdrovanie datagramov prenášaných pomocou duplexných liniek medzi dvoma bodmi.
- *Link Control Protocol* (riadiaci protokol linky), ktorý zabezpečuje nadväzovanie, konfiguráciu a testovanie spojenia.
- Skupinu *NCP*(*Network Control Protocol*) protokolov určených na nadväzovanie a konfiguráciu protokolov sieťovej vrstvy.

Pred tým ako je *PPP* spojenie pripravené na použitie protokolmi sieťovej vrstvy, musí byť inicializované pomocou nasleducúcich krokov:

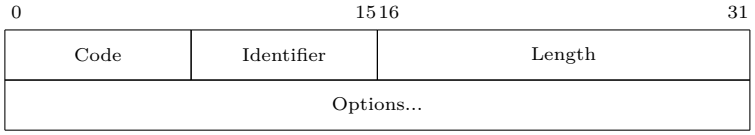
1. Vyjednanie konfigurácie pomocou protokolu *Link Control Protocol*. Po odsúhlasení konfigurácie sa spojenie považuje za otvorené, no nemusí byť pripravené na použitie sieťovou vrstvou. Súčasťou dohodnutej konfigurácie je i typ autentizačnej metódy (pokiaľ je autentizácia vyžadovaná).
2. Voliteľné overenie spojenia, prostredníctvom výmeny správ *LCP-Echo* sa detekuje zacyklenie a odhadne latencia.
3. Autentizácia pomocou autentizačného protokolu dohodnutého v prvom kroku, v prípade že bola autentizácia vyžiadaná.
4. Konfigurácia protokolov sieťovej vrstvy, pomocou zodpovedajúcich *NCP* protokolov (napríklad *IPCP* v prípade protokolu *IP*). V tejto fáze sa zároveň môže pomocou protokolu CCP (Compression Control Protocol) nakonfigurovať kompresia (prípadne šifrovanie) sieťovej vrstvy.

Úspešne inicializované spojenie môže byť následne používané protokolmi sieťovej vrstvy, klient i server však musia naďalej reagovať na správy protokolu *LCP* a prípadne i za behu zmeniť konfiguráciu spojenia.

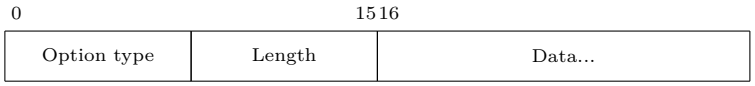
Prenášané datagramy sieťovej vrstvy a protokolov *PPP* sú zapuzdrené v *PPP* rámcoch. Obe strany môžu spojenie kedykoľvek ukončiť zaslaním správy *LCP-Terminate*. V nasledujúcej sekcii popíšeme podrobnejšie správy protokolu *LCP*.

Tabuľka 1.5: Prehľad správ protokolu LCP

LCP správa	Kód správy
Configure-Request	1
Configure-Ack	2
Configure-Nak	3
Configure-Reject	4
Terminate-Request	5
Terminate-Ack	6
Code-Reject	7
Protocol-Reject	8
Echo-Request	9
Echo-Reply	10
Discard-Request	11



Obr. 1.9: Štruktúra správ *LCP*



Obr. 1.10: Štruktúra správ *LCP-Option*

1.3.1 Link Control Protocol

Účelom protokolu *LCP* je umožniť vyjednanie konfigurácie PPP spojenia, ktorá bude vyhovovať obidvom stranám a zároveň poskytnúť prostriedky na otestovanie a ukončenie spojenia.

Prvý bajt každej *LCP* správy obsahuje kód určujúci typ správy. Prehľad *LCP* správ je možné vidieť v tabuľke 1.5.

Správy LCP-Configure

Kofigurácia spojenia prebieha pomocou výmeny správ *LCP-Configure-Request*, *LCP-Configure-Reject*, *LCP-Configure-Nak* a *LCP-Configure-Ack*. Všetky tieto správy majú rovnakú štruktúru, znázornenú na obrázku 1.9.

Prvý bajt obsahuje hodnotu 1-4 podľa typu konfiguračnej správy. Pole *Identifier*-identifikátor slúži na spárovanie požiadaviek a odpovedí. Pole *Length* obsahuje dĺžku kompletnej správy. Pole *Options* môže obsahovať niekoľko parametrov špecifikovaných pomocou správ *LCP-Option*, definovaných protokolom *LCP*.

Tabuľka 1.6: Prehľad parametrov *LCP*

Kód	Názov parametru	Popis
1	Maximum-Receive-Unit	Maximálna dĺžka *PPP* správy (vrátane hlavičky rámca), implicitne 1500B.
3	Authentication-Protocol	Autentizačný protokol, bližšie rozobraný v 1.4.1.
4	Quality-Protocol	Protokol kontroly kvality spojenia.
5	Magic-Number	Číslo, ktorým sa má protistrana identifikovať v správach *LCP-Echo*, slúži na detekciu zacyklenia.
7	Protocol-Field-Compression	Zapnutie kompresie čísla protokolu v *PPP* rámci.
8	Address-And-Control-Field-Compression	Zapnutie kompresie (vynechania) polí *address* a *control* v *PPP* rámci.

Každá správa typu *LCP-Option*(obrázok 1.10) obsahuje v prvom bajte kód typu parametru. Ďalší bajt obsahuje dĺžku správy a je nasledovaný dátami špecifickými pre daný typ parametru. Jednotlivé parametre sú uvedené v tabuľke 1.6.

Samotné vyjednávanie vyhovujúcej konfigurácie prebieha tak, že každá zo strán zašle správu *LCP-Configure-Request* s parametrami, ktoré požaduje.

V prípade, že protistrana prijíma všetky parametre uvedené v požiadavke, odpovie správou *LCP-Configure-Ack*, ktorá musí obsahovať kópiu požadovaných parametrov a jej identifikátor sa bude zhodovať s identifikátorom požiadavky.

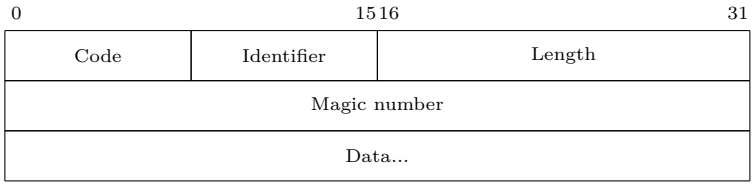
Pokiaľ požiadavka obsahovala parametre ktoré sú pre prijímateľa neznáme alebo sú nepodporované, odpovie správou *LCP-Configure-Reject*. Táto správa bude obsahovať nepodporované parametre z požiadavky v poradí v akom sa vyskytovali v požiadavke.

Ak sú všetky parametre v požiadavke podporované, ale je požadovaná hodnota, ktorú prijímateľ neakceptuje, odpovie správou *LCP-Configure-Nak*. Správa bude obsahovať tie parametre, ktorých nastavenie bolo neprijateľné v pôvodnom poradí, nastavené na nové hodnoty akceptované prijímateľom.

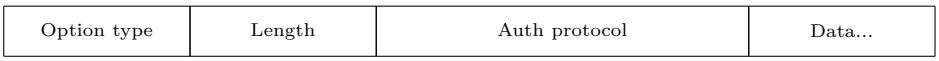
Linka sa považuje za otvorenú, keď si obe strany navzájom potvrdia požadované konfigurácie.

Správy LCP-Echo

Správy *LCP-Echo*(obrázok 1.11) slúžia na otestovanie spojenia. Každá zo strán použije hodnotu *Magic number*, ktorú vyžiadala protistrana počas konfigurácie. V prípade, že prijatá odpoveď *LCP-Echo-Reply* obsahuje rovnakú



Obr. 1.11: Štruktúra správ *LCP-Echo*



Obr. 1.12: Štruktúra správ *LCP-Auth-Method-Option*

hodnotu *Magic number* ako požiadavka *LCP-Echo-Request*, znamenatá to že linka je zacyklená.

Čas odpovede na správy *LCP-Echo* sa pri *PPTP* používa na odhad hodnoty time-out používanej v okienkovom protokole *GRE*(1.2.1).

Správy *LCP-Echo* môžu obsahovať ľubovoľné dáta v poli *data*. Využitie tohoto poľa a formát prenášaných dát je ponechaný na jednotlivých implementáciách.

Správy LCP-Terminate-Request a LCP-Terminate-Ack

Strana spojenia, ktorá sa chystá spojenie ukončiť, by mala pred samotným ukončením odoslať správu *LCP-Terminate-Request* a počkať na prijatie odpovede *LCP-Terminate-Ack*. V prípade, že *LCP-Terminate-Ack* nedorazí, mal by žiadateľ pokus o korektné ukončenie spojenia opakovať.

Prijatie *LCP-Terminate-Ack* bez vyžiadania ukončenia spojenia indikuje, že protistrana vyžaduje opätovnú konfiguráciu *PPP* spojenia.

V nasledujúcich sekciách podrobnejšie popíšeme konfiguráciu autentizácie protokolu *PPP* a jednotlivé autentizačné protokoly.

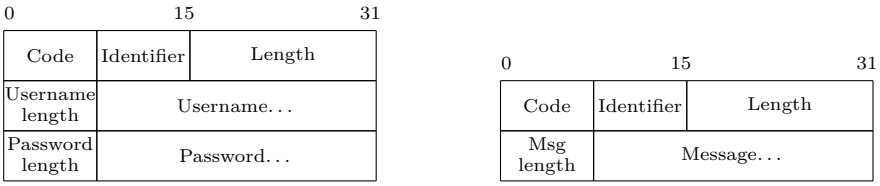
1.4 Autentizačné metódy PPTP/PPP

1.4.1 Autentizačné metódy protokolu LCP

Pri konfigurácii spojenia sa obidve strany dohodnú na autentizačnom protokole prostredníctvom vyžiadania parametru *LCP-Auth-Method-Option*, ktorého štruktúru je možné vidieť na obrázku 1.12.

Pole *Auth protocol* obsahuje kód protokolu vyžiadaného autentizačného protokolu. Konkrétna metóda pri protokole *Challenge-Response authentication protocol* je naviac určená jednobajtovou hodnotou *Data*.

V nasledujúcej sekcii podrobnejšie popíšeme jednotlivé autentizačné metódy.



(a) PAP-Authentication-Request (b) PAP-Authentication-Ack/Nak

Obr. 1.13: Štruktúra správ autentizačného protokolu *PAP*

Pasword authentication protocol

Password authentication protocol (PAP) je protokol definovaný v dokumente RFC1334 [12]. Tento protokol je najjednoduchšou z dostupných autentizačných metód pre protokol *PPP*.

Po odsúhlasení tejto autentizačnej metódy v konfiguračnej fáze *PPP* spojenia odošle klient správu *PAP-Authenticate-Request* (obr. 1.13a), obsahujúcu užívateľské meno a heslo užívateľa ako otvorený *ASCII* text.

Na požiadavku server reaguje odoslaním správy *Authenticate-Ack* v prípade úspechu a *Authenticate-Nak* v prípade že pokus o prihlásenie bol neúspešný. Obidve tieto správy majú rovnakú štruktúru (obr. 1.13b), odlišujú sa iba rozdielnou hodnotou poľa *Code* a môžu zahŕnať krátky textový reťazec.

Jedná sa o najmenej bezpečnú autentizačnú metódu a každý kto má prístup k prenášanej komunikácii je schopný získať a potenciálne zneužiť prihlasovacie údaje.

Existujú voľne dostupné nástroje(viď. 2.1.4), ktoré zbierajú prihlasovacie údaje rôznych služieb vrátane *PPTP*, zneužívajúc práve nezabezpečný protokol *PAP*.

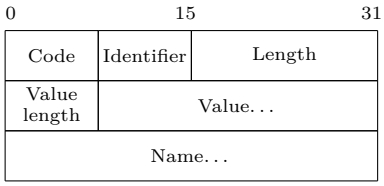
Autentizácia pomocou *PAP* zároveň vylučuje použitie šifrovania *MPPE*, keďže protokol nedefinuje spôsob odvodenia *RC4* kľúča z hesla.

Challenge-Handshake authentication protocol

Challenge-Handshake authentication protocol(*CHAP*) umožňuje autentizáciu nielen pri nadväzovaní spojenia, ale i kedykoľvek počas priebehu spojenia a poskytuje o niečo vyššiu bezpečnosť ako *PAP*.

Existuje niekoľko variant protokolu *CHAP*. Konkrétna varianta je určená jednobajtovým kódom v poli *Data* v správe *LCP-Auth-Method-Option* (obr.1.12).

Dokument *RFC1994* [24] definuje variantu *CHAP-MD5*. Varianta *CHAP-SHA1* je až na použitú hešovaciu funkciu identická s *CHAP-MD5*. Táto metóda vyžaduje aby pred samotnou autentzáciou klient i server poznali tzv. tajomstvo - reťazca znakov , ktorý je známy iba týmto dvom stranám. Autentizácia prebieha nasledovne:



Obr. 1.14: Štruktúra správy *CHAP-Challenge/Response*

1. Server odošle klientovi správu *CHAP-Challenge* (obr. 1.14), obsahujúcu náhodný reťazec - tzv. výzvu v poli *Value* a dĺžku tejto hodnoty v poli *Value length*.

2. Klient spočíta *MD5/SHA1* hash zreťazenia výzvy a tajomstva. Výsledok odošle späť serveru pomocou správy *CHAP-Response*(obr. 1.14), opäť v poli *Value*.

3. Server takisto spočíta *MD5* heš zreťazenia výzvy a tajomstva. Pokiaľ sa výsledná hodnota zhoduje s hodnotou, ktorú obdržal od klienta, klient úspešne preukázal znalosť tajomstva a autentizácia bola úspešná. Server na základe úspechu/neúspechu autentizácie odošle správu *CHAP-Authentication-Ack/Nak*. Štruktúra týchto správ je zhodná so správami *PAP-Authentication-Ack/Nak* (obr. 1.13b). Jediný rozdiel predstavujú hodnoty poľa *Code* určujúce typ správy.

Podľa *RFC1994* [24] hodnota *Name* v správach *CHAP-Challenge* a *CHAP-Response* určuje, ktoré tajomstvo má protistrana použiť. Návrh počíta s možnosťou, že pri jednom sedení môže byť použitá autentizácia pomocou niekoľkých tajomstiev (vyžiadaných serverom pri *CHAP-Challenge*). Typicky, implementácia *PPTP* odosiela v poli *Name* v správe *CHAP-Challenge* meno servera. Klient prostredníctvom tohoto poľa odošle v správe *CHAP-Response* svoje užívateľské meno, na základe ktorého server vyhľadá hodnotu tajomstva konkrétneho klienta ktoré, je teda jeho heslom.

Zdieľané tajomstvo (heslo) musí byť dostatočne dlhý a náhodný reťazec. V prípade, že by to tak nebolo, mohol by potenciálny útočník, ktorý zachytí správy *CHAP-Challenge* a *CHAP-Response* hrubou silou nájsť hodnotu tajomstva a správnosť hodnoty by bolo možné overiť spočítaním hashu výzvy a tajomstva.

Použitá hešovacia funkcia by mala byť dostatočne kryptograficky silná. Bohužiaľ, v súčastnosti túto požiadavku nespĺňa ani jedna z podporovaných hešovacích funkcií *MD5* [10] a *SHA1* [25].

Ďalšou podmienkou pre bezpečnú implementáciu *CHAP* je aby hodnoty výziev neboli predvítateľné a neopakovali sa v priebehu spojenia.

Bohužiaľ, aj v prípade že by bola každá z vymenovaných podmienok splnená (silné heslo, silný hešovací algoritmus, nepredvídateľné výzvy), autenti-

zácia pomocou *CHAP* sá da jednoducho prekonať v prípade že útočník má prístup ku komunikácii medzi autorizovaným klientom a *PPTP* serverom používajúcim metódu *CHAP* [9].

Klient je podľa *RFC1994* [24] povinný odpovedať na každú správu *CHAP-Challenge* i po počiatočnej úspešnej autentizácii. Mohlo by sa zdať že opakovaná autentizácia počas priebehu spojenia zamedzuje neautorizovanému užívateľovi aby autentizované sedenie 'ukradol'. Útočník by po krátkom čase obdržal výzvu protokolu *CHAP*, na ktorú by nebol schopný odpovedať, čo by viedlo k zlyhaniu autentizácie a ukončeniu spojenia.

Paradoxne práve tento mechanizmus opakovaných žiadostí o autentizáciu je fatálnou zraniteľnosťou protokolu *CHAP*. Pokiaľ má útočník prístup k existujúcemu spojeniu (je schopný odpočúvať komunikáciu a komunikovať s obidvoma stranami spojenia) medzi autorizovaným klientom a *PPTP* serverom a súčasne toto spojenie využíva metódu *CHAP*, stačí ak každú výzvu prijatú od serveru prepošle autorizovanému klientovi s podvrhnutou adresou serveru. Injektovať takýto datagram do existujúceho *GRE*/*PPP* kanálu nie je vôbec zložité, stačí zvoliť vhodne vysoké sekvenčné číslo *GRE* vrstvy a upraviť identifikátor požiadavky *CHAP-Request*. Autorizovaný klient je povinný na výzvu odpovedať a útočníkovi stačí jeho odpoveď preposlať serveru (s upraveným identifikátorom).

Rovnako ako *PAP*, použitie metódy *CHAP* vylučuje použitie šifrovania *MPPE* (nie je definovaná metóda odvodenia kľúča).

Microsoft Challenge-Handshake authentication protocol

Protokol Microsoft Challenge-Handshake authentication protocol (*MS-CHAP*) je rozšírením protokolu *CHAP* vyvinutým spoločnosťou Microsoft, jeho špecifikácia je dostupná v dokumente *RFC2433* [32].

Správa s výzvou *MSCHAP-Challenge* má rovnakú štruktúru ako pri protokole *CHAP* a výzva má vždy dĺžku 8 bajtov. Odpoveď na výzvu *MSCHAP-Response* je kompatibilná s *CHAP-Response*. Hodnota value má 49 bajtov a je rozdelená na 3 časti:

- 24 bajtov - Odpoveď na výzvu vytvorená pomocou hešovacieho algoritmu *LMHash*.
- 24 bajtov - Odpoveď na výzvu vytvorená pomocou hešovacieho algoritmu *MD4* (*NTHash*).
- 1 bajt určujúci, či sa má použiť odpoveď vytvorená pomocou *NTHash*.

Prvá z hodnôt je z hesla užívateľa odvodená nasledovným spôsobom:

1. Vypočíta sa hodnota *LMHash* z užívateľského hesla. Malé znaky anglickej abecedy sa v hesle zamenia za veľké. *LMHash* môže byť použitý iba

pokiaľ je heslo dlhé maximálne 14 znakov. V prípade, že je kratšie, doplní sa na dĺžku 14 bajtov hodnotami 0. Týchto 14 bajtov sa rozdelí na 2 časti po 7 bajtov. Každá z týchto častí tvorí jeden *DES* kľúč (56 bitov bez parity). Následne je algoritmom *DES* v móde *ECB*[10] bez zarovnania zašifrovaný reťazec *"KGS!@#$%"* pomocou obidvoch kľúčov. Výsledkom tejto operácie sú dva 8 bajtové reťazce ktorých zreťazenie tvorí 128 bitovú hodnotu LM-Hash užívateľského hesla.

2. Hodnota hešu sa zarovná na 168 bitov nulovými bitmi. Následne je rozdelená na 3 časti po 56 bitov. Každá z týchto častí sa použije ako samostatný *DES* kľúč a zašifruje v móde *ECB* 8 bajtovú výzvu prijatú v správe *MSCHAP-Challenge*. Zreťazenie výsledných hodnôt tvorí prvú 24 bitovú časť poľa *Value* v odpovedi *MSCHAP-Response*.

Druhá hodnota je získaná spočítaním 128 bitovej hodnoty z užívateľkého hesla pomocou hešovacej funkcie *MD4* a následným použitým tejto hodnoty na zašifrovanie výzvy podľa druhého bodu postupu uvedeného vyššie.

Protokol *MSCHAP* narozdiel od *CHAP* nevyžaduje aby systém oproti ktorému sa klient autentizuje mal jeho heslo uložené ako otvorený text. Stačí mu znalosť hodnoty príslušného hešu, z ktorého odvodí *DES* kľúče rovnakým spôsobom ako klient a pomocou dešifrovania troch 8 bajtových častí odpovede v prípade úspešnej autentizácie získa hodnotu výzvy ktorú zaslal klientovi.

Staré systémy od spoločnosti Microsoft ukladali heše užívateľských hesiel získané pomocou algoritmu LMHash. Od verzie Windows NT až do súčasnosti ukladajú tieto systémy používateľské heslá zahešované algoritmom MD4 (NT-Hash). Protokol *MS-CHAP* umožňuje používať oba hešovacie algoritmy pre zachovanie kompatibility so starými systémami.

Algoritmus *LMHash* je relatívne jednoduché prelomiť. Napriek názvu nejde o skutočný jednosmerný hešovací algoritmus podľa definície. Heslo môže mať maximálne 14 znakov, čo pri 95 tlačiteľných znakov ASCII znamená $95^{14} \approx 2^{92}$ kombinácií. Keďže algoritmus nerozlišuje veľké a malé znaky, a heslo je rozdelené na dve časti po 7 znakov z ktorých správnosť každej časti sa dá overiť samostatne, na získanie kompletného hesla je postačujúce overenie $69^7 \approx 2^{43}$ možností. Naviac, je možné na prvý pohľad identifikovať heslá kratšie ako 8 znakov, keďže v tomto prípade bude druhú polovicu hešu tvoriť hodnota 0xAAD3B435B51404EE [11]. Aj samotný dokument RFC2433 *MS-CHAP* varuje pred používaním algoritmu *LMHash*. V prípade že *LMHash* nie je používaný, je príslušných 24 bajtov poľa *Value* v správe *MSCHAP-Challenge* vyplnených hodnotami 0. Hešovací algoritmus *MD4* (*NTHash*) v súčasnosti rovnako nie je považovaný za bezpečný kryptografický hešovací algoritmus. Naviac, pri jeho použití nie je využívané tzv. solenie hesla, čo umožňuje použitie tabuliek predpočítaných hodnôt na rýchle prelomenie slabších hesiel.

[10] Electronic Code Book

Mohlo by sa zdať, že obavy o kvalitu použitých hešovacích funkcií nemajú priamu súvislosť s autentizačnou metódou *MSCHAP*, keďže počas tejto metódy nie je heš prenášaný po sieti, ale použitý na vytvorenie trojice DES kľúčov ktorými zašifruje výzvu. Bohužiaľ to tak nie je a hodnota hešu (trojice DES kľúčov) sa zo zachytených odpovedí na výzvu dá získať. Ako bolo spomínané vyššie, hodnota hešovacej funkcie je z pôvodných 128 bitov zarovnaná na 168 bitov pomocou nulových bitov. To znamená že tretí DES kľúč obsahuje posledných 16 bitov hodnoty hešovacej funkcie a zvysných 40 bitov kľúča tvoria nuly. Na získanie posledných dvoch bajtov hešu je teda postačujúce overiť 2^{16} možností. Znalosť týchto posledných dvoch bajtov sa dá využiť na efektívnejšie hľadanie pôvodného hesla [20].

Heš z hodnôt odpovede na výzvu je možné v súčastnosti získať so zaručenou úspešnosťou omnoho jednoduchšie. Moxie Marlinspike v roku 2013 na konferencii DEFCON predviedol *FPGA*[11] implementáciu na nájdenie 2 neznámych *DES* kľúčov zo zachyteného nadväzovania spojenia pomocou *MSCHAP-v2*. Efektívnu implementáciu na *FPGA* uľahčil i fakt, že obidva kľúče šifrujú rovnakú hodnotu. Je teda možné nájsť oba kľúče pomocou jedného prechodu priestoru kľúčov a porovnávaním s očakávanou hodnotou. Predvedená implementácia bola schopná overiť priestor všetkých kľúčov za 23 hodín [14].

V súčasnosti je prelomenie *DES*,*MSCHAP* a *MSCHAPv2* dostupné ako cloudová služba sprostredkovaná webom `http://crack.sh` v cenách od 30 do 300 dolárov podľa priority a typu úlohy. Znalosť hešu plne postačuje na autentizáciu pomocou *MS-CHAP*/*MS-CHAPv2*, Je dobré podotknúť, že v prípade ak užívateľ nepoužíva bezpečné heslo, je pravdepodobné že útočník ho bude schopný získať za pomoci dostupných nástrojov na prelamovanie *MD4* prostredníctvom predpočítaných tabuliek.

V prípade, že je po úspešnej autentizácii pomocou metódy *MS-CHAP* vyžiadané šifrovanie *MPPE*, je počiatočný kľúč odvodený jedine z hodnoty hešu *NTHash* užívateľského hesla. To znamená, že počiatočný kľúč *RC4* je rovnaký pri každom sedení, až po najbližšiu zmenu užívateľského hesla a zároveň je rovnaký v rámci jedného sedenia pre obidve strany spojenia.

Aplikovaním operácie *XOR* na šifrovanú komunikáciu v smere od klienta a ku klientovi, prípadne na komunikáciu z viacerých sedení, obdrží útočník *XOR* otvoreného textu, z ktorého je často možné i pri čiastočnej znalosti otvoreného textu odkryť podstatnú časť komunikácie.

Microsoft Challenge-Handshake authentication protocol v2

Hlavnou zmenou v protokole *MSCHAPv2* oproti *MSCHAP* je pridanie autentizácie serveru klientom, ktorá v pôvodnom protokole chýbala. Z protokolu bolo taktiež odstránené použitie *LMHash* a upravený spôsob odvozdovania počiatočného kľúča *MPPE*.

[11] Field Programmable Gate Array

Autentizácia prebieha podobne ako pri *MSCHAP* a začína odoslaním výzvy v klientovi. Správa s výzvou má rovnakú štruktúru ako pri *CHAP* a výzva (*authenticator_challenge*) je vždy dlhá 16 bajtov. Klient na výzvu odpovedá zaslaním správy *MSCHAPv2-Response*, ktorá je kompatibilná s *CHAP-Response* a *MSCHAP-Response*. Hodnota 49 bajtového poľa *value* je rozdelená nasledovne:

1. 16 bajtov - Náhodný reťazec 16 bajtov vygenerovaný klientom, predstavujúci výzvu pre overenie serveru (*peer_challenge*) a zároveň potrebný pre overenie klienstkej odpovede na výzvu.

2. 8 bajtov - Zarovnanie 8 nulovými bajtmi pre zachovanie kompatibility so štruktúrou odpovede *MS-CHAP*.

3. 24 bajtov - Odpoveď na výzvu.

4. 1 bajt - Obsahuje hodnotu 1 pre zachovanie kompatibility so štruktúrou odpovede *MS-CHAP*.

Odpoveď na výzvu je vypočítaná rovnako ako v prípade *MSCHAP*. Klient však pri *MSCHAPv2* nepoužije 16 bajtovú hodnotu výzvy priamo (tak ako pri *MSCHAP* použil prijatú 8 bajtovú hodnotu). Vygeneruje náhodný 16 bajtový reťazec (*peer_challenge*), ktorý zreťazí s prijatou 16 bajtovou výzvou a užívateľským menom. Spočíta heš *SHA-1* výsledného reťazca a prvých 8 bajtov tohoto hešu použije na vytvorenie 24 bitovej odpovede na výzvu rovnako ako v prípade *MSCHAP*.

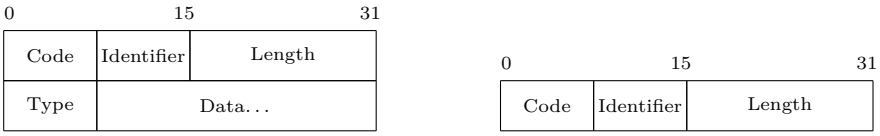
Server po prijatí správy s odpoveďou na výzvu rovnakým spôsobom odvodí 8 bajtovú hodnotu a použije ju na overenie odpovede identicky s *MS-CHAP*.

Pokiaľ bola autentizácia klienta úspešná odošle mu správu *Authenticate-Ack*, ktorá je kompatibilná so správou *Authenticate-Ack* protokolov *PAP/CHAP/MS-CHAP*(obr. 1.13b). V prípade *MS-CHAPv2* však bude prvých 40 bajtov pola *Message* obsahovať ASCII hexadecimálny zápis 20 bajtovej odpovede na výzvu servera, pomocou ktorých môže klient overiť jeho autenticitu. Táto 20 bajtová hodnota je spočítaná nasledovne:

1. Server, ktorý má užívateľské heslo uložené ako *NTHash*(*MD4*) heš, zahešuje túto hodnotu ešte raz pomocou *MD4*, čím získa hodnotu *PasswordHashHash*.

2. Server zreťazí *PasswordHashHash*, hodnotu odpovede na výzvu ktorú obdržal od klienta, *ASCII* reťazec s textom "Magic server to client constantä výsledok zahešuje pomocou *SHA1*.

3. Server zreťazí výsledok *SHA1* z predchádzajúceho kroku, 8 bajtovú hodnotu odvodenú z 16 bajtov výzvy (viď. vyššie) a *ASCII* reťazec s textom "Pad to make it more than one iteration". *SHA1* heš výsledného reťazca je výslednou 20 bajtovou hodnotou, ktorá je odoslaná klientovi.



(a) *EAP-Request/EAP-Response* (b) *EAP-Success/EAP-Failure*

Obr. 1.15: Štruktúra správ *EAP*

Klient po prijatí správy *Authenticate-Ack* spočíta predpokladanú hodnotu odpovede na výzvu rovnakým spôsobom ako server a porovná s prijatou odpoveďou. V prípade, že sa zhodujú je server pravdepodobne autentický.

Napriek tomu, o koľko je druhá verzia *MSCHAPv2* komplikovanejšia, autentizácia nie je o nič bezpečnejšia. Jedinou neznámou hodnotou v celej schéme je *MD4* heš hesla užívateľa, ktorý je možné získať prelomením *DES* rovnako ako pri metóde *MSCHAP* [14].

Počiatočné kľúče pre prúdovú šifru *RC4* pri použití *MPPE* sú pri *MSCHAPv2* odvodené z hešu užívateľského hesla za použitia rozdielnych konštánt v závislosti na smere komunikácie. Tým je zabezpečené, že kľúče pre komunikáciu v oboch smeroch sú rozdielne.

1.4.2 Extensible Authentization Protocol

Protokol *EAP* - rozšíriteľný autentizačný protokol zastrešuje skupinu autentizačných protokolov. Okrem autentizácie *PPP* spojení je používaný i na autentizáciu v rámci bezdrôtových sietí. Existujú 4 druhy správ protokolu *EAP*, ktorých štruktúru je možné vidieť na obrázku (obr. 1.15):

- *EAP-Request* - Požiadavka (obr. 1.15a), určená hodnotou 1 v poli *Code*.
- *EAP-Response* - Odpoveď (obr. 1.15a), určená hodnotou 2 v poli *Code*.
- *EAP-Success* - Úspech (obr. 1.15b), určená hodnotou 3 v poli *Code*.
- *EAP-Failure* - Zlyhanie (obr. 1.15b), určená hodnotou 4 v poli *Code*.

Pole *Identifier* obsahuje identifikátor požiadavky a slúži na spárovanie *EAP-Response*, *EAP-Success*, *EAP-Failure* s príslušnou požiadavkou. *Length* obsahuje dĺžku *EAP* správy vrátane dát. Pole *Type* určuje typ správy - hodnoty vyššie ako 3 určujú konkrétnu použitú autentizačnú metódu.

Bezprostredne po vyjednaní autentizačného protokolu *EAP* prostredníctvom *LCP*, odosiela server klientovi správu typu *Identity-Request* (*Type 1*), ktorá je požiadavkou na zaslanie identity (užívateľského mena) a zároveň môže v dátovej časti obsahovať meno serveru. Klient odpovie správou *Identity-Response* s užívateľským menom v poli *Data*.

Identitu užívateľa server môže použiť na získanie povolených autentizačných metód pre daného užívateľa. Následne užívateľovi zašle požiadavku *EAP-Request* preferovanej autentizačnej metódy. Užívateľ odpovie správou *EAP-Response* príslušného protokolu, pokiaľ ho podporuje. V prípade, že požadovaný typ autentizácie nepodporuje nastaví typ správy na hodnotu 3 (*EAP-Legacy-Nak*) a do poľa *data* vloží bajt obsahujúci číslo určujúce hodnotu *Type* autentizačnej metódy, ktorú si praje použiť. Dokument RFC3748 [1] síce deklaruje, že v prípade zaslania *EAP-Legacy-Nak* môže dátová časť obsahovať ľubovoľný počet bajtov, z ktorých každý určuje jednu z podporovaných metód. Preskúmaním zdrojových kódov implementácie *EAP* v *pppd*[12] sme však zistili, že server zvyšné bajty ignoruje. Okrem správ *Identity*(1) a *Legacy-Nak*(3), definuje dokument RFC3748 [1] správu typu *Notification*(2), ktorá umožňuje zaslanie informačnej správy pre užívateľa. Správy s hodnotou *Type* vyššou ako 3 náležia konkrétnemu typu použitej autentizačnej metódy.

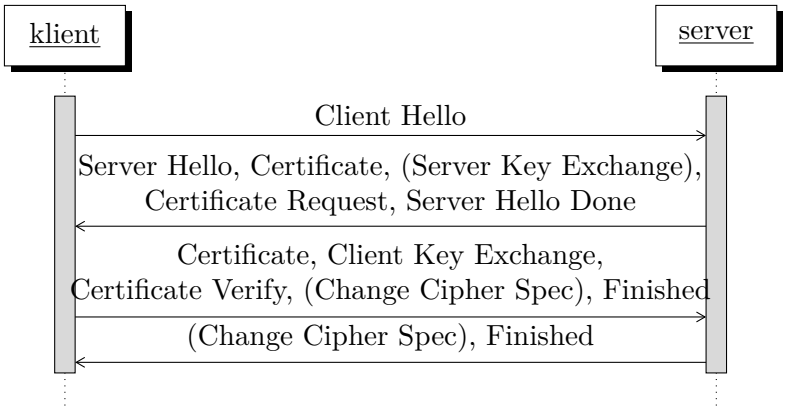
Spoločnosť *IANA* (Internent Assigned Numbers Authority) v čase písania tejto práce eviduje 49 autentizačných protokolov *EAP*[31]. Niektoré z týchto protokolov sú kópiou autentizačných metód *PPP*, ako napríklad *EAP-MD5*(4)[13], *EAP-MS-Authentication*(26)[14] a používajú dokonca identickú štruktúru správ, rozdiel je len v zapuzdrení do *EAP*. Implementácie *PPTP* s ktorými sme sa stretli používajú okrem týchto metód dve ďalšie, ktorým sú venované nasledujúce sekcie.

EAP-TLS

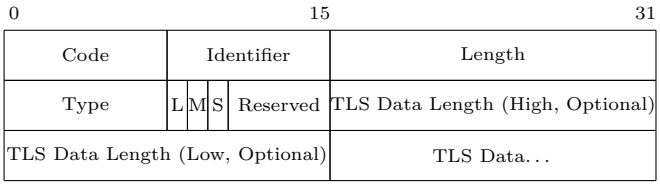
Protokol *TLS*[15] poskytuje okrem iného aj bezpečné overenie autenticity oboch komunikujúcich strán pri nadväzovaní spojenia pomocou digitálnych certifikátov.

Priebeh nadväzovania *TLS* spojenia s autentizáciou obidvoch strán pomocou digitálnych certifikátov je možné vidieť na obrázku 1.16. Nadväzovanie spojenia začína klient zaslaním správy *Client Hello* serveru. Táto správa obsahuje výčet podporovaných algoritmov, šifier, rozšírení a preferencie klienta. Na túto správu odpovedá server sériou správ počínajúc správou *Server Hello* ktorou klientovi oznamuje zvolené parametre (vybrané z možností *Client Hello*). Ďalej zasiela svoj certifikát (resp. reťaz certifikátov). V prípade pokiaľ to zvolená šifrová sada vyžaduje, je zaslaná správa Server Key Exchange s dodatočnou informáciou nutnou pre výmenu kľúčov. Správou *Certificate Request* server žiada od klienta autentizáciu prostredníctvom certifikátu. Koniec série správ zasielaných serverom je označený správou *Server Hello Done*. Klient odpovie sériou znova správ. Pomocou správy *Certificate* zašle serveru svoj certifikát (reťaz certifikátov) a prostredníctvom správy *Certificate verify* dokáže

[12] Implementácia *PPP* využívaná serverom *PoPToP*
[13] *EAP-MD5* je metóda identická s *CHAP-MD5*
[14] *EAP-MS-Authentication* je metóda identická s *MSCHAPv2*
[15] Transport Layer Security



Obr. 1.16: Sekvenčný diagram nadväzovania *TLS* spojenia



Obr. 1.17: Štruktúra správ *EAP-TLS-Request*/*EAP-TLS-Response*

vlastníctvo súkromného kľúča k zaslanému certifikátu. Správou *Client Key Exchange* odošle informácie potrebné k výmene (odvodeniu) kľúčov. Správa *Change Cipher Spec* je signálom, že klient prijal všetky potrebné informácie na odvodenie kľúčov a je pripravený začať šifrované spojenie. Ukončenie nadväzovania spojenia je signalizované zaslaním správy *Finished*. Server v prípade, že overenie klienta prebehlo v poriadku odpovie zaslaním správ *Change Cipher Spec* a *Finished* čím signalizuje, že je pripravený začať šifrované spojenia a ukončuje nadväzovanie spojenia [17].

Protokol *EAP-TLS* v RFC5216 [22] definuje zapuzdrenie *TLS* do *EAP* správ. Komunikácia prebieha prostredníctvom správ *EAP-TLS-Request* a *EAP-TLS-Response*, ktorých štruktúru je možné vidieť na obrázku 1.17.

Význam jednotlivých častí správy špecifických pre *EAP-TLS* je nasledovný:

1. *L*(Length present) - Bit indikujúci prítomnosť hodnoty *TLS Data Length*
2. *M*(More) - Bit indikujúci, že túto správu budú nasledovať ďalšie, prijímateľ musí s interpretáciou *TLS* správ počkať pokiaľ bude kompletná.
3. *S*(Start) - Bit indikujúci začiatok autentizácie
4. *Reserved* - Nepoužívané bity, ich hodnota musí byť nastavená na 0

5. *TLS Data Length* - Dĺžka kompletnej série *TLS správ* zasielaných v aktuálnej série *EAP-TLS* správ. Dĺžka je spravidla prítomná v prvej správe série a umožňuje prijímateľovi v predstihu alokovať potrebnú pamäť.

6. *TLS Data* - Časť dát série prenášaných *TLS* správ.

Narozdiel od *TLS*, pri *EAP* začína komunikáciu server. Autentizácia pomocou *EAP-TLS* začína odoslaním správy *EAP-TLS-Request* bez *TLS* dát s bitom *S* (Start) nastaveným na hodnotu 1.

Klient zašle správu *Server Hello* prostredníctvom jednej alebo niekoľkých *EAP-TLS-Response* správ. Na každú zo správ, ktorá má bit *M* nastavený na hodnotu 1, musí server odpovedať odoslaním správy *EAP-TLS-Request* bez TLS dát čím potvrdzuje prijatie *EAP-TLS-Response*.

Server odpovie ďalšou sériou, prostredníctvom ktorej prenesie *Server Hello*, *Certificate*,(*Server Key Exchange*), *Certificate Request*, *Server Hello Done*. Po overení autenticity serveru klient odošle sériu *EAP-TLS-Request* prenášajúcu TLS správy *Certificate*,*Client Key Exchange*, *Certificate Verify*, (*Change Cipher Spec*), *Finished*, ktorých prijatie klient potvrdzuje pomocou prázdnych správ *EAP-TLS-Response*.

V prípade úspešnej autentizácie server zašle sériu správ obsahujúcich *Change Cipher Spec*, *Finished*, ktoré klient potvrdí prázdnou správou *EAP-TLS-Reponse*. Úspešnú autentizáciu server potvrdí odoslaním *EAP-Success*. V prípade, že sa klient nepreukázal platným certifikátom, môže server zaslať správu *EAP-Failure* ešte pred dokončením nadviazania spojenia *TLS*.

Dokument RFC5126 definuje i spôsob odvodenia *MPPE RC4* počiatočných kľúčov z *TLS* hlavného kľúča (*master_key*) a hodnôt *client_random* a *server_random*. Autentizácia pomocou správne nakonfigurovaného protokolu *TLS* je v súčasnosti považovaná za bezpečnú.

Bohužiaľ nutnosť vydávania a práce s užívateľskými certifikátmi často používateľov i správcov *PPTP* serverov odrádza od využívania tejto metódy.

Protected EAP

Autentizačná metóda *PEAP*(Chránená EAP) sa snaží skombinovať pohodlie autentizácie užívateľským menom a heslom s bezpečím poskytovaným protokolom *TLS*.

Spôsobom indentickým s *EAP-TLS* sa vytvorí bezpečné *TLS* spojenie, avšak vo fáze nadväzovania sa pomocou *TLS* overí iba autenticita serveru. Klient sa autentizuje prostredníctvom niektorej z *EAP* metód pričom samotná *EAP* autentizácia prebieha vo vnútri šifrovaného *TLS* spojenia zapuzdrená do správ typu *EAP-TLV*. *PEAPv0* podporuje ako jedinú metódu pre "vnútornú"*EAP* autentizáciu, metódu *MsCHAPv2*.

Táto metóda je bezpečná za predpokladu správnej konfigurácie a používania. Bohužiaľ mnohé implementácie klientov pri PEAP nekontrolujú certifikát serveru (Android, ChromeOS. . .) a v prípade, že tak robia (Windows, iOS. . .),

užívatelia odsúhlasia pripojenie aj v prípade varovania pred nedôveryhodným serverom. Útočník tak môže pomocou útoku Man-in-the-middle prebrať úlohu serveru, nadviazať *TLS* spojenie pomocou nedôveryhodného certifikátu a následne prelomiť autentizáciu *MSCHAPv2* spôsobom popísaným v časti 1.4.1.

Kapitola 2

Analýza

2.1 Existujúce nástroje

Nástroj cielený výhradne na bezpečnostný audit *VPN* serverov využívajúcich *PPTP* sa nám bohužiaľ nepodarilo nájsť. Niektoré z komplexných skenerov zraniteľností alebo nástrojov na prieskum siete však umožňujú získanie omezených informácií aj o stave *PPTP* v sieti. Uvedieme preto stručný prehľad podpory *PPTP* protokolu v nájdených nástrojoch.

2.1.1 nmap

Nmap (Network mapper) [13] je nástroj s otvoreným zdrojovým kódom určený na prieskum sietí a bezpečnostný audit. Jeho funkcionalita je rozšíriteľná pomocou skriptovacieho jazyka *NSE*[16] Script.

V základnej sade skriptov, ktoré sú distribuované spolu s nástrojom nmap sa nachádza skript *pptp-version.nse*, ktorý umožňuje zistiť verziu *PPTP* serveru (*firmware_version*), názov serveru(*host_name*) a označenie (*vendor_string*).

```
  $ nmap -Pn -sSV -p1723 192.168.56.101
Starting Nmap 6.40 ( http://nmap.org ) at 2017-03-01 14:56 CET
Nmap scan report for 192.168.56.101
Host is up (0.031s latency).
PORT     STATE SERVICE VERSION
1723/tcp open  pptp    linux (Firmware: 1)
Service Info: Host: local

Service detection performed. Please report any incorrect
   results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.80 seconds
```

Príklad 2.1: Ukážka použitia nástroja *Nmap* na získanie základných informácií o *PPTP* servri

[16]Nmap Scripting Engine

2.1.2 THC-pptp-bruter

Program *THC-pptp-bruter* [27] je nástroj primárne určený na systematické hádanie hesla *PPTP VPN*, ktorá používa metódu *MSCHAP* resp. *MSCHAPv2*.

Je cielený na zneužitie implementácie *MSCHAP* a *MSCHAPv2* overovania v systémoch MS Windows, ktorá umožňuje pri využití viacerých paralelných spojení previesť slovníkový útok rýchlosťou približne 300 hesiel za minútu.

Okrem toho poskytne základné informácie o *PPTP* serveri rovnako ako spomínaný skript *pptp-version.nse* pre *Nmap*.

```
$cat wordlist | thc-pptp-bruter 192.168.0.5
Hostname 'WEBSERV', Vendor 'Microsoft Windows NT', Firmware:
    2195
5 passwords tested in 0h 00m 00s (5.00 5.00 c/s)
9 passwords tested in 0h 00m 02s (1.82 4.50 c/s)
...
```

Príklad 2.2: Ukážka použitia nástroja *THC-pptp-bruter*

2.1.3 Nessus

Pre komerčný skener zraniteľností *Nessus* vyvíjaný spoločnosťou *Tenable* existuje niekoľko rozširujúcich modulov týkajúcich sa *PPTP*. Modul *PPTP Detection* umožňuje detekciu *PPTP* serveru na skenovanej sieti. V čase písania tejto práce je okrem tohoto základného modulu dostupných ešte 17 modulov testujúcich prítomnosť známych zraniteľností konkrétnych implementácií [26]. Výstup skenu však neposkytuje žiadne informácie o konfigurácii *PPTP* serveru.

2.1.4 Nástroje na analýzu PPTP komunikácie

Existuje niekoľko nástrojov určených na analýzu sieťovej aktivity a získanie prihlasovacích údajov pri autentizácii *PPTP*, či už priamym zachytením hesla prenášaného v otvorenej podobe pri autentizácii protokolom *PAP*, alebo zachytením informácií prenášaných pri autentizácií metódou *MSCHAP* a následnou kryptoanalýzou s využitích známych zraniteľností [20].

Medzi tieto nástroje patrí [15]:

- *Anger* (`http://www.securiteam.com/tools/6F00X000AU.html`)
- *Dsniff* (`https://www.monkey.org/~dugsong/dsniff/`)
- *PPTP-sniff* (`https://packetstormsecurity.com/files/14454/pptp-sniff.tar.gz.html`)
- *chapcrack* (`https://github.com/moxie0/chapcrack`)

Program *chapcrack* si zaslúži špeciálnu pozornosť. Ide o program, ktorý Moxie Marlinspike prezentoval pri predstavení metódy na prelomenie *MS-CHAPv2*[14]. Program pracuje so záznamom sieťovej komunikácie vo formáte *pcap*. V poskytnutej komunikácii vyhľadáva autentizáciu *PPTP* spojení pomocou autentizačných metód *MSCHAP* a *MSCHAPv2* a vypíše nájdené hodnoty výziev a odpovedí. Z tých môže užívateľ pomocou služby *crack.sh*[17] získať *NTHash* užívateľského hesla, ktorý vie *chapcrack* použiť na dešifrovanie zachytenej *PPTP* komunikácie.

2.2 Možnosti implementácie *PPTP*

Implementácia nástroja na získanie informácií o *PPTP* serveri je podobná implementácii *PPTP* klienta. Bude potrebné implementovať *TCP* protokol *PPTP* riadiaceho spojenia, protokol *GRE* operujúci na linkovej vrstve, protokol *PPP* vrátane *LCP* a *EAP* metód.

Riešenie bude určite zahŕňať implementáciu parsovania množstva správ rôznych sieťových protokolov a implementáciu niekoľkých automatov riadicich príslušné protokoly.

Nutnosť práce so sieťovou vrstvou vyžaduje aby sme pracovali s tzv. *Raw* soketmi [18]. Tie nám narozdiel od *TCP*/*UDP* soketov umožnia špecifikovať formát dát už na sieťovej vrstve a implementovať *GRE*/*PPP*.

Pre implementáciu požadovaného nástroja sme preto zvažovali niekoľko možností, ktoré uvedieme v nasledujúcich sekciách.

2.2.1 Nízkoúrovňová implementácia priamo pomocou *Raw* soketov v C/C++

Jednou z možností ako nástroj implementovať je priame využitie *Raw* soketov.

Odhliadnuc od pracnosti, v prípade že by sme chceli nástroj spúšťať na OS *Windows*, čelili by sme problému. Spoločnosť *Microsoft* predstavila podporu pre *Raw* sokety vo *Windows 2000*. Pri vydaní záplaty *Service Pack 2* pre *Windows XP* bola však podpora *Raw* soketov výrazne obmedzená a neumožňuje odosielanie ľubovoľných dát [5].

Tento problém by bolo možné vyriešiť využitím projektu *WinPcap*. Ten sa skladá z ovládača, ktorý pristupuje k sieťovej karte priamo prostredníctvom rozhrania *NDIS*, nízkoúrovňovej knižnice komunikujúcej s ovládačom a upravenej knižnice *libpcap*. Hlavným účelom knižnice *libpcap* je zjednodušenie záznamu a analýzy sieťovej komunikácie. Knižnica však umožnuje i manipuláciu s paketmi na rovnakej úrovni ako *Raw* sokety [30].

[17] `http://crack.sh`

[18] Raw sokety sú nutné za predpokladu, že riešenie má bežať v užívateľskom priestore a nemá byť implementované na úrovni jadra operačného systému

Na parsovanie správ protokolov by bolo možné použiť časť kódu z *PPTP* serveru (*PopTop*) a klientov s otvoreným zdrojovým kódom, išlo by však pravdepodobne o veľmi pracnú variantu.

2.2.2 Implementácia modulu pre *Nmap* pomocou *NSE*

Na prvý pohľad lákavou možnosťou implementácie je využitie skriptovacieho jazyka *Nmap Scripting Engine*. *NSE* podporuje *Raw* sokety na nízkoúrovňovú manipuláciu s *IP* paketmi prostredníctvom knižnice *libcap* spomínanej vyššie. Atraktívna je aj integrácia vytvoreného nástroja s programom *Nmap* a rozšírenie jeho funkcionality. Skriptovací jazyk *NSE* je ale vhodnejší skôr pre jednoduché testy, nie pre implementáciu zložitejších sieťových protokolov. Toto tvrdenie dokladá i fakt, že medzi návrhmi *NSE* modulov je i modul na testovanie prístupových údajov k *PPTP* serveru pomocou hrubej sily, ktorého pokus o realizáciu skončil neúspechom so záverom, že protokol *PPTP* je komplexnejší ako sa na začiatku zdalo [16].

2.2.3 Implementácia v jazyku Python so *Scapy*

Scapy je nástroj a zároveň knižnica s otvoreným zdrojovým kódom na nízkoúrovňovú analýzu a manipuláciu so sieťovými paketmi. Oficiálne podporuje operačné systémy založené na *Linuxe*, *MacOS* i *Windows*. Interne na prístup k sieťovým rozhraniam využíva knižnicu *libpcap*(resp. *WinPcap*).

Medzi hlavné prednosti *Scapy* patrí jednoduchosť použitia. Akcie, ktoré by pri implementácii v jazyku *C* zabrali desiatky riadkov kódu je často možné implementovať pomocou jedného či dvoch riadkov kódu. Zrejme práve preto je *Scapy* obľúbeným nástrojom v oblasti sieťovej analýzy, či penetračného testovania, kde viac ako na rýchlosti spracovania sieťového toku záleží na flexibilite nástroja [21]. Z tohoto dôvododu sme sa rozhodli pre implementáciu požadovaného nástroja na bezpečnostný audit zvoliť práve knižnicu *Scapy*.

Napriek tomu, že *Scapy* obsahuje implementáciu množstva používaných sieťových protokolov, bohužiaľ *PPTP* medzi nimi nie je a podpora protokolov súvisiacich s *PPP* je minimálna. Rozhodli sme sa teda chýbajúce protokoly potrebné pre implementáciu nástroja do *Scapy* doplniť a pokúsiť sa o začlenenie týchto zmien do projektu.

V nasledujúcej časti sa pokúsime projekt *Scapy* stručne predstaviť.

2.3 Scapy

Projekt *Scapy* umožňuje jednoduchú manipuláciu so sieťovými paketmi. Umožňuje zostavovať, dekódovať, upravovať, posielať a prijímať pakety množstva protokolov. Dokáže spárovať požiadavky s odpoveďami, pracovať so záznamom sieťovej komunikácie vo formáte *pcap*, dovoľuje jednoducho pristupovať k jednotlivým sieťovým vrstvám komunikácie. Okrem toho poskytuje možnosť

pohodlnej implementácie nových protokolov, vrátane implementácie sieťových automatov [21]. *Scapy* pozostáva z niekoľkých modulov jazyka *Python 2.x* [19].

Okrem toho, že je možné použiť *Scapy* ako externú knižnicu vo vlastných skriptoch v jazyku *Python*, poskytuje aj interaktívne tzv. *REPL*[20] prostredie. Toto prostredie je vhodné na experimentovanie so sieťovými paketmi, možnosť spúšťania kódu v jazyku *Python* spolu s využitím funkcionality *Scapy* je mocným a efektívnym nástrojom pre sieťovú analýzu.

2.3.1 Inštalácia

Aktuálnu stabilnú verziu *Scapy* je možné nainštalovať do aktívneho prostredia *Python* pomocou nástroja *pip*, ktorý slúži na manažment *Python* balíčkov spustením príkazu:

```
$ pip install scapy
Collecting scapy
Installing collected packages: scapy
Successfully installed scapy-2.3.3
```

Príklad 2.3: Inštalácia *Scapy* pomocou *pip*

Po inštalácii je možné spustiť *Scapy* v interaktívnom móde jednoducho pomocou príkazu *scapy*,čím sa otvorí príkazový riadok *REPL* rozhrania:

```
$ scapy
Welcome to Scapy (2.3.3)
>>>
```

Príklad 2.4: Spustenie *Scapy*

Uvedieme zopár príkladov použitia na vytvorenie predstavy, ako sa so *Scapy* pracuje.

2.3.2 Príklad použitia *REPL* rozhrania

Prvý príklad ukazuje odoslanie správy *ICMP Echo-Request* (ping) na *IP* adresu domény *cvut.cz* a prijatie odpovede:

```
>>> reply=sr1(IP(dst='cvut.cz')/ICMP()/"test")
>>> reply.show()
###[ IP ]###
  version= 4L
  ihl= 5L
  tos= 0x0
  len= 32
  id= 57395
```

[19] Existuje aj neoficiálna verzia *Scapy* pre *Python 3.x* dostupná na `https://github.com/phaethon/scapy`

[20] Read-Eval-Print-Loop, tj. Prečítaj-Vyhodnoť-Vypíš-Opakuj

```
  flags=
  frag= 0L
  ttl= 55
  proto= icmp
  chksum= 0x4b0b
  src= 147.32.3.202
  dst= 192.168.1.12
  \options\
###[ ICMP ]###
     type= echo-reply
     code= 0
     chksum= 0x1826
     id= 0x0
     seq= 0x0
###[ Raw ]###
        load= 'test'
###[ Padding ]###
           load= '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
               \x00\x00\x00'
```

Príklad 2.5: Ping na *IP* adresu domény *cvut.cz*

Výraz `IP(dst='cvut.cz')` vytvorí instanciu *IP* vrstvy, kde ako cieľovú adresu vyplní *IP* adresu domény *cvut.cz*. Rovnako *ICMP()* vytvorí instanciu *ICMP* paketu.

Operátor / slúži na poskladanie jednotlivých vrstiev na seba a pri aplikácii na textový reťazec *'test'* z tohoto reťazca vytvorí instanciu *RawPacket* slúžiacu na zapuzdrenie surových dát.

Hodnoty, ktoré neboli explicitne špecifikované *scapy* doplní podľa dostupných informácií. Ako zdrojovú *IP* adresu použije adresu priradenú sieťovému adaptéru, ktorý bude použitý na odoslanie. Hodnotu polí označujúcich dĺžku dát a typ protokolu vyššej vrstvy doplní podľa vyšších vrstiev a podobne. V prípade, že dáta nejdú odvodiť, každé pole má preddefinovanú hodnotu, ktorá by mala zodpovedať častému prípadu použitia. Napríklad typ *ICMP* paketu má preddefinovanú hodnotu *Echo-Request*, vďaka čomu ju v príklade nebolo treba explicitne špecifikovať. Funkcia *sr1* slúži na odoslanie jedného paketu a čakanie na jednu odpoveď. Jej návratovou hodnotou je práve objekt reprezentujúci túto prijatú odpoveď, ktorého obsah je v príklade podrobne vypísaný pomocou zavolania funkcie *.show()*.

Ako ďalší príklad uvedieme sledovanie všetkých *DNS* dotazov a odpovedí prenášaných prostredníctvom sieťového adaptéru *eth0* pomocou funkcie *sniff*:

```
>>> sniff(iface='eth0', filter='udp',
          prn=lambda pkt: pkt[DNS].summary() if DNS in pkt
              else None)
DNS Qry "fit.cvut.cz."
DNS Ans "147.32.232.248"
DNS Qry "root.cz."
DNS Ans "91.213.160.118"
```

```
^C<Sniffed: TCP:0 UDP:12 ICMP:0 Other:0>
```

Príklad 2.6: Sledovanie *DNS* dotazov a odpovedí pomocou *Scapy*

Funkcia *sniff* umožňuje sledovanie a záznam komunikácie na sieťovom rozhraní, ktoré je špecifikované hodnotou argumentu *iface*, prípadne sledovanie a záznam komunikácie na všetkých dostupných sieťových rozhraniach nastavením tejto hodnoty na *'any'*.

Argumentom *filter* je možné špecifikovať paketový filter pomocou syntaxe *Berkley Packet Filter*[21], ktorá je používaná knižnicou *pcap*. Filtrovanie prebieha na úrovni tejto knižnice, čo umožňuje nechať *Scapy* parsovať len pakety, ktoré prejdú *pcap* filtrom a zrýchliť tak spracovanie.

Argument *prn* umožňuje predať funkcii *sniff* funkciu, ktorá bude aplikovaná na každý prijatý paket, ktorý prejde filtrom a výsledok tejto funkcie bude vypísaný. Na prítomnosť jednotlivých vrstiev v pakete sa je možné dotazovať pomocou operátora *in* a k samotným vrstvám môžeme pristupovať pomocou operátoru *[]*. Lambda výraz v príklade teda v prípade že zachytený paket obsahuje *DNS* vrstvu vráti výsledok volania *.summary()* na tejto vrstve.

Funkcia *.summary()* je definovaná pre všetky vrstvy a vracia stručný popis obsahu, v tomto prípade dotazovanú doménu alebo výslednú *IP* adresu.

2.3.3 Pridanie novej vrstvy

Implementácia novej vrstvy v *Scapy* často nevyžaduje viac ako špecifikáciu štruktúry jednotlivých správ. Scapy poskytuje širokú škálu rôznych typov polí, ktoré môže správa obsahovať. Zároveň je možné v prípade potreby implementovať vlastné typy polí a ľubovoľne meniť správanie vrstvy pri vytváraní paketu alebo parsovaní. Ako príklad uvedieme implementáciu vrstvy *GRE* používanej na vytvorenie tunelu pri *PPTP*, ktorej štruktúru je možné vidieť na obrázku 1.8:

```
class GRE_PPTP(Packet):

    name = "GRE␣PPTP"
    fields_desc = [BitField("chksum_present", 0, 1),
                   BitField("routing_present", 0, 1),
                   BitField("key_present", 1, 1),
                   BitField("seqnum_present", 0, 1),
                   BitField("strict_route_source", 0, 1),
                   BitField("recursion_control", 0, 3),
                   BitField("acknum_present", 0, 1),
                   BitField("flags", 0, 4),
                   BitField("version", 1, 3),
                   XShortEnumField("proto", 0x880b,
                       ETHER_TYPES),
                   ShortField("payload_len", None),
```

[21] http://biot.com/capstats/bpf.html

```
                    ShortField("call_id", None),
                    ConditionalField(XIntField("seqence_number"
                        , None), lambda pkt: pkt.seqnum_present
                         == 1),
                    ConditionalField(XIntField("ack_number",
                        None), lambda pkt: pkt.acknum_present
                        == 1)]

    def post_build(self, p, pay):
        p += pay
        if self.payload_len is None:
              pay_len = len(pay)
              p = p[:4] + chr((pay_len >> 8) & 0xff) + chr(
                  pay_len & 0xff) + p[6:]
        return p


bind_layers( GRE_PPTP, IP, proto=2048)
```

Príklad 2.7: Implementácia *GRE* v *Scapy*

Implementácia každej vrstvy dedí od triedy *Packet*. Hodnota členskej premennej *name* určuje názov vrstvy. To, čo je pre definíciu vrstvy najviac podstatné, je špecifikácia štruktúry pomocou zoznamu *field_desc*. Každé pole správy vrstvy je definované pomocou instancie príslušnej triedy.

Trieda *BitField* umožňuje špecifikovať polia s ľubovoľným počtom bitov uchovávajúce číselnú hodnotu. Prvým argumentom konštruktoru je meno poľa, druhým argumentom je prednastavená hodnota a posledný argument špecifikuje koľko bitov dané pole zaberá.

XShortEnumField špecifikuje 16 bitové pole, ku ktorému náleží výčet hodnôt s popismi. V prípade *GRE* určuje toto pole protokol vyššej vrstvy, keďže *GRE* môže teoreticky prenášať protokoly ako *Ethernet*, v príklade je použitý výčet hodnôt *ETHER_TYPE*, ktorý je už v *Scapy* definovaný.

Prítomnosť niektorých polí v správe môže byť podmienená inou hodnotou, prípadne zložitejšou podmienkou. Trieda *ConditionalField* umožňuje špecifikovať takéto polia pomocou funkcie indikujúcej ich prítomnosť. V príklade vyššie je použitý lamba výraz kontrolujúci nastavenie príslušného bitu.

Obsah paketu po zostavení môžeme upraviť implementovaním funkcie *post_build*. V príklade táto funkcia po zostavení paketu vyplní pole *payload_len* dĺžkou dát prenášaných na vyšších vrstvách. Táto dĺžka nie je známa pred zostavením paketu, preto obsah tohoto poľa nemohol byť špecifikovaný rovnako ako zvyšné.

2.3.4 Implementácia sieťového automatu

Okrem iného umožňuje *Scapy* aj pohodlnú implementáciu sieťových automatov. Sieťový automat je implementovaný triedou, ktorá dedí od triedy *Auto-*

maton. Stavy automatu a akcie vykonávané na základe rôznych podnetov sú potom implementované ako metódy tejto triedy. Ich úloha je určená niektorým z dekorátorov triedy *ATMT*. Pre názornosť uvedieme implementáciu jednoduchého automatu umožňujúceho zistenie otvoreného portu pomocou odpovede na *TCP SYN* paket:

```
class SYNScan(Automaton):

    def parse_args(self, target, port):
        Automaton.parse_args(self)
        self.target = target
        self.port = port

    def master_filter(self, pkt):
        return IP in pkt and pkt[IP].src == self.target\
                and TCP in  pkt and pkt[TCP].sport == self.
                    port

    @ATMT.state(initial=1)
    def BEGIN(self):
        pass

    @ATMT.state()
    def WAITING(self):
        pass

    @ATMT.state(final=1)
    def END(self, result):
        return result

    @ATMT.condition(BEGIN)
    def send_syn(self):
        tcp_syn_pkt = IP(dst=self.target) / TCP(flags='S')
        self.send(tcp_syn_pkt)
        raise self.WAITING()

    @ATMT.receive_condition(WAITING)
    def receive_reply(self, pkt):
        if pkt[TCP].flags == 0x12:  # SYN-ACK
            raise self.END('Open')
        elif pkt[TCP].flags == 0x4:  # RST
            raise self.END('Closed')

    @ATMT.timeout(WAITING, timeout=2)
    def timeout(self):
        raise self.END('Not reachable')
```

Príklad 2.8: *Scapy* automat na zistenie otvoreného *TCP* portu

Automat sa skladá z troch stavov: *BEGIN*, *WAITING* a *END*. Metódy definujúce stav sú označené použitím dekorátoru *ATMT.state()*. Počiatočný a koncový stav sú určené pomocou nenulovej hodnoty argumentov dekorátoru *initial*

a *final*.

Metóda *parse_args* slúži na spracovanie argumentov, ktoré boli automatu predané. Typ paketov, na ktoré bude automat reagovať je možné obmedziť implementáciou metódy *master_filter*, ktorá pre daný paket vráti hodnotu *True*, v prípade že má byť paket automatom spracovaný. V príklade vyššie sú touto funkciou ignorované všetky pakety ktoré neprichádzajú z *IP* adresy cieľa a nemajú *TCP* vrstvu s číslom zdrojového portu zhodným s testovaným číslom portu.

Dekorátor *ATMT.condition(STATE)* označuje akciu ktorá sa vykoná po prejdení do stavu *STATE*. V uvedenom príklade sa pri prejdení do počiatočného stavu spustí metóda *send_syn*, ktorá vytvorí *TCP SYN* paket, odošle ho a prejde do stavu *WAITING*. Prechod do stavu je realizovaný vyvolaním výnimky, ktorá je návratovou hodnotou (dekorátoru) metódy definujúcej daný stav.

Metódy označené dekorátorom *ATMT.receive_condition(STATE)* sa spúšťajú po prijatí sieťového paketu, keď sa automat nachádza v stave *STATE*. Metóda *receive_reply()* skontroluje či prijatý paket obsahuje príznaky *SYN-ACK*, čo značí že testovaný port bol otvorený, alebo *RST* čo značí opak.

Je samozrejme možné, že odpoveď na *SYN* nikdy nepríde. Táto situácia je ošetrená prostredníctvom metódy *timeout*, ktorá je označená dekorátorom *ATMT.timeout*, ktorý zabezpečí jej spustenie v prípade že automat sa v stave *WAITING* nachádza dlhšie ako 2 sekundy.

Automat je možné spustiť zavolaním metódy *run()*. Návratová hodnota metódy definujúcej koncový stav je zároveň návratovou hodnotou metódy *run()*:

```
>>> SynScan('192.168.1.1', 80).run()
Open
```

Príklad 2.9: *SYN* skenovanie otvoreného portu pomocou *Scapy* automatu

Okrem funkcionality v príklade umožňujú automaty definovať reakcie na dáta prijaté prostredníctvom ľubovoľného súborového deskriptoru. Automaty je možné zapuzdriť do triedy *SuperSocket*. Z pohľadu scapy sa tak automat stáva soketom a je možné ho použiť na implementáciu protokolov sieťových vrstiev. Bohužiaľ táto funkcionalita nie je zdokumentovaná, *Scapy* však obsahuje implementáciou jednoduchého *TCP* klienta (triedu *TCP_Client*), ktorou je možné sa inšpirovať.

2.4 Zhrnutie

V kapitole návrh sme rozobrali existujúce nástroje pre bezpečnostný audit *PPTP* serverov. Medzi dostupnými nástrojmi sú také, ktoré poskytujú základné informácie o testovanom serveri, testujú známe konkrétne zraniteľnosti

špecifických implementácií alebo slúžia na získanie hesla či prípadné prelomenie šifrovanej komunikácie. Žiaden z dostupných nástrojov neposkytuje informácie o povolených autentizačných metódach, pritom táto informácia pre úroveň bezpečnosti kľúčová.

Ďalej sme sa venovali možnostiam implementácie protokolu *PPTP*. Prebrali sme možnosti implementácie pomocou *Raw* soketov, knižníc *libpcap* a *WinPcap* a projektu *Scapy*. Vďaka veľkej flexibilite a obľúbenosti sa *Scapy* zdá vhodným kandidátom na implementáciu požadovaného nástroja. Posledná časť kapitoly teda stručne približuje projekt *Scapy*.

Kapitola 3

Návrh

V tejto kapitole popíšeme návrh nástroja na bezpečnostný audit *PPTP* serveru, ktorý sme sa rozhodli nazvať *PPTP Auditor*. Na úvod uvedieme požiadavky, ktoré na nástroj kladieme. Popíšeme nutné rozšírenie knižnice *Scapy*. Na koniec navrhneme spôsob akým bude nástroj fungovať a komunikovať so serverom pre získanie čo najviac informácií o jeho konfigurácii.

3.1 Požiadavky

PPTP Auditor bude konzolová aplikácia implementovaná v jazyku *Python 2.7*. Umožní užívateľovi otestovať konfiguráciu *PPTP serveru* a poskytne základné informácie, ktoré je možné získať z riadiaceho spojenia:

- verziu protokolu hlásenú *PPTP* serverom
- verziu firmwaru (zodpovedá v niektorých implementáciách verzii serveru)
- hodnotu *host_name*
- hodnotu *vendor_string*

Prostredníctvom riadiaceho spojenia server zasiela výrazne väčšie množstvo informácií. Mnohé z týchto hodnôt nie sú pre súčasné použitie *PPTP* relevantné a vzťahujú sa k vytáčanému pripojeniu. Zároveň však tieto hodnoty nie sú dané skutočnými parametrami hardvéru a moderné implementácie často volia konštanty ktoré neskôr umožňujú identifikovať o akú implementáciu sa jedná. Z tohoto dôvodu by mal navrhovaný nástroj zobierať nielen hodnoty uvedné vyššie, ale všetky informácie zasielané prostredníctvom riadiaceho spojenia, ktoré môžu prispieť k identifikácii serveru. Medzi tie patria všetky informácie zo správ *Start-Control-Connection-Reply* a *Outgoing-Call-Reply*, okrem chybových kódov a konštánt identifikujúcich typ správy.

Kľúčovým poznatkom na posúdenie bezpečnosti *PPTP* serveru je zoznam povolených autentizačných metód. *PPTP Auditor* by teda mal získať stav všetkých používaných autentizačných metód protokolu *PPP*:

- Password Authentication Protocol (PAP)
- Challenge Handshake Authentication Protokol + MD5 (CHAP+MD5)
- Challenge Hanshake Authentication Protokol + SHA1 (CHAP+SHA1)
- Microsoft Challenge Handshake Authentication Protocol v1 a v2 (MS-CHAP,MSCHAPv2)
- Extensible Authentication Protocol (EAP)

V správach z výzvov protokolov *CHAP* a *MSCHAP* môže server odosielať svoj názov. *PPTP Auditor* teda zozbiera aj tieto informácie pokiaľ je niektorý z týchto protokolov povolený. V prípade, že je povolený protokol *EAP*, mal by program zistiť, ktoré autentizačné metódy protokolu *EAP* sú povolené. Implementácie protokolu *EAP* môžu povoliť, alebo zakázať jednotlivé metódy na základe poskytnutej identity. Program teda musí dovoliť užívateľovi poskytnúť identitu *EAP*, ktorú použije pri testovaní. V prípade, že *PPTP Auditor* detekuje ako povolenú niektorú z metód *EAP-TLS* alebo *PEAP*, mal by umožniť získať certifikát serveru.

Všetky akcie ktoré program pri testovaní vykoná musia byť zaznamenané. Nástroj teda musí poskytovať možnosť logovania priebehu testovania, prípadne i záznamu komunikácie s testovaným serverom.

3.2 Rozšírenie Scapy o protokoly používané pri *PPTP*

Pre implementáciu požadovaného nástroja budeme musieť implementovať protokoly, ktoré v *Scapy* doteraz chýbajú. Keďže sa jedná o protokoly, ktoré je možné bežne pozorovať v sieťovej prevádzke, pokúsime sa implementáciu začleniť do projektu *Scapy*. Tým zjednodušíme prácu komukoľvek, kto bude chciet v budúcnosti skúmať *PPTP* alebo *PPP* komunikáciu, prípadne implementovať ďalšie nástroje.

Pre implementáciu nástroja *PPTP Auditor* je nutné do *Scapy* implementovať vrstvu *PPTP* riadiaceho spojenia. Za predpokladu, že testovacie scenáre budú zahŕňať iba moderné použitie *PPTP*, kde spojenie prebieha iba prostredníctvom *IP* siete budeme potrebovať pridať podporu pre všetky správy *PPTP* riadiaceho spojenia uvedené v tabuľke 1.1 okrem správ *Incoming-Call-Reply/ Request/ Connected* a *WAN-Disconnect-Notify*.

Scapy obsahuje podporu štandardnej vrstvy *GRE*. *PPTP* používa takzvanú *Enhanced-GRE* alebo *PPTP-GRE*(obr. 1.8), ktorej štruktúra je odlišná

od štandarnej *GRE*. Typ tejto vrstvy je možné odlíšiť pri parsovaní podľa hodnoty poľa *version*. Bude teda potrebné implementovať *PPTP-GRE* vrstvu ako špecializáciu už existujúcej *GRE*.

Scapy vie rozpoznať *PPP* vrstvu, chýba však implementácia *LCP* vrstvy ktorá bude nevyhnutná pre zistenie povolených autentizačných metód. Bude teda treba implementovať *LCP* vrátane podpory pre správy typu *LCP-Option*.

Vrstvy *EAP*, dokonca vrátane *EAP-TLS* sú v *Scapy* implementované. Bohužiaľ stávajúca implementácie je chybná. Správy typu *EAP-Identity* ignorujú identitu zaslanú serverom. Hoci server túto informáciu nie je povinný zaslať, môže nám v niektorých prípadoch prezradiť dodatočné informácie. Súčasná implementácia *EAP-TLS* úplne ignoruje fragmentáciu *TLS* správ a správne funguje iba v prípade, že celá séria *TLS* správ je zaslaná v jednej *EAP-TLS* správe. V prípade *PPP* je maximálna veľkosť správy väčšinou dosť malá a k fragmentácii pri *EAP-TLS* dochádza často. Bude teda potrebné stávajúcu implementáciu *EAP* opraviť.

3.3 Návrh nástroja PPTP-Auditor

Priebeh testovania bude rozdelený na niekoľko samostatných fáz:

1. V prvej fáze sa nadviaže riadiace spojenie a nástroj z odpovedí *Start-Control-Connection-Reply* a *Outgoing-Call-Reply* zistí základné informácie o *PPTP* serveri. Po inicializácii *GRE* spojenia zistí pomocou protokolu *LCP* povolené *PPP* autentizačné metódy. Riadiace spojenie bude implementované pomocou *Scapy* automatu, rovnako ako automat na zistenie povolených metód.

2. Pokiaľ bolo v predchádzajúcej fáze zistené, že niektorá z autentizačných metód *CHAP*,*MSCHAP* alebo *MSCHAPv2* je povolená, vyžiada si autentizáciu postupne pomocou každej z týchto metód. V zaslanej správe s výzvou v poli *optional_name* sa môže nachádzať názov servera, ktorý nástroj zaznamená.

3. Pokiaľ bolo v prvej fáze zistené, že je povolená autentizačná metóda *EAP*, pomocou ďalšieho *Scapy* automatu *PPTP Auditor* vyjedná *EAP* autentizáciu. Následne zaznamená autentizačnú metódu vyžiadanú serverom a požiada o autentizáciu metódou ktorej stav ešte nie je známy. V prípade že vyžiadaná metóda nie je serverom povolená, server ukončí *PPP* spojenie. Postup je teda potrebné opakovať, kým nie je známy stav všetkých testovaných metód. Pokiaľ je vyžiadanou metódou *EAP-TLS* alebo *PEAP*, nástroj bude v autentizácii pokračovať až do bodu kedy obdrží certifikát serveru.

4. *PPTP Auditor* zobrazí všetky zozbierané informácie a varovanie o nebezpečných autentizačných metódach ktoré boli povolené.

PPTP Auditor musí umožňovať záznam všetkých informácií potrebných pre neskoršiu rekonštrukciu priebehu testovania. Pre tieto účely umožní užívateľovi špecifikovať súbor do ktorého bude logovať odosielanie a prijatie všetkých správ, prípadné chyby alebo neočakávané udalosti. Zároveň využije funkcionality *Scapy* a umožní záznam všetkej komunikácie s testovaným serverom do súboru vo formáte *pcap*.

V nasledujúcich odstavcoch popíšeme bližsie fungovanie jednotlivých súčastí nástroja.

3.3.1 Riadiace spojenie *PPTP*

Pomocou riadiaceho spojenia bude *PPTP Auditor* komunikovať rovnako ako by komunikoval klient. Je teda potrebné nadviazať *TCP* spojenie na port 1723 (prípadne iný port špecifikovaný používateľom) a zaslať serveru správu *Start-Control-Connection-Request*. Server odpovie správou *Start-Control-Connection-Reply* z ktorej *PPTP Auditor* vyčíta základné informácie o serveri (verziu protokolu, tzv. firmware verziu, názov hostiteľa a *vendor_string*).

Následne *PPTP-Auditor* vyžiada vytvorenie *GRE* tunelu zaslaním správy *Outgoing-Call-Request* a počká na odpoveď *Outgoing-Call-Reply*. Tým získa tzv. číslo volania ktoré predá automatu zabezpečujúcemu *PPP* komunikáciu. Počas priebehu *PPP* spojenia prostredníctvom *GRE* musí riadiace spojenie reagovať na správy *Echo-Request* a *Call-Disconnect-Notify*. V prípade, ukončenia *PPP* spojenia zo strany *PPTP Auditor*-a by mala byť serveru zaslaná správa *Call-Clear-Request*.

Vo všetkých fázach testovania bude použitá rovnaká implementácia *PPTP* riadiaceho spojenia, odlišný bude len použitý automat implentujúci *PPP*.

3.3.2 Vrstva GRE

Knižnica *Scapy* umožňuje používať automat, ktorý je implementovaný prostredníctvom rozšírenia triedy *Automaton*, ako štandardný soket. Bude teda vhodné implementovať vrstvu *GRE* takýmto spôsobom. Získame tak soket, prostredníctvom ktorého môžeme priamo odosielať a prijímať *PPP* pakety. Vďaka tomu nebudeme musieť pri implementácii *PPP* automatov riešiť súčasne správanie *GRE* vrstvy ako zvyšovanie identifikátora pri odosielaní paketov, zahadzovanie paketov ktoré prídu mimo poradia a potvrdzovanie prijatia.

3.3.3 Zistenie povolených *PPP* autentizačných metód

Automat zisťujúci ktoré autentizačné metódy protokolu *PPP* sú povolené pomocou správ *LCP-Configure-Request/Ack/Nak/Reject* bude pracovať nasledovne:

- Pokiaľ od serveru prijme požiadavku na určitú autentizačnú metódu, označí ju ako povolenú a odpovie správou *LCP-Configure-Nak*. Pomocou tejto správy navrhne autentizačnú metódu ktorej stav nie je známy.

- Bude odosielať serveru požiadavky *LCP-Configure-Request* navrhujúce autentizačnú metódu ktorej stav ešte nie je známy. Pokiaľ neskôr zaznamená akceptovanie požiadavky serverom oznámené správou *LCP-Configure-Ack* označí metódu ako povolenú. Pokiaľ server odošle *LCP-Configure-Nak*, alebo na požiadavku opakovane nereaguje bude metóda považovaná za zakázanú.

Pre správne fungovanie musí automat podporovať aj spracovanie parametrov spojenia *magic_number* a odpovedať na správy *LCP-Echo-Request*. Výčet povolených metód je možné získať prostredníctvom jedného spojenia, keďže protokol *LCP* umožňuje meniť konfiguráciu kedykoľvek počas spojenia (nesmie ale dôjsť k zlyhaniu autentizácie).

3.3.4 Získanie mena servera z CHAP, MSCHAP a MSCHAPv2

Pokiaľ je stav všetkých *PPP* autentizačných metód z predchádzajúceho štádia známy, a bolo zistené že niektorá z metód *CHAP*, *MSCHAP* alebo *MSCHAPv2* je povolená, pre každú z týchto metód bude spojenie spustené ešte raz a bude vyžiadaná autentizácia pomocou príslušnej metódy. Po schválení autentizačnej metódy vyčkáme na prijatie správy s výzvou, ktorá v poli *optional_name* často obsahuje názov serveru.

3.3.5 Zistenie povolených EAP autentizačných metód

PPP automat implementujúci zisťovanie povolených *EAP* autentizačných metód bude implementovaný pomocou samostnatného *Scapy* automatu.

Pred samotnou autentizáciou si server vyžiada identitu používateľa. V tejto požiadavke zároveň môže zaslať svoju identitu (meno), protokol to však nevyžaduje. *PPTP Auditor* použije *EAP* identitu poskytnutú užívateľom. V prípade, že užívateľ identitu neposkytol, skúsi identitu *'user'*. Implementácia *PPTP* serveru *PoPToP EAP* identitu ignoruje a pokračuje v autentizácii v prípade akejkoľvek zaslanej hodnoty. Protokol však umožňuje zamietnutie autentizácie pomocou zaslania správy *EAP-Failure* v prípade, že bola poskytnutá nesprávna identita. Nástroj bude detekovať aj tento prípad.

Autentizáciu *EAP* začne klient zaslaním požiadavky preferovaného protokolu. *PPTP*. Na tieto požiadavky automat odpovie správou *Legacy-Nak*, pomocou ktorej vyžiada autentizáciu protokolom ktorého stav ešte nie je známy. V prípade, že je daný protokol povolený, odpovie server novou požiadavkou prostredníctvom príslušného protokolu. V opačnom prípade zašle správu *EAP-*

Failure a ukončí spojenie[22]. Je preto potrebné tento postup opakovať, kým nie je známy stav všetkých povolených metód.

Pokiaľ je príslušným protokolom *EAP-TLS* alebo *PEAP* musí automat *PPTP* pokračovať v nadväzovaní *TLS* spojenia prostredníctvom fragmentovaných *EAP* správ do bodu kedy od serveru prijme certifikát.

3.4 Varovanie používateľa na základe výsledkov testov

Po zozbieraní informácií o serveri ich *PPTP Auditor* vypíše na výstup spoločne s varovaniami ohľadom potenciálne nebezpečnej konfigurácií.

Je dobré dodať, že je možné pomocou útoku *Man-in-the-middle* vynútiť autentizáciu najslabšou metódou povolenou obidvoma stranami.

V prípade, že má server povolenú autentizačnú metódu *PAP*, útočník s prístupom ku komunikácii jednoducho získa užívateľské údaje. Zároveň všetka tunelovaná komunikácia nemôže byť chránená *MPPE* šifrovaním.

Šifrovanie je takisto zakázané v spojení s *CHAP+MD5* alebo *CHAP+SHA1*. Zároveň je možné tieto metódy triviálne obísť pomocou útoku *man-in-the-middle* pokiaľ má útočník prístup ku komunikácii oprávneného užívateľa.

Autentizáciu metódami *MSCHAP* a *MSCHAPv2* je možné prelomiť prostredníctvom služby *crack.sh* s absolútnou úspešnosťou. Zo získaného *MD4* hešu užívateľského hesla je možné v prípade, že heslo nie je dostatočne náhodné , zistiť pôvodné heslo pomocou tzv. *rainbow tables*. Okrem toho sú pri *MSCHAP* a zapnutom *MPPE* šifrovaní použité identické kľúče v obidvoch smeroch, vďaka čomu vie útočník získať *XOR* otvoreného textu komunikácie.

Autentizácia pomocou *EAP* umožňuje autentizáciu aj pomocou metód uvedených vyššie, zabalených do protokolu *EAP*. Okrem toho sa však pri *PPTP* spojeniach používajú metódy *EAP-TLS* a *PEAP*. Nie sú známe bezpečnostné slabiny metódy *EAP-TLS* za predpokladu bezpečnej konfigurácie a implementácie *TLS* protokolu. Pri oboch metódach je však nutné skontrolovať certifikát serveru oproti dôveryhodnej autorite. Ignorovanie nevalidného certifikátu pri *PEAP* môže spôsobiť že klient odošle napríklad údaje *MSCHAPv2* zabezpečným kanálom priamo útočníkovi, ktorý ich môže zneužiť rovnako ako pri niektorej z nechránených metód.

Bohužiaľ aj pri "bezpečnej"konfigurácii *PPTP*, bude protokol stále zraniteľný prostredníctvom útokov na slabé šifrovanie *MPPE* [28]. Injektovaním paketov do komunikácie je útočník schopný vynútiť resetovanie *RC4* kľúča pred vygenerovaním nového, čo spôsobí, že komunikácia bude opakovane šifrovaná rovnakým kľúčom. *PPTP Auditor* teda zobrazí varovanie pred použitím *PPTP* ako takého.

[22] Ukončenie spojenia nie je protokolom vyžadované, je to však implementované správanie linuxovej implementácie *PPTP* serveru *PopTop* a zdá sa, že i *PPTP* serveru v novších verziách Microsoft Windows

3.5 Zhrnutie

V kapitole bol popísaný návrh nástroja *PPTP Auditor* na hodnotenie bezpečnosti konfigurácie *PPTP* serverov pomocou knižnice *Scapy*. V úvode sme špecifikovali požiadavky na informácie ktoré nástroj o testovanom serveri zozbiera.

Pre implementáciu nástroja musíme *Scapy* rozšíriť o podporu chýbajúcich vrstiev protokolov súvisiacich s *PPTP*. Týmto rozšírením sa zaoberá druhá časť kapitoly. Aby bol nástroj schopný získať požadované informácie je potrebná podpora protokolu riadiaceho spojenia *PPTP*, protokolu *LCP*, autentizačných protokolov *CHAP* a *MSCHAP*/*MSCHAPv2*, protokolu *EAP* a autentizačných metód *EAP-TLS*, *EAP-PEAP* a *EAP-CHAP* a *EAP-MS-Authentication*.

Hlavnými komponentami nástroja bude niekoľko automatov implementovaných pomocou rozhrania *Automaton* knižnice *Scapy*. Ich funkcionalita je popísaná v nasledujúcich častiach kapitoly a postupne sa zaoberá automatom implementujúcim riadiace spojenia, automatom zisťujúcim ktoré *PPP* autentizačné metódy sú povolené, automatom, ktorý sa znaží zistiť názov serveru zasielaný v správach s výzvou pri autentizácii *CHAP*, *MSCHAP* a *MSCHAPv2* a automatom získavajúcim informácie o *EAP* metódach.

Na záver kapitoly sa zaoberáme varovaniami, ktoré program oznámi užívateľovi na základe povolených autentizačných metód.

Kapitola 4

Implementácia

V úvode tejto kapitoly bude popísaná implementácia chýbajúcich protokolov do knižnice *Scapy*. Väčšina týchto rozšírení bola v čase písania tejto práce začlenená do projektu *Scapy* a je bude súčasťou nasledujúceho vydania stabilnej verzie projektu.

Druhá časť tejto kapitoly rozoberá implementáciu konzolového nástroja *PPTP-Auditor*.

4.1 Implementácia chýbajúcich protokolov do Scapy

Ako bolo spomínané v predchádzajúcej kapitole, pre prácu s *PPTP* bolo potrebné rozšíriť knižnicu *Scapy* o niekoľko protokolov. Samozrejme bolo možné implementovať potrebnú funkcionalitu ako súčasť testovacieho nástroja.

Začlenením zmien do projektu *Scapy* však umožníme ostatným vývojárom jednoduchú implementáciu nástrojov používajúcich dané protokoly a užívateľom *Scapy* pohodlné experimentovanie s *PPTP* a *PPP* protokolmi.

Rozhodli sme sa neimplementovať podporu iba pre minimálne množstvo správ, ktoré bolo potrebné pre vývoj nástroja *PPTP-Auditor* ale implementovať v rámci možností podporu všetkých správ daných protokolov.

Podpora všetkých správ protokolu *PPTP*, ktoré sú uvedené v tabuľke 1.1, je už v súčastnosti súčasťou vývojovej vetvy projektu *Scapy*. *Scapy* automaticky parsuje správy riadiaceho spojenia pomocou implementovanej vrstvy pokiaľ *TCP* komunikácia prebieha z/do portu 1723.

Základné informácie o *PPTP* serveri zo správy *PPTP-Start-Control-Connection-Reply* je teraz možné získať s pomocou *Scapy* prostredníctvom pár riadkov kódu:

```
>>> s=socket.socket()
>>> s.connect(("192.168.56.101", 1723))
>>> ss=StreamSocket(s, PPTP)
>>> ss.sr1(PPTPStartControlConnectionRequest()).show()
```

```
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
###[ PPTP Start Control Connection Reply ]###
  len       = 156
  type      = Control Message
  magic_cookie= 0x1a2b3c4d
  ctrl_msg_type= Start-Control-Connection-Reply
  reserved_0= 0x0
  protocol_version= 256
  result_code= OK
  error_code= None
  framing_capabilities=
  bearer_capabilities=
  maximum_channels= 1
  firmware_revision= 1
  host_name = 'local'
  vendor_string= 'linux'
```

Príklad 4.1: Získanie základných informácii o *PPTP* serveri pomocou rozšíreného *Scapy*

Existujúca podpora pre správy *GRE* vrstvy bola rozšírená o podporu modifikovaného *GRE* ktoré je používané pri *PPTP*. Na základe verzie protokolu v hlavičke *Scapy* sa rozpoznaný *GRE* paket rozparsuje pomocou správnej triedy. Podpora modifikovanej *GRE* hlavičky je taktiež súčasťou vývojovej verzie *Scapy*.

Okrem toho sme implementovali zjednodušené riadenie *GRE* vrstvy prostredníctvom *Scapy* automatu *GREPPTPConnection* ktorý je súčasťou testovacieho nástroja.

Tento automat sa dá použiť prostredníctvom rozhrania *SuperSocket* na vytvorenie *Scapy* soketu umožňujúceho zasielanie a prijímanie dát dátovej vrstvy prostredníctvom *GRE* tunelu. V súčastnosti tento soket neimplementuje okienkový protokol tak ako je popísané v časti 1.2.1. Pre implementáciu experimentálnych nástrojov to však nie je potrebné. Pomocou triedy *GREPPTPConnection* je možné jednoducho získať soket prostredníctvom ktorého sa dá jednoducho komunikovať cez *GRE*/*PPP* tunel:

```
>>> s~= GREPPTPConnection.grelink(PPP, '192.168.56.101',
    call_id, peer_call_id)
>>> s.send(PPP()/SomePPPPayload)  # Odosli PPP data cez GRE
>>> pkt = s.recv()                # Prijmi PPP data cez GRE
```

Príklad 4.2: Vytvorenie a použitie soketu na komunikáciu prostredníctvom *GRE* tunelu a *PPP* protokolu.

Táto funkcionalita v čase písania tejto práce ešte nie je začlenená do projektu *Scapy*, po miernych úpravách sa však o začlenenie pokúsime. Plánované zmeny

by mali umožniť inicializovať *GRE/PPP* tunel prostredníctvom riadiaceho protokolu *PPTP* a následne umožniť užívateľovi *Scapy* manipulovať s *PPP* komunikáciou vo vytvorenom tuneli.

Medzi zmeny, ktoré už sú súčasťou *Scapy* patrí podpora všetkých správ protokolu *LCP* (prehľad správ je možné vidieť v tabuľke 1.5). Tieto správy sú automaticky rozpoznané v kontexte akejkoľvek *PPP* komunikácie na základe čísla protokolu v *PPP* hlavičke.

Podpora pre autentizačný protokol *PAP* umožňuje jednoducho vyhľadať prihlasovacie údaje v zázname akejkoľvek *PPP* komunikácie:

```
>>> pkts=rdpcap('PPTP_negotiation.cap')
>>> [(pkt.username, pkt.password) for pkt in pkts if
    PPP_PAP_Request in pkt]
[('joe','secret_password')]
```

Príklad 4.3: Vyhľadanie prihlasovacích údajov v zázname *PPP* komunikácie používajúcej *PAP*

Implementácia podpory pre správy *CHAP* bola nutná pre získanie informácií o mene serveru, ktoré je často zasielané v poli *optional_name* v správe s výzvou. Parsovanie jednotlivých častí hodnoty *value* protokolov *MSCHAP* a *MSCHAPv2* nebolo implementované. Implementácia *PAP* a *CHAP* v uvedenom rozsahu je v čase písania práce schvalovaná a bude čoskoro súčasťou *Scapy*.

Ako bolo spomínané v predchádzajúcej kapitole, správy *EAP* neboli pôvodne v *Scapy* implementované správne. Neumožňovali spracovanie fragmentovaných *EAP-TLS* správ a správne parsovanie hodnoty *Identity* v správe *Identity-Request*. Tieto chyby boli opravené a oprava začlenená do projektu *Scapy*.

Pre implementáciu nástroja bolo potrebné parsovať správy *PEAP* a správy s výzvou protokolu *EAP-MS-Authentication*(*MSCHAPv2*). Tieto sú však zatiaľ implementované iba ako súčasť nástroja *PPTP Auditor*, po doplnení podpory pre chýbajúce správy príslušných protokolov budú takisto pravdepodobne začlenené do projektu *Scapy*.

4.2 Implementácia nástroja PPTP-Auditor

Implementovaný nástroj *PPTP-Auditor* sa skladá z dvoch modulov jazyka *Python*:

- *scapy_pptp* - Implementácia komponent ktoré sú vhodnými kandidátmi na neskoršie začlenenie do knižnice *Scapy*. Obsahuje implementáciu automatu *GRE* protokolu, *PEAP* a čiastočnú implementáciu *EAP-MS-Authentication*.

- *pptp_auditor* - Spustiteľný modul, ktorý obsahuje implementáciu samotného nástroja *PPTP-Auditor*

Samotné testovanie pri spustení nástroja prebieha spôsobom ktorý je popísaný v kapitole Návrh. Nasledujúce sekcie stručne popíšu kľúčové časti modulu *pptp_auditor*.

4.2.1 Automat pre riadiace spojenie PPTP

Trieda *PPTPAutomaton* implementujúca riadiace spojenie *PPTP* umožňuje prostredníctvom parametra konštruktoru predať instancii automat ovládajúci *PPP* spojenie. Pri nadväzvaní spojenia zozbiera informácie zo správy *Start-Control-Connection-Reply*. Po inicializácii spojenia predá tomuto automatu príslušné čísla volaní a spustí v samostnatnom vlákne pomocou funkcie *.runbg()*.

Počas behu *PPP* automatu, odpovedá instancia triedy *PPTPAutomaton* na správy *Echo-Request*. Po ukončení behu *PPP* automatu odošle správu *Call-Disconnect-Notify* a vráti informácie zozbierané zo správ riadiaceho spojenia.

4.2.2 PPP automat na získanie zoznamu povolených autentizačných metód

Automat ktorý pomocou *LCP* protokolu zistí ktoré z autentizačných metód protokolu *PPP* sú povolené je implementovaný v triede *LCPEnumAuthMethodsAutomaton*.

Automat po štarte odošle správu *Configure-Request* s požiadavkou na jednu z autentizačných metód. Následne reaguje na správy protokolu *LCP* nasledovne:

- Pokiaľ prijme správu *Configure-Request* od serveru a táto správa obsahuje požiadavku na autentizačnú metódu, označí danú metódu ako povolenú. Následne na túto správu odpovie pomocou *Configure-Nak*, v ktorej navrhne autentizáciu pomocou metódy ktorej stav ešte nie je známy. To však nezaručuje, že server vo svojej nasledujúcej požiadavke použije navrhnutú metódu. *PPTP Auditor* teda zároveň odošle požiadavku *Configure-Request* s danou metódou.

- V prípade, že prijme správu *Configure-Ack* na niektorú z odoslaných požiadaviek, označí požadovanú metódu ako podporovanú. V tomto prípade odošle novú požiadavku *Configure-Request* požadujúcu zmenu autentizačnej metódy na takú, ktorej stav ešte nie je známy.

- Po prijatí správy *Configure-Nak* na niektorú z prechádzajúcich požiadaviek, je príslušná autentizačná metóda označená ako nepodporovaná. Zároveň je alternatívna metóda ktorú server navrhol označená ako povolená.

- Pokiaľ je v ktoromkoľvek kroku známy stav všetkých metód, automat odošle správu *Terminate-Request* a skončí.

4.2.3 *PPP* automat na získanie mena servera pomocou *CHAP*, *MSCHAP*, *MSCHAPv2*

Pre každú z metód *CHAP-MD5*,*CHAP-SHA1*, *MSCHAP* a*MSCHAPv2*, ktorá je povolená je nadviazané nové pripojenie a spustený automat *CHAPAutomaton*.

Ako argument konštruktoru tento automat príjma príslušnú *CHAP* metódu. Po spustení túto metódu vyžiada od serveru pomocou správy *Configure-Request*. Po dohodnutí sa na danej autentizačnej metóde (táto dohoda by mala byť možná, keďže príslušná metóda bola detekovaná ako povolená) automat čaká na prijatie správy *Challenge-Request*. Po zaznamenaní hodnoty *optional_name* (pokiaľ je prítomná) je *PPP* spojenie ukončené.

4.2.4 *PPP* automat na získanie informácií o povolených *EAP* metódach.

Pokiaľ bolo detekované, že metóda *EAP* je povolená, je spojenie nadviazané znova. Tentoraz je *PPP* spojenie riadené automatom implementovaných v triede *EAPAutomaton*.

Ako argument prijíma zoznam *EAP* metód ktoré má otestovať. Po spustení vyjedná autentizačnú metódu *EAP* a následne čaká na prijatie správy *Identity-Request*. Táto správa môže taktiež obsahovať meno serveru, ktoré bude zaznamenané. Ako odpoveď automat odošle *Identity-Response* s identitou ktorú poskytol užívateľ, prípadne preddefinovanou identitou *'user'*. V prípade, že je daná identita zamietnutá, môže server v reakcii na *Identity-Response* zaslať správu *EAP-Failure* a ukončiť spojenie. Pokiaľ server pokračuje v autentizácii pošle správu *EAP-Request* preferovanej autentizačnej metódy. Táto metóda je označená ako povolená a automat odpovie správou *EAP-Legacy-Nak* prostredníctvom ktorej požaduje autentizáciu alternatívnou metódou ktorej stav nie je známy. V prípade, že server navrhnutú metódu podporuje, odošle novú požiadavku *EAP-Request* prostredníctvom tejto metódy. V opačnom prípade odpovie správou *EAP-Failure*.

Pretože po každom pokuse o vyžiadanie nepodporovanej metódy server ukončuje spojenie, je potrebné tento postup opakovať, pokiaľ nie je známy stav všetkých požadovaných metódoch.

V prípade, že je prijatá požiadavka na autentizáciu pomocou *EAP-TLS* alebo *PEAP*, automat v autentizácii pokračuje v autentizácii až do momentu kedy prijme komplentnú odpoveď na *TLS* správu *ClientHello*, ktorá obsahuje certifikát (prípadne sériu certifikátov) serveru. Certifikáty sú uložené do súboru špecifikovaného užívateľom a spojenie je následne ukončené.

4.2.5 Problémy so zamietaním protokolu *GRE*

Počas implementácie a priebežného testovania sa na systéme Linux objavili problémy v prípade, že server odoslal *GRE/PPP* paket ako prvý. Jadro operačného systému na *GRE* paket odpovedalo odoslaním správy *ICMP-Protocol-Unreachable*. Po prijatí tejto správy server oprávnene ukončil spojenie. Problémom je, že *Raw* sokety používané *Scapy* nie sú viazané na konkrétny protokol, obsluha *IP* protokolu v jadre teda nevie o obsluhe *GRE* protokolu a odpovie spomínanou *ICMP* správou.

Súčasné *API* Scapy neumožňuje pohodlne tento problém vyriešiť. Je možné ho však obísť nastavením pravidiel lokálneho firewalu tak aby bolo zablokované odosielanie *ICMP-Protocol-Unreachable* správ na testovaný server. *PPTP Auditor* pred testovaním nastaví príslušné pravidlo prostredníctvom príkazu *iptables*. Po skončení testovania je toto pravidlo odstránené a túto funkcionalitu je možné vypnút prostredníctvom argumentu príkazovej riadky.

4.2.6 Obmedzená podpora triedy Automaton v OS Windows

Bohužiaľ, knižnica *Scapy* momentálne neumožňuje plnohodnotné použitie sieťových automatov implementovaných pomocou triedy *Automaton*. V súčasnosti je však možné pozorovať v komunite vyvíjajúcej projekt *Scapy* snahu o zlepšenie kompatibility s MS Windows. Je teda možné očakávať, že v blízkom čase bude aj tento problém opravený. Do tej doby je bohužiaľ možné nástroj *PPTP Auditor* spustiť len na systémoch ktoré používajú tzv. *BSD* sokety.

4.2.7 Inštalácia a použitie nástroja *PPTP Auditor*

Zaužívanou konvenciou pri projektoch implementovaných v jazyku *Python* je distribúcia zdrojových kódov balíčka spoločne so skriptom *setup.py*, ktorý slúži na inštaláciu balíčka vrátane všetkých závislostí. Pri dodržaní štandardnej štruktúry projektu je možné tento skript vytvoriť veľmi jednoducho pomocou modulu *setuptools*.

Aktuálnu verziu nástroja *PPTP Auditor* je možné získať z repozitára na serveri *GitHub* a nainštalovať do aktívneho *Python* prostredia nasledovne:

```
$ git clone https://github.com/jsebechlebsky/pptp_auditor.
    git
$ cd pptp_auditor
$ python setup.py install
```

Príklad 4.4: Inštalácia nástroja *PPTP Auditor* z verejného repozitára na serveri *GitHub*

Následne je možné nástroj spustiť pomocou príkazu *pptp_auditor*.

V čase písania tejto práce *PPTP-Auditor* závisí na vývojovej verzii projektu *Scapy*. Pre niektorých užívateľov môže byť inštalácia tejto verzie *Scapy* do prostredia *Python* používaného systémom nežiaduca. Táto situácia sa dá vyriešiť inštaláciou nástroja do tzv. virtuálneho *Python* prostredia vytvoreného pomocou nástroja *virtualenv*. Pre väčšie pohodlie je súčasťou distribúcie skript *pptp_auditor.sh* prostredníctvom ktorého je možné nástroj spúšťať. Pri prvom spustení sa vytvorí virtuálne prostredie do ktorého sa nainštalujú všetky závislosti.

V prípade, že si užívateľ praje používať nástroj prostredníctvom tohoto skriptu je na inštaláciu a spustenie postačujúce spustiť nasledovné príkazy:

```
$ git clone https://github.com/jsebechlebsky/pptp_auditor.
    git
$ cd pptp_auditor
$ ./pptp_auditor.sh
usage: PPTP Auditing tool [-h] [-p PORT] [-l LOGFILE]
                          [-i IDENTITY] [-c CERT_FILE]
                          [-e] [-r PCAP_FILE] [-di] [-d] [-a
                              ]
                          target
PPTP Auditing tool: error: too few arguments
```

Príklad 4.5: Inštalácia nástroja a spustenie pomocou skriptu *pptp_auditor.sh*

Ako je vidieť v príklade vyššie, program vypísal chybu. Pre spustenie je potrebný aspoň jeden argument - adresa (prípadne doména) cieľového *PPTP* serveru. Stručný výpis všetkých dostupných argumentov je možné získať spustením programu s argumentom *–help*:

```
$ git clone https://github.com/jsebechlebsky/pptp_auditor.
    git
$ cd pptp_auditor
$ ./pptp_auditor.sh --help
usage: PPTP Auditing tool [-h] [-p PORT] [-l LOGFILE] [-i
    IDENTITY]
                          [-c CERT_FILE] [-e] [-r PCAP_FILE]
                              [-di] [-d] [-a]
                          target

positional arguments:
  target                Adress of PPTP server

optional arguments:
  -h, --help            show this help message and exit
  -p PORT, --port PORT  PPTP port
  -l LOGFILE, --log LOGFILE
                        Filename for log
  -i IDENTITY, --identity IDENTITY
                        Identity to use with EAP
  -c CERT_FILE, --dump_cert_file CERT_FILE
```

```
                        File to dump server TLS certificate to
-e, --test_all_eap_methods
                        Test all EAP auth methods
-r PCAP_FILE, --record_pcap PCAP_FILE
                        Record communication with target to
                            pcap file
-di, --dont_drop_icmp
                        Dont drop ICMP protocol-unreachable
                            packets
-d, --log-debug         Log debug information
-a, --log-append        Append output to logfile instead of
    truncating it
```

Príklad 4.6: Výpis všetkých dostupných argumentov program *PPTP Auditor*

Význam jednotlivých argumentov je nasledovný:

- **-p**/–**port** - V prípade, že testovaný *PPTP* server očakáva riadiace spojenie na inom ako štandardnom porte 1723, umožňuje tento argument špecifikovať číslo portu.
- **-l**/–**logfile** - Umožňuje špecfikovať súbor do ktorého je zapisovaný log testu. Pokiaľ tento argument nie je použitý, log sa zapisuje do súboru *log.txt* v adresári z ktorého bol nástroj spustený.
- **-i**/–**identity** - Identita ktorá sa použije pri testovaní *EAP*, pokiaľ je *EAP* autentizácia povolená. V prípade, že užívateľ tento parameter nešpecifikuje použije sa hodnota *'user'*.
- **-c**/–**dump_cert_file** - Umožňuje špecifikovať súbor do ktorého sa zapíše certifikát serveru v prípade, že je povolená autentizačná metóda *PEAP* alebo *EAP-TLS*. Pokiaľ nie je tento argument špecifikovaný, certifikát sa neuloží.
- **-e**/–**test_all_eap_methods** - Vynúti test všetkých typov *EAP* metód, ktoré sú uvedené v na webe organizácie *IANA* [31]. V prípade, že tento argument nie je špecifikovaný, otestuje sa povolenie metód *EAP-MS-Authentication(MSCHAPv2)*, *EAP-TLS*, *PEAP* a *EAP-MD5* ktoré používajú známe implementácie *PPTP* pre systémy *Linux* a *Windows*.
- **-r**/–**record_pcap** - Zapne záznam sieťovej komunikácie s testovaným systémom do špecifikovaného súboru vo formáte *pcap*. V prípade, že tento argument nie je špecifikovaný, komunikácia nie je zaznamenávaná.
- **-di**/–**dont_drop_icmp** - Pokiaľ je *PPTP-Auditor* spustený na linuxe, počas testovania sa pokúsi zmeniť nastavenie firewallu pomocou *iptables* tak aby boli zahadzované správy *ICMP-Protocol-Unreachable*. Tento argument umožňuje toto správanie vypnúť.

- **-d**/–**log_debug** - Argument zapne logovanie informácií určených pre ladenie programu.
- **-a**/–**log_debug** - Pokiaľ tento argument nie je špecifikovaný a súbor do ktorého sa ide zapisovať log testu existuje, bude prepísaný. Pokiaľ si užívateľ praje v súbore zachovať pôvodný obsah, môže použiť tento argument a log bude zapisovaný na koniec súboru. Je tak možné zapisovať záznam z niekoľkých testov do jedného súboru.

Pre otestovanie serveru s preddefinovanými nastavenia stačí nástroj spustiť s *IP* adresou alebo doménou testovaného serveru. Interpreter Pythonu musí mať pre prácu s *Raw* soketmi, prípadne nastavenie firewallu pomocou iptables príslušné oprávnenie. Najjednoduchším riešením je spustiť *PPTP-Auditor* s právami superužívateľa.

```
$ sudo ./pptp_auditor.sh 192.168.56.102
PPTP Auditor
Probing enabled LCP authentication methods
Connecting to 192.168.56.101:1723 ... Connected
Connecting to 192.168.56.101:1723 ... Connected
Connecting to 192.168.56.101:1723 ... Connected
Probing enabled EAP authentication methods 0/4
Connecting to 192.168.56.101:1723 ... Connected
================================================
                  PPTP info
================================================
PPTP server domain:       pptpt.sk
PPTP server aliases:      Unknown
PPTP server IP:           192.168.56.101
PPTP server port:         1723
Protocol version:         1.0
Maximum_channels:         0
Firmware revision:        0
Framing capabilities:     Synchronous Framing supported
Bearer capabilities:      Analog access supported+Digital
   access supported
Host name:
Vendor string:            Microsoft
Connection speed:         1.58283174038 MiB/s
GRE window size:          16384
Packet processing delay:  16384
Physical channel id:      0
================================================
                PPP Authentication
================================================
PAP                       Disabled
CHAP+MD5                  Enabled,Name: WIN-9I4JRRCLPC4
CHAP+SHA1                 Disabled
MS-CHAP                   Unknown
MS-CHAP-v2                Enabled,Name: WIN-9I4JRRCLPC4
EAP                       Enabled
================================================
```

```
          EAP Authentication (Identity 'user')
================================================
EAP is disabled for identity 'user'
================================================
                    Warning
================================================
CHAP authentication is enabled. Connection is vulnerable to
    MitM attacks, no encryption is used.
MSCHAP/MSCHAPv2 authentication is enabled. NTHash of user
    password can be recovered by sniffingnetwork traffic.
You are using PPTP. The PPTP protocol is not considered to be
    really secure, even when configured properly.
```

Príklad 4.7: Príklad spustenia testu s preddefinovanými nastaveniami

4.3 Zhrnutie

V kapitole sme na úvod stručne popísali rozšírenie knižnice *Scapy* o podporu protokolov *PPTP*, *LCP*, *PAP*, *CHAP* a oprava podpory *EAP*. Tieto zmeny sú v súčasnosti oficiálnou súčasťou projektu *Scapy*. Zároveň sme spomenuli implementáciu *GRE* vrstvy pomocou triedy *Automaton*, ktorá umožňuje jednoducho používať *GRE* spojenie prostredníctvom soketového rozhrania.

Ďalej sme sa zaoberali implementáciou samotného nástroja. Kľúčovými komponentami nástroja je niekoľko sieťových automatov implementovaných pomocou triedy *Automaton*. Tie slúžia na obsluhu riadiaceho protokolu, získanie informácie o povolených autentizačných metódach protokolu *PPP*, získanie mena serveru pomocou protokolov *CHAP* a získanie informácií o povolených autentizačných metódach protokolu *EAP*.

Na záver kapitoly je popísaný spôsob distribúcie a inštalácie nástroja *PPTP Auditor* z verejného repozitára a možnosť pohodlného spúštania nástroja vo virtuálnom prostredí Python prostredníctvom skriptu *pptp_auditor.sh*.

Kapitola 5

Testovanie

Prvá časť tejto kapitoly sa venuje automatickým jednotkovým testom komponent projektu *Scapy*, pomocou testovacieho nástroja *UTScapy*.

V druhej časti rozoberieme testovanie samotného nástroja *PPTP Auditor* na rôznych konfiguráciách *PPTP* serveru *PopTop* a *PPTP* serveru ktorý je súčasťou operačného systému *Windows Server 2012*.

5.1 Testy pridaných protokolov v Scapy

Súčasťou projektu *Scapy* je program *UTScapy*, ktorý slúži na testovanie jednotlivých komponent [29]. Podmienkou pre prijatie nového kódu do projektu je, že zároveň s kódom musia byť poskytnuté *UTScapy* testy, ktoré nový kód pokrývajú.

UTScapy testy sú definované v textových súboroch. V rámci súboru sú testy rozdelené do testovacích sád. Každý test môže byť označený pomocou kľúčových slov, čo umožňuje spúšťať testy súvisiace so špecifickou funkcionalitou alebo protokolom.

Pre ukážku uvedieme časť testu parsovania a zostavovania správ *LCP-Echo*:

```
+ PPP LCP Tests
= Test LCP Echo Request / Reply
~ ppp lcp lcp_echo

lcp_echo_request_data = 'c021090700080000002a'.decode('hex')
lcp_echo_reply_data = str(PPP()/PPP_LCP_Echo(id=7,
    magic_number=77, data='defgh'))

lcp_echo_request_pkt = PPP(lcp_echo_request_data)
lcp_echo_reply_pkt = PPP(lcp_echo_reply_data)

assert lcp_echo_reply_pkt.answers(lcp_echo_request_pkt)
assert not lcp_echo_request_pkt.answers(lcp_echo_reply_pkt)
```

Príklad 5.1: Príklad spustenia testu s preddefinovanými nastaveniami

Prvý riadok začínajúci znakom '+' označuje začiatok novej testovacej sady s názvom *PPP LCP Tests*. Znak '=' na druhom riadku určuje názov testu - *Test LCP Echo Request / Reply*. Tretí riadok, začínajúci '~' vymenúva kľúčové slová ktoré patria k danému testu. Nasledujúce riadky obsahujú kód testu v jazyku Python. Ten serializuje a parsuje správy *LCP-Echo-Request*, *LCP-Echo-Reply* a testuje funkčnosť metódy *answers*, ktorá slúži na spárovanie požiadaviek s odpoveďami.

Podobným spôsobom boli implementované testy pre všetky zmeny ktoré boli začlenené do projektu *Scapy* a je možné ich nájsť v súbore *tests/pptp.uts*. Všetky testy v projekte *Scapy* sú automaticky spúšťané pri akejkoľvek zmene v kóde prostredníctvom služby *Travis CI*[23], čím sa zamedzuje nechcenému vzniku nových chýb.

5.2 Testovanie nástroja

Nástroj sme otestovali oproti poslednej stabilnej verzii *PPTP* serveru *PoPToP 2.4.7* bežiacom v systéme *Debian 8 - Jessie* a *PPTP* serveru, ktorý je súčasťou operačného systému *Windows Server 2012*. Oba systémy bežali vo virtuálnom stroji vytvorenom pomocou nástroja *VirtualBox*. Samotný nástroj bol spúšťaný v prostredí *Python 2.7.6* pod linuxovou distribúciou *Xubuntu*.

Server *PoPToP* bol naviac upravený pomocou záplaty umožňujúcej použitie *EAP-TLS*. Táto funkcionalita nie je súčasťou oficiálneho repozitára projektu *PoPTop*, ale je momentálne spravovaná separátne Janom Keijserom [7].

Počas testovania sme povoľovali a zakazovali jednotlivé autentizačné metódy. Vo všetkých prípadoch reagoval nástroj správne a príslušnú metódu označil ako povolenú alebo zakázanú podľa skutočného nastavenia.

Certifikát *EAP-TLS* sa zhodoval s certifikátom ktorý sme pre účely testovania vygenerovali a použili na linuxovom serveri.

Obsah logov a záznam komunikácie sme skontrolovali pomocou nástroja *Wireshark*.

Odskúšali sme rôzne kombinácie nastavení argumentov príkazovej riadky. V prípade *Windows Server 2012* dochádzalo k zamietaniu *EAP* autentizácie v prípade použitia nesprávnej identity (viď. ukážka spustenia nástroja v predchádzajúcej kapitole).

Výstupy testovania pomocou *PPTP-Auditor*-a sú pomerne dlhé, uvedieme teda len pár príkladov spolu s výstupmi. Nižšie je možné vidieť výsledok testu *PPTP* serveru vo *Windows Server 2012*. Argumenty príkazového riadku určujú, že *PPTP-Auditor* má použiť *EAP* identitu *test*, uložiť *TLS* certifikát

[23] `https://travis-ci.org/secdev/scapy`

serveru do súboru *certificate.pem*, zaznamenať komunikáciu so serverom do súboru *capture.pcap* a zapísať log testu do súboru *log.txt*.

```
$sudo ./pptp_auditor.sh  192.168.56.101 --identity test --
    dump_cert_file certificate.pem --record_pcap capture.pcap
    --log log.txt
PPTP Auditor
Probing enabled LCP authentication methods
Connecting to 192.168.56.101:1723 ... Connected
Connecting to 192.168.56.101:1723 ... Connected
Connecting to 192.168.56.101:1723 ... Connected
Probing enabled EAP authentication methods 0/4
Connecting to 192.168.56.101:1723 ... Connected
Probing enabled EAP authentication methods 2/4
Connecting to 192.168.56.101:1723 ... Connected
Dumping TLS Certificate chain to certificate.pem
Probing enabled EAP authentication methods 3/4
Connecting to 192.168.56.101:1723 ... Connected
================================================
                    PPTP info
================================================
PPTP server domain:       pptpt.sk
PPTP server aliases:      Unknown
PPTP server IP:           192.168.56.101
PPTP server port:         1723
Protocol version:         1.0
Maximum_channels:         0
Firmware revision:        0
Framing capabilities:     Synchronous Framing supported
Bearer capabilities:      Analog access supported+Digital
    access supported
Host name:
Vendor string:            Microsoft
Connection speed:         1.58283174038 MiB/s
GRE window size:          16384
Packet processing delay:  16384
Physical channel id:      0
================================================
                PPP Authentication
================================================
PAP                       Disabled
CHAP+MD5                  Enabled,Name: WIN-9I4JRRCLPC4
CHAP+SHA1                 Disabled
MS-CHAP                   Disabled
MS-CHAP-v2                Enabled,Name: WIN-9I4JRRCLPC4
EAP                       Enabled
================================================
          EAP Authentication (Identity 'test')
================================================
EAP-TLS                   Disabled
PEAP                      Enabled
MS-EAP-Authentication     Enabled,Name: WIN-9I4JRRCLPC4
MD5-Challenge             Disabled
================================================
```

```
                    PEAP Certificate
================================================
Serial:
    146389520567653123238684996969702610820
Issuer
CN:                         WIN-9I4JRRCLPC4-CA
Subject
CN:                         WIN-9I4JRRCLPC4-CA
Validity:                   Nov 08 13:11:03 2016 GMT to Nov 08
    13:21:03 2021 GMT
================================================
                    Warning
================================================
CHAP Authentication is enabled. Connection is vulnerable to
    MitM attacks, no encryption is used.
MSCHAP/MSCHAPv2 Authentication is enabled. NTHash of user
    password can be recovered by sniffingnetwork traffic.
PEAP Authentication is enabled. Make sure all clients are
    validating server certificate.
MS-EAP (MSCHAPv2) Authentication is enabld. NTHash of user
    password can be recovered by sniffing network traffic
You are using PPTP. The PPTP protocol is not considered to be
    really secure, even when configured properly.
```

Príklad 5.2: Test *PPTP* serveru vo *Windows Server 2012* pomocou nástroja *PPTP-Auditor*

Vo výstupe je možné vidieť, že server má povolenú autentizáciu pomocou metód *CHAP-MD5*, *MSCHAPv2*, *EAP-MS-Authentication (MSCHAPv2)* a *PEAP*. Medzi zaujímavé informácie ktoré sa dajú z výstupu vyčítať patrí názov servera, ktorým sa server identifikoval pri *CHAP*/*MSCHAP* autentizácii.

Podobný spustený rovnakým príkazom oproti serveru *PoPToP* vypíše nasledovné informácie:

```
$sudo ./pptp_auditor.sh  192.168.56.102 --identity notauser
    --dump_cert_file certificate.pem --record_pcap capture.
    pcap --log log.txt
PPTP Auditor
Probing enabled LCP authentication methods
Connecting to 192.168.56.102:1723 ... Connected
Connecting to 192.168.56.102:1723 ... Connected
Probing enabled EAP authentication methods 0/4
Connecting to 192.168.56.102:1723 ... Connected
Dumping TLS Certificate chain to certificate.pem
Probing enabled EAP authentication methods 2/4
Connecting to 192.168.56.102:1723 ... Connected
================================================
                    PPTP info
================================================
PPTP server domain:       Unknown
PPTP server aliases:      Unknown
PPTP server IP:           192.168.56.102
```

```
PPTP server port:          1723
Protocol version:          1.0
Maximum_channels:          1
Firmware revision:         1
Framing capabilities:      None
Bearer capabilities:       None
Host name:                 local
Vendor string:             linux
Connection speed:          256.0 MiB/s
GRE window size:           16
Packet processing delay:   16
Physical channel id:       0
================================================
                PPP Authentication
================================================
PAP                        Enabled
CHAP+MD5                   Disabled
CHAP+SHA1                  Disabled
MS-CHAP                    Disabled
MS-CHAP-v2                 Enabled
EAP                        Enabled
================================================
          EAP Authentication (Identity 'notauser')
================================================
MD5-Challenge              Enabled,Name: pptpd
EAP-TLS                    Enabled
PEAP                       Disabled
MS-EAP-Authentication      Disabled
================================================
                    EAP-TLS Certificate
================================================
Serial:                    15216008527498866602
Issuer
C:                         CZ
ST:                        Czech Republic
O:                         CVUT
OU:                        FIT
CN:                        PPTP Test CA
emailAddress:              sebecjan@fit.cvut.cz
Subject
C:                         CZ
ST:                        Czech Republic
O:                         CVUT
OU:                        FIT
CN:                        PPTP Test CA
emailAddress:              sebecjan@fit.cvut.cz
Validity:                  Nov 09 12:25:20 2016 GMT to Nov 04
    12:25:20 2036 GMT
================================================
                    Warning
================================================
PAP Authentication is enabled. User credentials are sent in
   plaintext, no encryption is used.
MSCHAP/MSCHAPv2 Authentication is enabled. NTHash of user
```

```
    password can be recovered by sniffing network traffic.
EAP-MD5 Authentication is enabled. Connection is vulnerable
    to MitM attacks, no encryption si used.
You are using PPTP. The PPTP protocol is not considered to
    be really secure, even when configured properly.
```

Príklad 5.3: Test *PPTP* serveru *PoPTop* pomocou nástroja *PPTP-Auditor*

Na výsledkoch testu je možné vidieť že testovaný server mal povolené autentizačné metódy *PAP*, *MSCHAPv2*, *EAP-MD5* a *EAP-TLS*. Narozdiel od *Windows Server 2012* server neprezdradil svoj názov. Identita *EAP* bola v tomto prípade nastavená na meno neexistujúceho používateľa 'notauser', no v prípade *PoPToP* autentizácia nebola z tohoto dôvodu predčasne ukončená a otestovať povolené *EAP* autentizačné metódy sa podarilo.

5.3 Zhrnutie

V úvode kapitoly sme stručne popísali testy komponent, ktoré boli pridané do projektu *Scapy*. Tieto testy boli implementované s využitím nástroja *UTScapy*, ktorý slúži na jednotkové testovanie komponent projektu *Scapy*.

V druhej časti kapitoly sme stručne popísali testovanie nástroja oproti používaným *PPTP* serveru *PoPToP* a *PPTP* serveru ktorý je súčasťou operačného systému *Windows Server 2012*. Pri všetkých testoch nástroj správne detekoval povolené autentizačné metódy.

Záver

Cieľom tejto práce bolo zoznámenie sa s protokolom *PPTP* a návrh nástroja na bezpečnostný audit *PPTP* serveru. Tento nástroj mal byť navrhnutý tak, aby získal o serveri čo najväčšie množstvo informácií a zároveň umožňoval neskoršiu analýzu priebehu testovania.

Práca začína popisom sieťových protokolov používaných pri *VPN* pripojeniach pomocou *PPTP*. Okrem protokolu riadiaceho spojenia sme sa venovali protokolu *PPP* a jeho zapuzdreniu v *GRE* vrstve, ktorá umožňuje tunelovanie *PPP* cez *IP* siete. Zároveň sme rozobrali jednotlivé autentizačné metódy protokolu *PPP* a *EAP* vrátane ich bezpečnostných problémov.

Následne sme sa venovali dostupným nástrojom na bezpečnostnú analýzu *PPTP* serverov. Žiaden z dostupných nástrojov neposkytuje informácie o povolených autentizačných metódach. Po rozobratí možností implementácie nástroja na analýzu konfigurácie *PPTP* serveru sme došli k záveru, že bude vhodné na implementáciu použiť knižnicu *Scapy*. Stručnému predstaveniu tejto knižnice je venovaný záver kapitoly.

Na začiatku kapitoly *Návrh* sme špecifikovali požiadavky na nástroj pre hodnotenie bezpečnosti *PPTP*. Pokračovali sme popisom zmien a rozšírení, ktoré bolo potrebné implementovať do knižnice *Scapy* pre prácu s protokolmi používanými v *PPTP*. Následne sme popísali návrh kľúčových častí testovacieho nástroja ktorých účelom je získanie informácií o povolených autentizačných metódach a riadiaceho spojenia. Na konci kapitoly sme uviedli bezpečnostné varovania ktorými nástroj na základe výsledku testu informuje užívateľa.

V úvode nasledujúcej kapitoly sme sa venovali implementácii zmien do knižnice *Scapy*. Tieto zmeny boli oficiálne začlenené do projektu *Scapy* a umožňujú komukoľvek pohodlné experimentovanie s protokolmi *PPTP*, *PPP* a súvisiacimi autentizačnými metódami. Ďalej sme pokračovali popisom implementácie niekoľkých sieťových automatov, ktoré sú kľúčovou súčasťou testovacieho nástroja nazvaného *PPTP Auditor*. Na konci kapitoly sme popísali inštaláciu nástroja *PPTP Auditor* z verejného repozitára a jeho použitie.

V poslednej kapitole tejto práce sme stručne popísali implementáciu jednotkových testov pre testovací nástroj *UTScapy* pokrývajúcich kód pridaný do projektu *Scapy*. V závere informuje o otestovaní nástroja na *PPTP* serveri *PoPToP* a *PPTP* serveri, ktorý je súčasťou operačného systému *Windows Server*.

Výsledkom tejto práce je verejne dostupný nástroj *PPTP Auditor*, ktorý je možné získať z repozitára na serveri *GitHub*[24] a podpora nových protokolov používaných nielen pri *PPTP* spojeniach v projekte *Scapy*[25].

[24] `https://github.com/jsebechlebsky/pptp_auditor`
[25] `https://github.com/secdev/scapy`

Literatúra

[1] Aboba, B.; Blunk, L.; Vollbrecht, J.; aj.: Extensible Authentication Protocol (EAP). RFC 3748, RFC Editor, June 2004. Dostupné z: `http://www.rfc-editor.org/rfc/rfc3748.txt`

[2] Dommety, G.: Key and Sequence Number Extensions to GRE. RFC 2890, RFC Editor, September 2000. Dostupné z: `http://www.rfc-editor.org/rfc/rfc2890.txt`

[3] Erwin, M.; Scott, C.; Wolfe, P.: *Virtual Private Networks: Turning the Internet Into Your Private Network*. O'Reilly Media, 1998, ISBN 1565925297.

[4] Farinacci, D.; Li, T.; Hanks, S.; aj.: Generic Routing Encapsulation (GRE). RFC 2784, RFC Editor, March 2000. Dostupné z: `http://www.rfc-editor.org/rfc/rfc2784.txt`

[5] Griffiths, I.: Raw Sockets Gone in XP SP2. Dostupné 1.3.2017. Dostupné z: `http://www.interact-sw.co.uk/iangblog/2004/08/12/norawsockets`

[6] Hamzeh, K.; Pall, G.; Verthein, W.; aj.: Point-to-Point Tunneling Protocol (PPTP). RFC 2637, RFC Editor, July 1999. Dostupné z: `http://www.rfc-editor.org/rfc/rfc2637.txt`

[7] Keijser, J. J.: EAP-TLS patch for pppd. Dostupné z: `https://www.nikhef.nl/~janjust/ppp/doc_pptp_serversetup.html`

[8] Kosiur, D.: *Building and managing virtual private networks*. John Wiley & Sons, Inc., 1998, ISBN 978-0471295266.

[9] Krahmer, S.: Cheating CHAP. 2002, `http://www.zelja.com/AdminHeaven/misc/security/cheating_chap.pdf`.

[10] Kumar, H.; Kumar, S.; Joseph, R.; aj.: Rainbow table to crack password using MD5 hashing algorithm. In *Information & Communication Technologies (ICT), 2013 IEEE Conference on*, IEEE, 2013, s. 433–439.

[11] L0pht: Windows NT rantings from the L0pht. Dostupné 1.3.2017. Dostupné z: `http://seclists.org/bugtraq/1997/Jul/135`

[12] Lloyd, B.; Simpson, W. A.: PPP Authentication Protocols. RFC 1334, RFC Editor, October 1992. Dostupné z: `http://www.rfc-editor.org/rfc/rfc1334.txt`

[13] Lyon, G. F.: *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Nmap Project, 2009, ISBN 0979958717.

[14] Marlinspike, M.; Hulton, D.: Defeating PPTP VPNs and WPA2 Enterprise with MS-CHAPv2. Dostupné 1.3.2017. Dostupné z: `https://www.youtube.com/watch?v=gkPvZDcrLFk`

[15] McNab, C.: *Network Security Assesment*. O'Reilly Media, druhé vydání, 2007, ISBN 978-0-596-51030-5.

[16] Nmap/Script Ideas. Dostupné z: `https://secwiki.org/w/Nmap/Script_Ideas`

[17] Ristic, I.: *Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications*. Feisty Duck, 2013.

[18] Rodriguez, A.; Gatrell, J.; Karas, J.; aj.: *TCP/IP Tutorial and Technical Overview (7th Edition)*. Prentice Hall, 2001, ISBN 0130676101.

[19] Schneier, B.; Wagner, D.: Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2). *Secure Networking—CQRE [Secure]'99*, 1999: s. 782–782.

[20] Schneier, B.; aj.: Cryptanalysis of Microsoft's point-to-point tunneling protocol (PPTP). In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, ACM, 1998, s. 132–141.

[21] Seitz, J.: *Black Hat Python: Python Programming for Hackers and Pentesters*. No Starch Press, 2014.

[22] Simon, D.; Aboba, B.; Hurst, R.: The EAP-TLS Authentication Protocol. RFC 5216, RFC Editor, March 2008. Dostupné z: `http://www.rfc-editor.org/rfc/rfc5216.txt`

[23] Simpson, W. A.: The Point-to-Point Protocol (PPP). STD 51, RFC Editor, July 1994. Dostupné z: http://www.rfc-editor.org/rfc/rfc1661.txt

[24] Simpson, W. A.: PPP Challenge Handshake Authentication Protocol (CHAP). RFC 1994, RFC Editor, August 1996. Dostupné z: http://www.rfc-editor.org/rfc/rfc1994.txt

[25] Stevens, M.; Bursztein, E.; Karpman, P.; aj.: The first collision for full SHA-1. 2017. Dostupné z: https://shattered.it/static/shattered.pdf

[26] Nessus Plugins. Dostupné 1.3.2017. Dostupné z: http://www.tenable.com/plugins/

[27] The Hacker Choice. Dostupné 3.1.2017. Dostupné z: http://www.thc.org/releases.php

[28] Tipton, H. F.; Nozaki, M. K.: *Information Security Management Handbook, Volume 6*. Auerbach Publications, 2012.

[29] UTScapy: Unit Testing with Scapy. Dostupné z: http://www.secdev.org/projects/UTscapy/

[30] WinPcap Documentation. Dostupné 1.3.2017. Dostupné z: https://www.winpcap.org/docs/docs_412/html/main.html

[31] Young, A.; Palekar, A.; Potter, D.; aj.: Extensible Authentication Protocol (EAP) Registry. Technická zpráva, Internet Assigned Numbers Authority, 2015. Dostupné z: https://www.iana.org/assignments/eap-numbers/eap-numbers.xhtml

[32] Zorn, G.; Cobb, S.: Microsoft PPP CHAP Extensions. RFC 2433, RFC Editor, October 1998. Dostupné z: http://www.rfc-editor.org/rfc/rfc2433.txt

Dodatok **A**

Zoznam použitých skratiek

ASCII American Standard Code for Information Interchange

CHAP Challenge Authentication Protocol

DES Data Encryption Standard

EAP Extensible Authentication Protocol

ECB Electronic Code Book

FPGA Field Programmable Gate Array

GRE Generic Routing Encapsulation

IANA Internent Assigned Numbers Authority

ICMP Internet Control Message Protocol

IP Internet Protocol

MPPE Microsoft Point-to-Point Encryption

MSCHAP Microsoft Challenge Authentication Protocol

NDIS Network Driver Interface Specification

PAP Password Authentication Protocol

PPTP Point to Point Tunelling Protocol

PPP Point to Point Protocol

RAS Remote Access Server

REPL Read-Evaluate-Print-Loop

TCP Transport Control Protocol

TLS Transport Layer Security

VPN Virtual Private Network

Dodatok **B**

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `DP_Sebechlebsky_Jan_2017.pdf` ..... text práce vo formáte PDF
- `pptp_auditor` ..... implemenácia nástroja PPTP Auditor
  - `scapy_pptp` ..... modul s rozšíreniami nezačlenenými do Scapy
  - `pptp_auditor` ..... spustiteľný modul nástroja
  - `LICENSE` ..... licencia nástroja PPTP Auditor
  - `README.md` ..... stručný popis nástroja PPTP Auditor
  - `pptp_auditor.sh` ..... pomocný skript pre inštaláciu a spustenie
  - `setup.py` ..... inštalačný skript nástroja PPTP Auditor