FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Komunikace skrze Captive portal
Student:	Bc. Martin Černáč
Vedoucí:	Ing. Aleš Padrta, Ph. D.
Studijní program:	Informatika
Studijní obor:	Počítačové systémy a sítě
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

1. Seznamte se s problematikou Captive portals a způsoby jejich obcházení.
2. Navrhněte protokol umožňující obejití Captive portals s důrazem na co nejvyšší propustnost.
3. Navržený protokol implementujte.
4. Výsledky vyhodnoťte a porovnejte s dostupnými řešeními.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 10. listopadu 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Komunikace skrze Captive portal

Bc. Martin Černáč

Katedra počítačových systémů
Vedoucí práce: Ing. Aleš Padrta, Ph. D.

8. května 2018

Poděkování

Rád bych poděkoval svému vedoucímu za cenné rady, věcné připomínky a vstřícnost při konzultacích.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 8. května 2018 .....

Odkaz na tuto práci

Černáč, Martin. *Komunikace skrze Captive portal.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018. Dostupný také z WWW: ⟨`https://github.com/octaroot/CTU-FIT-MasterThesis`⟩.

Abstrakt

Captive portál je software, zpravidla nasazován na veřejně dostupných sítích, který typicky omezuje či limituje přístup do sítě Internet. Tato diplomová práce se věnuje problematice obcházení takových limitací pomocí síťových tunelů. V rámci práce jsou vybrané tunelovací techniky implementovány a srovnány s dostupnými řešeními.

Klíčová slova Captive portál, síťový tunel, firewall, exfiltrace dat

Abstract

Captive portal is a piece of software, usually deployed on publicly available networks. The goal of such software is to control or limit access to the Internet. The goal of this thesis is to overcome these limitations by using network tunnels. The main focus of this thesis is to analyze, implement, test and compare selected methods of creating network tunnels.

Keywords Captive portal, network tunnel, firewall, data exfiltration

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Bezdrátové sítě se staly zcela běžným prostředkem mezilidské komunikace. Uživatelé bezdrátové sítě mají možnost si navzájem vyměňovat informace a nebýt přitom omezeni kabelovým spojením. Velkým přínosem bezdrátové sítě je tedy zvýšená mobilita uživatelů. Ta vedla k vlně popularity bezdrátových sítí počínaje mobilními telefony, využívajících bezdrátovou síť `GSM`, až po dnešní chytré spotřebiče a jejich zapojení do *Internet of Things*.

S rostoucími nároky uživatelů prošly rozsáhlým vývojem i bezdrátové sítě (vyšší prostupnost, nižší latence a další aspekty). Mezi dlouhodobě populární a velmi rozšířené typy bezdrátových sítí se řadí technologie `Wi-Fi`. Jedná se o technologii podporovanou širokým spektrem spotřební elektroniky (například televizory, tiskárny, mobilní telefony nebo počítače). Technologie `Wi-Fi` využívá bezlicenčním pásmo `ISM` a díky tomu je provozování vlastní `Wi-Fi` sítě legislativně nenáročné. Na trhu je navíc dostupná celá řada produktů, zajišťující provoz `Wi-Fi` sítě.

Z těchto důvodů došlo k velkému rozmachu takzvaných *hotspotů*, tedy veřejně přístupných míst s pokrytím `Wi-Fi` sítě. Taková `Wi-Fi` síť je zpravidla veřejně přístupná a uživatelům nabízí přístup do sítě Internet. Ačkoliv je velice snadné začít s provozem *hotspotu*, je nutné dbát na další aspekty provozu takové služby – zejména právní aspekty.

Uživatelé *hotspotu* by měli být srozuměni s pravidly používání konkrétní sítě, limitovanou odpovědností provozovatele a před začátkem užívání sítě doložit svůj souhlas s pravidly. Provozovatel navíc může mít zájem o některé identifikující informace o uživatelích *hotspotu*.

Technologie `Wi-Fi` však sama o sobě neumožňuje nic z výše uvedeného. Takovou situaci lze vyřešit například zapojením recepce v prostředí hotelu (uživatel písemně vyjádří souhlas s pravidly používání sítě, recepční vydá přístupové údaje do sítě). Častěji se však setkáváme s automatizovaným přístupem, realizovaným pomocí *captive portálu* (z angličtiny *Captive portal*) – a to jak na návštěvnických sítích drátových, tak i bezdrátových.

Řešení s pomocí *captive portálu* spočívá v detekci nově připojených uživa-

telů, které je nutné informovat o pravidlech provozu sítě. Po udělení souhlasu s pravidly je uživateli poskytnut přístup do Internetu a všechny následné interakce uživatele se sítí *captive portál* ignoruje (nezasahuje do nich).

Z principu věci tedy *captive portál* musí být schopen **nejprve zasahovat do veškerého síťového provozu** (uživatel doposud nedal souhlas s pravidly, neměl by mít možnost síť využívat) a **následně do provozu konkrétního uživatele nezasahovat vůbec**. Existuje celá řada technologických postupů pro docílení popsaného efektu. Mnohé z nich jsou však neefektivní a nepočítají s „neposlušným“ uživatelem, který se bude snažit omezující techniky překonat.

Právě proto jsem se rozhodl vypracovat diplomovou práci na téma obcházení *captive portálu*, zdůrazňující jejich technologickou nedokonalost a poukázat na lepší řešení řízení síťového přístupu (*Network Access Control*).

V této práci se proto budu zabývat popisem problematiky *captive portálů* a obecnými způsoby jejich obcházení. Jako demonstraci technologické nedokonalosti užití *captive portálu* pro zajištění řízení síťového přístupu rovněž navrhnu a implementuji protokol s důrazem na maximální prostupnost. Implementovaný protokol otestuji a provedu srovnání s dostupnými nástroji pro obcházení *captive portálů*.

Kapitola 1

Analýza současné situace

Tato kapitola se věnuje problematice *captive portálů*, motivací jejich nasazení v síti a častými problémy s používáním *captive portálu* jako nástroje pro zajištění řízení síťového přístupu.

1.1 Captive portál

Captive portál[1] představuje webovou aplikaci, často nasazovanou na veřejně přístupných sítích. Aplikace má za úkol informovat nově připojené klienty o podmínkách užití sítě a požadovat uživatelův souhlas s jejich dodržováním. Až do momentu souhlasu s podmínkami užití sítě je uživateli odepřen přístup do zbytku sítě. Z toho plyne první část názvu ***Captive*** *portál* – uživatel je „zajatý“, „uvězněný“ (v angličtině *captive*).

Pojem *captive portál* nemá v češtině ustálený překlad, v jiných akademických publikacích [2] [3] autoři používají počeštěný termín *captive portál* a proto je tento termín použit i v této práci.

1.1.1 Motivace nasazení

Captive portál je do provozu sítě často nasazován jako nástroj pro zajištění řízení síťového přístupu. Přístup do sítě je umožněn pouze klientům, kteří splní podmínky přístupu do sítě. Takovou podmínkou může být pouhé vyjádření souhlasu s používáním konkrétní sítě, ale může se jednat i o podmínku složitější, například:

- shlédnutí reklamního spotu dle výběru provozovatele
- uhrazení poplatku pro přístup do sítě
- poskytnutí některých osobních údajů a souhlasu s jejich zpracováním
- doložení oprávnění pro přístup do sítě (kód z účtenky, číslo hotelového pokoje, ...)

- zviditelnění provozovatele pomocí sociálních médií (například Facebook *check-in*)

Jak plyne z výše uvedeného výčtu, vyjma právních aspektů může být *captive portál* použit i pro shromažďování údajů o uživatelích sítě. Jedním z nástrojů pro takovou činnost je nabízení „přihlášení se“ do *captive portálu* pomocí účtu na některé ze sociálních sítí. Pokud uživatel takovou možnost využije, *captive portál* si od sociální sítě vyžádá informace o uživateli, jako například jméno, fotografii, pohlaví nebo datum narození. Po shromažďování takových informací je uživateli poskytnut přístup do zbytku sítě. Provozovatel tedy může uživatele například identifikovat nebo detekovat opakované návštěvy *hotspotu*. Na oplátku je uživateli „zdarma“ poskytnut přístup do sítě Internet.

Pro usnadnění nasazení takového řešení nabízí společnost Facebook službu *Facebook Wi-Fi*[4], cílenou na majitele obchodů. Jedná se o řešení na bázi *captive portálu*, které vyžaduje aby nově připojený uživatel měl konto na sociální síti Facebook. Po připojení na *hotspot* je uživatel vyzván ke sdílení informace o jeho návštěvě obchodu, jehož *hotspot* právě používá (jako protislužbu za poskytnutý přístup do Internetu).

Poněkud méně invazivní motivací pro zavedení *captive portálu* je monetizace *hotspotu*. Například prodejem reklamního místa – uživatel po připojení do sítě musí shlédnou reklamní spot, nebo vyplnit krátkou anketu. Provozovatel *hotspotu* získá z takové aktivity finanční odměnu a uživateli je odměněn přístupem do sítě Internet.

Některé *captive portály* alternativně umožňují uživateli doložit nárok na přístup do sítě. Například jednorázový kód z účtenky, čímž dokládá útratu v podniku, který *hotspot* provozuje. Nebo číslo hotelového pokoje, čímž dokládá svůj pobyt v hotelu, který zahrnuje (jinak zpoplatněný) přístup do sítě Internet.

1.1.2 Technologické pozadí

Úkolem *captive portálu* je detekovat nově připojené uživatele sítě, omezit jim přístup do sítě a nasměrovat je na webovou aplikaci captive portálu. Po splnění podmínek pro plnohodnotný přístup uživatele do zbytku sítě nesmí *captive portál* do komunikace dále zasahovat (tj. musí *detekovat*, že síťový provoz patří oprávněnému uživateli).

Ačkoliv se jedná o přímočarý cíl, je možné ho dosáhnout s pomocí celé řady technologií a postupů. Proto je možné se v praxi setkat s velmi vysokým počtem různorodých implementací *captive portálu*. Některé z nich jsou dostupné pod svobodnou licencí, jiné jsou součástí placeného produktu a v neposlední řadě existují řešení *na míru* – a to nejen *na míru* provozovateli, ale rovněž *na míru* konkrétnímu zařízení/hardware.

Přestože efektu *captive portálu* lze s velkou úspěšností docílit pouhým odkloněním HTTP provozu, existují mnohem sofistikovanější varianty, využívající například oddělené VLAN sítě. Obecně však platí, že *captive portál* při své práci může vycházet pouze z informací, které putují po síti. Detekce nově připojených uživatelů a identifikace oprávněných uživatelů je tedy zpravidla založena dvojici identifikátorů:

- globálně unikátní `MAC` adresa zařízení
- přidělená `IP` adresa zařízení

Captive portál lokálně ukládá informace o autentizovaných uživatelských zařízeních v síti (zaznamenává jejich `MAC` a `IP` adresy). Síťový provoz takových zařízení není narušován. Pokud však uživatel využívá zařízení, které *captive portál* na svém seznamu nenalezne, *captive portál* síťový provoz buď zahodí, nebo zmanipuluje takovým způsobem, aby se uživatel dostal na webovou aplikaci *captive portálu* a mohl se identifikovat. Záznamy na seznamu autentizovaných uživatelů sítě zpravidla podléhají periodickému mazání neaktivních uživatelů – uživatel je tedy nucen se po delší době nečinnosti opakovaně identifikovat *captive portálu*.

Alternativně k periodickému promazávání seznamu autentizovaných klientů může *captive portál* vyžadovat, aby uživatel po celou dobu používání sítě měl v prohlížeči otevřené speciální okno, jehož přítomnost instruuje *captive portál* k přidělení plnohodnotného síťového přístupu.

Ve chvíli, kdy je *captive portál* schopen rozeznat autentizované a neautentizované uživatele, musí rovněž mít možnost neautentizované uživatele nasměrovat na webovou aplikaci *captive portálu*. Takový cíl *captive portál* často naplňuje prováděním `MITM` útoku na nově připojené uživatele. Například při přístupu neautentizovaného uživatele na libovolnou webovou stránku protokolem `HTTP` je jeho provoz odkloněn a vrácena odpověď od *captive portálu*, která prohlížeč uživatele nasměruje na webovou aplikaci *captive portálu*. Kromě této techniky uvádím v následující části textu i několik dalších.

1.1.2.1 `ICMP` host redirect

Protokol `ICMP` specifikuje zprávy, které může směrovač poslat koncové stanici, pokud detekuje, že stanice v rámce své komunikace používá neoptimální síťovou cestu. Je zcela v režii cílové stanice, zda-li si nechá o svém směrování radit od ostatních zařízení v síti. Tato metoda spoléhá na situaci, kdy koncová stanice skutečně upraví svou směrovací tabulku a zanese do ní informace z `ICMP` *host redirect* zprávy. Právě s tímto úmyslem odesílá *captive portál* `ICMP` *host redirect* zprávu, když detekuje pokus o spojení uživatele se serverem v Internetu. `ICMP` zpráva se pokusí cílovou stanici uživatele přesvědčit, že ideální cesta vede skrze server provozující *captive portál*. Koncová stanice upraví své směrování a začne komunikovat se svým protějškem skrze *captive portál*, který

díky tomu může komunikaci manipulovat za účelem nasměrování uživatele na webovou aplikaci *captive portálu*.

1.1.2.2 HTTP 3xx redirect

Při pokusu o přístup na webovou stránku `www.example.com` je požadavek klienta odkloněn a odpověď na požadavek zaslána přímo z *captive portálu*. V odpovědi je zpravidla využita `HTTP` hlavička `302 Found`, která prohlížeč klienta nasměruje na webovou aplikace *captive portálu*, viz ukázka 1.1.

```
> GET / HTTP/1.1
> Host: www.example.com
>
< HTTP/1.1 302 Found
< Location: http://192.168.1.1/captive/
```

Ukázka 1.1: Ukázka přesměrování `HTTP` požadavku (zkráceno)

1.1.2.3 Podvržení DNS odpovědi

Captive portál monitoruje `DNS` dotazy klientů. Pokud `DNS` požadavek patří neautentizovanému klientovi, *captive portál* mu nazpět zašle odpověď s `IP` adresou webové aplikace *captive portálu* bez ohledu na dotazované doménové jméno. Jedná se o značně nebezpečnou techniku, protože může snadno dojít k otrávení `DNS` cache klienta. Pro minimalizaci takového vedlejšího efektu bývá v podvržené `DNS` odpovědi nastavena nulová životnost (hodnota `TTL = 0`). Takové nastavení by mělo zajistit, že podvržená odpověď nebude zanesena do lokální `DNS` cache. Ukázka 1.2 zachycuje evidentní podvržení `IP` adresy serveru `google.com`.

```
$ nslookup google.com
Server:         192.168.1.1
Address:        192.168.1.1#53

Non-authoritative answer:
Name:  google.com
Address: 192.168.1.1
```

Ukázka 1.2: Ukázka podvržení `DNS` odpovědi

1.1.3 Technické problémy

Největším problémem *captive portálů* je závislost na technologii `WWW`. Cílení na tuto technologii pramení ze značně rozmanitého pojetí Internetu napříč jeho

uživateli. Pro mnohé uživatele je totiž tvrzení „Nefunguje Internet“ synonymem pro „V prohlížeči se nepodařilo načíst mou domovskou stránku“. Díky tomu lze mnohé uživatele přesvědčit k provedení úkonů, které *captive portál* vyžaduje. Uživatel úkony provede, protože mu „nefunguje Internet“ a *captive portál* slibuje nápravu situace.

Z předcházejících tvrzení však plyne fakt, že *captive portál* je **závislý** na `WWW` a tím pádem **závislý na webovém prohlížeči**. V historii se ukázalo, že to představuje velký problém pro zařízení s podporou `Wi-Fi`, ale bez webového prohlížeče (nebo s velmi omezeným webovým prohlížečem). Demonstrovat takovou situaci lze na populárním[1] mobilním herním zařízení *Nintendo DS*. Tento problém v současnosti řeší protokol *WISPr*[6], který usnadňuje (v některých případech zcela eliminuje) nutnou interakci uživatele s webovou aplikací *captive portálu*.

S rostoucím rozmachem `HTTPS` na úkor nešifrovaného `HTTP` mají *captive portály* obtížnější práci s nasměrováním uživatele na webovou aplikaci *captive portálu*. *Captive portály* využívající podvržené certifikáty se budou muset od dubna 2018 vyrovnat s ještě větším stupněm nedůvěryhodnosti, díky zavedení nutnosti *Certificate Transparency* v prohlížeči Google Chrome[7]. *Captive portál* by se neměl snažit manipulovat s šifrovaným spojením, namísto snahy o modifikaci a *rozbití šifrování* by takový provoz měl být zahazován. Takový postup však nesdílejí všechny implementace *captive portálu*, jak je dále popsáno v podkapitole 1.1.4 *Netechnické problémy*.

Návrhovým problémem mnoha *captive portálů* je snaha manipulovat s obsahem komunikace uživatelů sítě. V mnohých případech je manipulace dosaženo pomocí `MITM` útoku. Síť, která zcela úmyslně provádí útoky na své uživatele (ať už s jakýmkoliv účelem) pochopitelně nemůže získat jakoukoliv důvěru uživatelů. **Síť s nulovou důvěrou by uživatelé neměli vůbec využívat**.

Mnohé softwarové produkty dokáží detekovat omezený síťový provoz – například operační systém Microsoft Windows, nebo webové prohlížeče Firefox a Chrome. Nadměrná manipulace se síťovým provozem neautentizovaných uživatelů však může tuto funkcionalitu potlačit, což je pro uživatele nežádoucí.

Jak bylo uvedeno v podkapitole 1.1.2 *Realizační technologie*, *captive portál* při své práci vychází z dat, která putují po síti. Do veřejné sítě *hotspotu* je však jednoduché získat přístup. Útočník na zmíněné síti může naslouchat a například pomocí naklonování `MAC` a `IP` adres se následně vydávat za jiné účastníky sítě, čímž se neautorizovaný útočník jeví *captive portálu* jako autentizovaný uživatel.

[1] prodáno přes 150 milionů kusů[5]

1.1.4 Netechnické problémy

V některých případech se *captive portály* chovají velmi invazivně. Na začátku roku 2015 společnost Gogo (poskytovatel připojení na palubách letadel) ve své síti začala využívat falešné certifikáty pro produkty firmy Google. Na situaci upozornila na svém Twitteru[8] Adrienne Porter Felt, zaměstnankyně firmy Google. Certifikáty byly vystaveny pro doménová jména `*.google.com`, tedy všechny domény třetího řádu domény `google.com`.

Mnoho uživatelů Internetu má ve svých prohlížečích nastavenou domovskou stránku na `www.google.com`. Po připojení se na palubní `Wi-Fi` síť v letadle a zapnutí prohlížeče byl uživatel okamžitě varován před nedůvěryhodným certifikátem. Vzhledem k tomu, že uživatel sám žádnou stránku nenavštívil (prohlížeč pouze načetl domovskou stránku), je pro uživatele snadné propadnout dojmu, že chyba není způsobena jeho počínáním a proto bude varování ignorovat.

Takové počínání samozřejmě není správné a poučená osoba by se ho měla vyvarovat. Zdaleka ne všechny uživatele Internetu však lze označit jako *poučené* uživatele. Běžní uživatelé nedisponují dostatečnými znalostmi pro porozumění problému, před kterým je prohlížeč varuje a varování budou ignorovat. **Vytvářet u uživatelů návyky „všechno potvrď a pak se dostaneš na Internet“ je neetické** a nemělo by k tomu docházet.

V případě *captive portálu*, který vyžaduje poskytnutí osobních informací by jejich počet měl být minimální a nakládání s nimi obezřetné. Uživatelé *hotspotu* zpravidla nemají zájem o *newsletter* provozovatele, ani si nepřejí být provozovatelem statisticky zkoumáni. Provozovatel si na takové akce samozřejmě vyhradí nárok v pravidlech používání sítě, které však (zpravidla na mobilních zařízeních) přečte jen malý zlomek uživatelů.

1.1.5 Alternativy captive portálů

Motivací *captive portálu* je řízení síťového přístupu. Takovou funkci však mnohem lépe[2] plní dedikované protokoly a softwarová řešení. Pro řízení přístupu na `Wi-Fi` hotspot lze například použít populární bezpečnostní protokol `WPA2`. Nikoliv však v módu *WPA-Personal*[2], nýbrž v režimu *WPA-Enterprise*. Tento režim vyžaduje, aby se uživatel identifikoval ještě **před** faktickým připojením do sítě – typicky pomocí uživatelského jména a hesla[3]. K ověření údajů tedy není zapotřebí webový prohlížeč, ale klientské zařízení musí podporovat *WPA-Enterprise* režim – nutná podpora pro *IEEE 802.1X* protokol. Příkladem takové sítě je celosvětová síťová infrastruktura *eduroam*, která pro autentizaci využívá protokol *IEEE 802.1X* a hierarchickou strukturu RADIUS serverů. Nasazení *WPA-Enterprise* je však z důvodu nutnosti provozu RADIUS serveru náročnější, než *WPA-Personal*. I přesto se však jedná o technicky vhod-

[2] Často označován jako *WPA-PSK*

[3] Protokol *IEEE 802.1X* podporuje i ověření pomocí certifikátu nebo tokenu

nější alternativu *captive portálu*, pokud je možné provozovat *hotspot* v režimu *WPA-Enterprise*.

1.2 Metody pro obcházení captive portálů

Captive portál s uživateli komunikuje pomocí `WWW`. Aby bylo možné uživatele nasměrovat na webovou aplikaci *captive portálu*, musí být uživatel úspěšně připojen do sítě. Díky takovému „odložení“ autentizace bylo popsáno několik způsobů pro obcházení *captive portálů*. Všechny dále popisované způsoby jsou založeny na neúplné nebo dokonce záměrně „špatné“ konfiguraci *captive portálu*.

Konfigurace firewallu, která úmyslně nefiltruje některý síťový provoz nemusí být dílem nezkušeného administrátora (proto tento stav označuji jako **„špatnou“** konfiguraci). Může se zkrátka jednat o jediný způsob, jak splnit požadavky pro provoz sítě – například kvůli proprietárnímu software, který vyžaduje nerušenou komunikaci na některých portech. Z hlediska síťové architektury by bylo lepší provozovat veřejnou síť s *captive portálem* bez takových klientů, tj. **pouze** jako síť pro hosty, nicméně hardware podporující pokročilé techniky jako provoz více oddělených `Wi-Fi` sítí nebo podporu `VLAN` je zpravidla dražší a pro nezkušené správce obtížnější na správu.

1.2.1 DNS tunelování

Protokol `DNS` je jedním z nejstarších protokolů dnešního Internetu. Slouží primárně k překladu mezi doménovými jmény (například `fit.cvut.cz`) a IP adresami uzlů v síti (například `147.32.232.248`). Častou nedokonalostí *captive portálů* je směrování `DNS` požadavků do Internetu. Pokud k takovému chování dochází i u neautentizovaných uživatelů, lze protokol `DNS` využít ke komunikaci se serverem v Internetu a tím pádem k obejití *captive portálu*.

1.2.2 ICMP tunelování

Protokol `ICMP` je rovněž velmi důležitým síťovým protokolem. Je využíván zpravidla k přenosu služebních informací jako například nedostupnost služby nebo nedosažitelnost uzlu v síti. I přesto, že není v praxi využíván aplikacemi pro přenos informací, lze ho k tomuto účelu využít. Vhodným využitím zpráv `Echo Request` a `Echo Reply` lze mezi dvěma síťovými uzly přenášet libovolná data. Protokol *ICMP* spadá do stejné *rodiny* protokolů jako `TCP` a `UDP`, ale nevyužívá ani jeden z nich. Právě proto bývá v konfiguraci firewallu často opomíjen. Pokud taková situace nastane, lze protokol `ICMP` využít ke komunikaci se serverem v Internetu a tím pádem k obejití *captive portálu*.

Tunelování pomocí `ICMP` je technicky možné díky RFC 792[9], kde je u typů zpráv `0` a `8` (*echo reply*, resp. *echo message*) specifikována proměnlivá délka zpráv.

1.2.3 Využití nefiltrovaných portů

Jak bylo uvedeno na začátku podkapitoly 1.2 *Metody pro obcházení captive portálů*, v konfiguraci firewallu se mohou z různých důvodů vyskytovat výjimky, které lze zneužít k tunelování provozu bez nutnosti maskovat komunikaci jako `DNS` nebo `ICMP` provoz. Zpravidla[10] se jedná o porty

- `TCP/22` – pro vzdálenou správu zařízení,
- `TCP/3128` – HTTP proxy servery (například za účelem cache obsahu),
- `UDP/53` – `DNS`, diskutováno v podkapitole 1.2.1 DNS tunelování.
- `UDP/5060` – `VoIP` telefonie

Důvodem k udělení výjimky pro port `TCP/22` bývá nutnost vzdálené správy některých zařízení pomocí protokolu `SSH`. Samotný protokol `SSH` lze využít pro tunelování, *port forwarding* nebo přímo jako `SOCKS` proxy, pokud komunikující aplikace podporuje nastavení proxy. Klient `OpenSSH`, implementující protokol `SSH`, tyto operace umožňuje provést velmi snadno, například lokální `SOCKS` proxy na portu 8080 lze spustit příkazem `ssh -D 8080 uzivatel@server`

TCP port 3128 bývá na firemních sítích využíván jako cache proxy pro často navštěvované webové stránky, aby se šetřilo síťovým provozem. Neautentizovaný klient se může pokusit takového proxy serveru využít pro obejití omezení *captive portálu* a úspěšně komunikovat se serverem v Internetu.

Tyto praktiky jsou však méně časté než dříve zmíněné `ICMP` a zejména `DNS` tunelování, zkrátka proto že `SSH` ani cache proxy server nejsou na rozdíl od služby `DNS` pro provoz Internetu klíčové.

1.2.4 Využití nefiltrovaných protokolů

Stejně jako `ICMP` tunelování využívá nekompletní sady pravidel blokování provozu na firewallu, lze k tunelování dat skrze *captive portál* využít méně známé protokoly transportní vrstvy. Na transportní vrstvě ISO/OSI modelu figuruje kromě dobře známých protokolů `TCP` a `UDP` rovněž protokoly jako `UDP-Lite`, `SCTP`, `DCCP` nebo `RUDP`. Je možné, že sada restriktivních pravidel na firewallu *captive portálu* bude různými způsoby omezovat provoz `TCP` a `UDP`, ale nebude pamatovat na výše zmíněné méně známé protokoly. Některé z vyjmenovaných protokolů jsou vhodné pro tunelování provozu, zejména se jedná o *relativně* mladý[4] protokol `SCTP`, kterému bude v této práci věnována detailnější pozornost.

[4]Specifikace z roku 2007

1.3 Existující software pro obcházení *captive portálů*

Tato podkapitola krátce shrnuje dostupná řešení pro tunelování dat s využitím rozmanitých protokolů.

1.3.1 Tunelování skrze DNS – 7. vrstva ISO/OSI

Idea tunelování síťového provozu pomocí protokolu DNS není nová. Už na přelomu tisíciletí[5] se objevil nástroj *NSTX* s podtitulkem *tunneling network-packets over DNS*. Od té doby byla zveřejněná řada nástrojů založených na stejných principech a se stejným cílem. Mezi populární[11] nástroje se řadí například *iodine*[6], *OzymanDNS* a *DNSCat*. Různé nástroje nabízejí různé funkce, podporují rozdílné platformy a liší se v konkrétních detailech DNS komunikace (autentizace, šifrování, užité typy DNS zpráv, ...). Mnohé aplikace jsou v současnosti funkční, ale dále nevyvíjené ve prospěch jiných nástrojů (například domovská stránka *NSTX* odkazuje zájemce na stránky *iodine*). Tunelování síťového provozu pomocí DNS je populární[11] i mezi tvůrci škodlivého software (*malware*), kteří se tak snaží vyhnout detekčním nástrojům. Paradoxně tunelování pomocí DNS lze zpravidla úspěšně detekovat[12].

1.3.2 Tunelování na transportní vrstvě – 4. vrstva ISO/OSI

„Skutečného“ tunelování lze dosáhnou s řadou protokolů transportní vrstvy ISO/OSI modelu. Příkladem je software *OpenVPN*, který při dostatečně upravené konfiguraci je schopen komunikovat na TCP/UDP portech, které by potenciálně mohly mít výjimky ve firewallu *captive portálu*.

sshuttle V případě možnosti navázání spojení pomocí SSH skrze *captive portál* na uzel v Internetu je přípustné využít SSH-implementovanou SOCKS proxy. Takové řešení připadá v úvahu pokud komunikující aplikace podporuje explicitní nastavení parametrů a typu SOCKS proxy. Protože však protokol SSH aplikační vrstvy ISO/OSI modelu staví na protokolu TCP transportní vrstvy, nejedná se o ideální řešení. Podrobnosti problému jsou detailně popsány v podkapitole 3.4.1 *Potíže s TCP jako prostředkem pro tunelování*.

Těchto potíží si je vědom autor projektu [13, sshuttle], jehož cílem je zpracování a agregace TCP provozu před odesláním skrze TCP spojení. Díky tomu dosahuje spojení lepších vlastností, než klasické tunelování skrze SSH. Požadavkem pro úspěšné spojení je však možnost vytvoření TCP spojení, která nemusí být vždy k dispozici.

[5] soudě dle data první veřejné verzovacího systému nástroje *NSTX*

[6] Název *iodine* je založen na první třech znacích *iod – IP-over-DNS* a faktu, že jód (anglicky *iodine*) má atomické číslo 53 – port používaný pro DNS komunikaci

1.3.3 Tunelování skrze ICMP – Internetová vrstva TCP/IP

Situace s nástroji pro tunelování pomocí ICMP není tak rozmanitá, jako s nástroji pro DNS tunelování. ICMP tunelování nevyužívá sofistikovaných „triků“ samotného protokolu pro výměnu informací. Přenášená data nemusí nijak transformovat a jednoduše je přenese jako součást zprávy. Blokování takového tunelování je snazší a nevyžaduje dedikovaný a funkčně značně omezený server, jako v případě DNS tunelu.

Kapitola 2

Návrh

Stěžejním cílem této diplomové práce je vytvoření protokolu pro obejití *captive portálů* s důrazem na co největší prostupnost. Tato kapitola shrnuje návrh takového protokolu.

2.1 Dosažení maximální prostupnosti

Navržený protokol by při obcházení omezení *captive portálu* měl upřednostňovat síťovou propustnost. Za tímto účelem bude vyvinutý software mít k dispozici více možných způsobů obejití *captive portálu* a na základě naměřených dat se bude schopen rozhodnout, který způsob tunelování je nejefektivnější, případně jaká kombinace více tunelů poskytuje nejlepší výsledky.

2.2 Možné technické prostředky

Integraci síťového tunelu do operačního systému lze řešit řadou způsobů v závislosti na operačním systému, jeho verzi a v závislosti na požadavcích pro přenositelnost. Snahu o nalezení ideálního řešení projevila celá řada softwarových projektů, jako například:

- *Tor*, známý *open-source* software pro stejnojmennou síť, která umožňuje anonymizaci uživatelů při pohybu na Internetu,
- *OpenVPN*, populární *open-source* software zejména pro vytváření šifrovaných tunelů (VPN).

Každé z výše uvedených softwarových řešení přistupuje k problému jinou cestou. Software anonymizační sítě *Tor* na klientské stanici vytváří `SOCKS` proxy a umožňuje jiným programům komunikovat skrze tuto proxy službu. Jedná se o velmi dobře přenositelné řešení, protože nespoléhá na specifickou podporu operačního systému. Nevýhodou tohoto řešení je přenesení problému

kompatibility z operačního systému na jednotlivé aplikace. Pokud aplikace nepodporuje, nebo uživateli nedovolí nastavit komunikace skrze `SOCKS` proxy, nebude schopna anonymizační sítě *Tor* využít.

Oproti tomu technologie *OpenVPN* je silně vázána na podporu ze strany operačního systému. Na klientské stanici vytváří virtuální síťové rozhraní, které se uživateli jeví jako jakékoliv jiné síťové rozhraní. Lze upravit systémovou směrovací tabulku, aby preferovala virtuální síťové rozhraní a komunikující aplikace tudíž nemusí podporovat komunikaci skrze `SOCKS` proxy. Nevýhodou je závislost na podpoře virtuálních síťových rozhraní v operačním systému.

2.3 Vybrané technické prostředky

Pro tuto práci byla zvolena implementace pomocí virtuálního síťového rozhraní, neboť toto řešení lze považovat za obecnější, neboť není nutná přímá podpora koncových aplikací. Podpora virtuálních rozhraní je zahrnuta v Linuxovém jádře od verze 2.2 (vydáno v roce 1999), *FreeBSD* 3.0 a *Solaris* 2.6. Z ostatních rodin operačních systémů je částečná podpora zahrnuta rovněž v *macOS*, *iOS* a *Andorid*. Pro platformu *Microsoft Windows* existují doplňky třetích stran, které přináší podporu virtuálních rozhraní.

Pro implementaci tunelu byl zvolen typ rozhraní *TUN*, které simuluje síťové rozhraní na třetí vrstvě ISO/OSI modelu a síťová data předává danému uživatelskému programu. Vzhledem k velmi silné vazbě linuxového jádra na jazyk *C* je software pro diplomovou práci vytvořen rovněž v jazyce *C*. Velká část softwarového řešení provádí nízkoúrovňové operace, pro které mají jiné jazyky omezenou podporu. Jedná se zejména o pokročilou práci s *file descriptory* virtuálních síťových rozhraní a pokročilou práci se síťovými sokety.

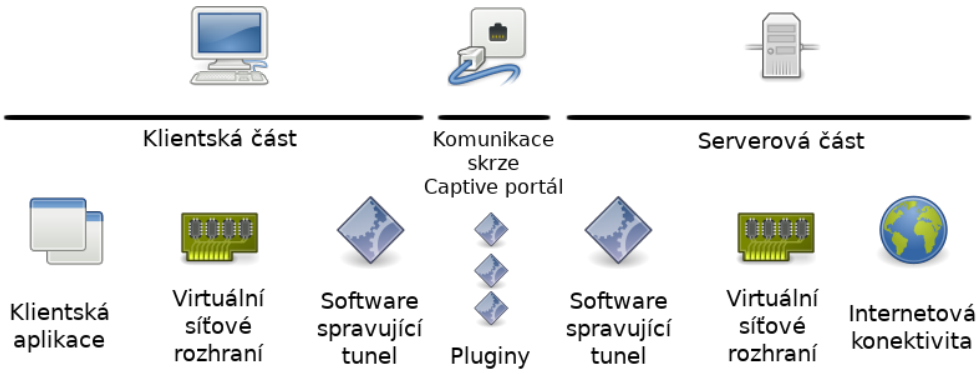
Virtuální síťové rozhraní navíc poskytuje uživateli volnost užití libovolného `IP` síťového rozsahu. Pokud by uživatel vyžadoval existenci *SOCKS* proxy, lze ji vytvořit po navázání tunelu například pomocí nástroje `SSH` jak bylo popsáno v podkapitole 1.2.3 *Využití nefiltrovaných portů*.

2.4 Struktura softwarového řešení

Vzhledem k důrazu na snadnou rozšiřitelnost je softwarové řešení navrženo do dvou částí:

- hlavní část programu, která vytváří a spravuje virtuální síťové rozhraní,
- samostatné pluginy, které řeší pouze omezenou funkcionalitu spojenou s tunelováním pomocí konkrétní technologie.

Hlavní část programu má rovněž na starost vytvoření prostředí pro jednotlivé pluginy. Každý plugin totiž pracuje ve svém vlastním vlákně a komunikuje s jednotným virtuálním rozhraním. Komunikace probíhá skrze funkce, které



Obrázek 2.1: Grafické znázornění komunikace skrze *Captive portál*. Klientská komunikace transparentně komunikuje skrze virtuální síťové rozhraní, které spravuje implementovaný software spravující tunel. Data, zapsaná do aplikací do rozhraní, jsou přečtena klientskou instancí implementovaného softwarového řešení, distribuována mezi aktivní pluginy, které různými způsoby obchází *Captive portál*, aby serverové protějšky pluginů data přijaly, předaly serverové části spravující tunelové spojení a ta data následně zapsala do virtuálního síťového rozhraní na straně serveru.

mají standardem `POSIX` garantovánu bezpečnost při paralelním přístupu[7]. Hlavní část programu tedy při spuštění zpracuje uživatelské vstupy, vytvoří virtuální síťové rozhraní, spustí pluginy a na konci práce programu pluginy ukončí a rozhraní zruší. *File descriptor* síťového rozhraní je v programu sdílen (pouze pro čtení) všemi pluginy.

Jednotlivé pluginy jsou schopny ověřit možnost navázání spojení skrze *captive portál* a změřit datovou prostupnost tunelu. Každému vytvořenému pluginu je věnována jedna část kapitoly 3 Implementace, vysvětlující vnitřní činnost pluginu.

2.5 Komunikace mezi serverem a klientem

Vzhledem k předpokládanému užití softwarového řešení (obcházení limitací *captive portálu* pro osobní použití) bude software vytvořen jako jeden celek, který je schopen plnit funkci jak serveru, tak klienta. Volba plyne z uživatelem specifikovaných přepínačů při spuštění. Z toho vyplývá i základní softwarový návrh komunikace stylu klient – server, kterou započne klient a server zpočátku pouze reaguje na jeho požadavky a až po úspěšném navázání tunelového spojení (autentizace) je zahájena obousměrná komunikace.

I přes rozmanité metody dosažení tunelového spojení bude každý z implementovaných pluginů komunikovat pomocí dvou základních kategorií zpráv:

[7] Anglicky vlastnost *thread-safety*

- servisní zprávy pro řízení tunelového spojení,
- datové zprávy, nesoucí samotná data tunelového spojení.

Je zodpovědností jednotlivých pluginů, jak mezi těmito kategoriemi zpráv rozlišovat.

2.5.1 Popis základních servisních zpráv

Každý z implementovaných pluginů se bude navíc v rámci možností řídit následující strukturou servisních zpráv:

`CONNECTION_REQUEST` Tyto zprávy odesílá klient a přijímá server. Pokud by klientovi dorazila zpráva tohoto typu, bude ignorována. Server zprávu přijme a pokud momentálně není k serveru úspěšně připojen žádný jiný klient, server odpoví zprávou typu `AUTH_CHALLENGE`, čímž klienta vyzve k autentizaci. V případě, že při přijetí zprávy `CONNECTION_REQUEST` server již spravuje existující spojení s jiným klientem, bude žadateli odeslána zpráva typu `CONNECTION_REJECT` a požadavek na navázání tunelu tím pádem zamítnut.

`AUTH_CHALLENGE` Tento typ zpráv posílá server klientovi jako jednu možnou odpověď na zprávu typu `CONNECTION_REQUEST`. Server případné příchozí zprávy typu `AUTH_CHALLENGE` ignoruje. Klient po přijetí zpracuje výzvu k autentizaci (protokol *Challenge-Response*, popsáno v následující podkapitole 2.6) a výsledek předá serveru (viz následující odstavec `AUTH_RESPONSE`). V závislosti na tom, jestli server uživatele autentizuje bude klientovi odeslána zpráva typu `CONNECTION_ACCEPT` a tunelové spojení zahájeno, nebo bude jeho požadavek zamítnut zprávou `CONNECTION_REJECT`.

Tato zpráva je doplněna o data, která klient potřebuje k autentizaci. Jedná se o výzvu (*challenge*) serveru, kterou klient musí zpracovat. Délka dat je předem známá na základě volby bezpečnostních prostředků, podrobněji popsaných v následující podkapitole 2.6.

`AUTH_RESPONSE` Zprávy tohoto typu odesílá klient serveru. Klient příchozí zprávy tohoto typu ignoruje. Obsahem zprávy je odpověď *Challenge-Response* protokolu na základě předem přijaté zprávy typu `AUTH_CHALLENGE`. V závislosti na tom, jestli server uživatele autentizuje bude klientovi odeslána zpráva typu `CONNECTION_ACCEPT` a tunelové spojení zahájeno, nebo bude jeho požadavek zamítnut zprávou `CONNECTION_REJECT`.

Tato zpráva je doplněna o data, která klient předává serveru za účelem autentizace. Jedná se o odpověď (*response*) na výzvu (*challenge*) serveru, kterou klient zpracoval a server po předání vyhodnotí. Délka dat je předem známá na základě volby bezpečnostních prostředků, podrobněji popsaných v následující podkapitole 2.6.

`CONNECTION_ACCEPT` Touto zprávou server stvrzuje úspěšné navázání spojení s adresovaným klientem. Server příchozí zprávy tohoto typu ignoruje. Po celou dobu spojení odmítá server jakékoliv další pokusy o připojení jiných klientů.

`CONNECTION_REJECT` Touto zprávou server oznamuje zamítnutí klientova požadavku na vytvoření tunelového spojení se serverem. Server příchozí zprávy tohoto typu ignoruje. Zprávu klient obdrží pokud se pokusí připojit k již obsazenému serveru, nebo se mu nepodaří autentizovat.

2.5.2 Rozšířené a specializované servisní zprávy

Pět výše zmíněných typů servisních zpráv představuje základní sadu zpráv pro komunikaci klienta a serveru, kterou v rámci možností budou dodržovat všechny implementované pluginy. Některé pluginy mohou využívat dalších specializovaných zpráv pro konkrétní účel navázání a spravování komunikace. Tato podkapitola dále shrnuje jejich abstraktní popis. Tyto zprávy nemusí implementovat všechny pluginy, pokud existují jiné technické prostředky pro dosažení jejich funkcionalit.

`KEEPALIVE` Zprávy `KEEPALIVE` slouží k detekci problémů se spojením klienta a serveru. Zprávy odesílá klient (ohlašuje svou aktivitu serveru) a server na ně stejnou zprávou odpovídá (ujišťuje klienta, že spojení je v pořádku). Do jisté míry pakety `KEEPALIVE` napodobují nástroj `ping`. Pokud však opakovaně nedojde k přijetí zprávy `KEEPALIVE`, klient i server začnou považovat spojení za přerušené nebo ukončené. Server poté začne opět přijímat požadavky pro připojení `CONNECTION_REQUEST`.

`DATA` Zprávy tohoto typu nesou data tunelu. Odesílá a přijímá je jak klient, tak server.

2.6 Ověření nového klienta

Serverová část softwarového řešení před navázáním tunelového spojení vyžaduje autentizaci klienta. I přesto, že tato diplomová práce explicitně neřeší otázky soukromí a bezpečnosti Internetových služeb, považuji alespoň základní úroveň kontroly přístupu za adekvátní. Právě proto je jako součást praktické části diplomové práce implementováno ověřování klientů pomocí protokolu *výzva-odpověď* (známý také jako *Challenge-response protocol*).

Protokol *výzva-odpověď* přímo nespecifikuje konkrétní prostředky pro autentizaci – může se například jednat o triviální dvojici údajů jako přihlašovací jméno a heslo, nebo komplexnější metody využívající asymetrické kryptografie. Pro účely této práce je jako technický prostředek použito sdílené tajemství (tzv. `keyfile`), který může obsahovat jako značné množství (řádově tisíce)

bajtů dat, tak i relativně krátké (řádově desítky) znaků. Je tudíž možné jako sdílené tajemství použít buď soubor s binárními daty, distribuován mezi server a klient alternativním kanálem, nebo pouze krátký textový soubor se zapamatovatelným heslem.

Na základě sdíleného tajemství vytvoří klient otisk (*hash*), který na *výzvu* serveru zašle jako *odpověď*. Server stejnou cestou vytvoří svůj otisk dat a výsledek porovná. Pokud jsou otisky stejné, patrně klient a server znají sdílené tajemství a klientovi je povoleno navázat tunelové spojení.

Otisk, nebo-li *hash*, je z dat tajemství generován předvolenou bezpečnou *hashovací funkcí*. Klient zasílá serveru pouze otisk, aby bylo zabráněno odposlechnutí tajemství. Vyzrazení sdíleného tajemství je zabráněno díky jedné ze stěžejních vlastností *hashovací funkce – jednosměrnost*. Díky této vlastnosti je výpočetně extrémně náročné z výstupu (*otisk*) *hashovací funkce* získat vstupní hodnotu (*sdílené tajemství*). Vyzrazení tajemství není však jediným bezpečnostním rizikem při implementaci protokolu *výzvy-odpověď*.

2.6.1 Bezpečnostní aspekty

Útočník, který je schopen odposlouchávat síťový provoz během úspěšné autentizace klienta, může odposlechnutou komunikaci využít k následné vlastní autentizaci. Pro zabránění *útoků přehráním* (známého také jako *Replay attack*) je každý proces autentizace doplněn o unikátní hodnotu (často označovanou jako *sůl*, anglicky *salt*) vázanou na konkrétní požadavek o spojení. Server tuto unikátní hodnotu vygeneruje, zapamatuje si ji a zašle klientovi. Klient následně při vytváření otisku tuto unikátní hodnotu spojí s daty sdíleného tajemství a výsledný otisk tedy závisí nejen na sdíleném tajemství, ale rovněž na unikátní hodnotě požadavku. Díky tomu má každý požadavek jiný *správný* otisk.

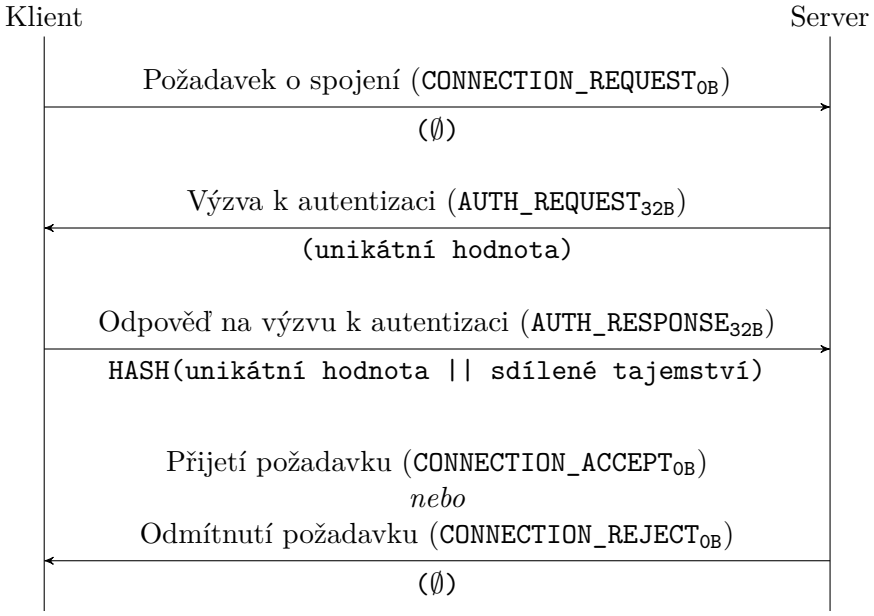
Při použití soli je klíčové vygenerování dostatečně náhodné hodnoty. Pokud není zajištěna dostatečná náhodnost dat, může s dostatečnou znalostí protokolu útočník hodnotu soli uhodnout nebo případně dopředu předvídat. Pro generování pseudonáhodných hodnot je v práci použita knihovna *OpenSSL*, konkrétně funkce `RAND_bytes()`, která dle specifikace[14] generuje kryptograficky silná pseudonáhodná čísla.

2.6.2 Volba bezpečnostních prostředků

Jako hashovací funkce byla pro účely softwarového řešení zvolena funkce `SHA256` z rodiny hashovacích funkcí `SHA` (*Secure Hash Algorithms*). Popsána byla v roce 2001 a v současné době je považována za bezpečnou hashovací funkci, i přestože je náchylná k *Length extension* útoku, který však pro potřeby této diplomové práce není relevantní. Popularitu a rozšířenost `SHA256` lze doložit velmi častým užitím ve světě digitálních certifikátů a také jakou nezbytnou část `SSL/TLS` spojení.

Metoda útoku hrubou silou na unikátní hodnotu každého autentizačního požadavku s přihlédnutím na omezenou délku platnosti požadavku a délku unikátní hodnoty (jednotky až desítky vteřin vs. 256 bitů, tj. 2^{256} možných kombinací) nepředstavuje bezpečnostní hrozbu.

Očekávaná délka dat zprávy `AUTH_CHALLENGE` tedy představuje 256 bitů (32 bajtů) a očekávaná délka dat odpovědi (`AUTH_RESPONSE`) představuje rovněž 32 bajtů.



Obrázek 2.2: Sekvenční diagram autentizace klienta pomocí protokolu *výzva-odpověď*

Kapitola 3

Implementace

Po návrhu struktury softwarového řešení této diplomové práce následuje samotná implementace. Tato kapitola proto popisuje implementaci řídící části software a jednotlivých *pluginů* pro vytvoření síťového tunelu.

3.1 Řídící část softwarového řešení

Hlavní část programu má na starosti nastartování virtuálního síťového řešení (a v konečné fázi jeho korektní uzavření), kontrolu a přípravu prostředí pro jednotlivé pluginy a umožnění distribuce práce mezi jednotlivými pluginy (a jejich případnými kombinacemi). Dále zahrnuje pomocné části kódu, společné pro všechny pluginy (například nakládání se sdíleným tajemstvím pro autentizaci).

Software vykonává celou řadu nízkoúrovňových operací a značně zasahuje do běhu operačního systému (vytváření nových síťových rozhraní). Z těchto důvodů je nutné software spouštět s oprávněním uživatele `root`.

3.1.1 Požadavky na prostředí

Pro správnou funkci pluginu pro tunelování pomocí protokolu `SCTP` je nutná podpora tohoto protokolu v rámci jazyka C. Například pro překladač `gcc` ji lze zapnout přepínačem `-lsctp`.

Generování pseudonáhodných dat a výpočet zvolené hashovací funkce je závislé na podpoře knihovny `OpenSSL` (funkce `RAND_bytes` a `SHA256`). Pro překladač `gcc` ji lze zapnout přepínačem `-lssl` nebo případně `-lcrypto`.

K zajištění flexibility více souběžně pracujících komunikačních pluginů je každému pluginu vytvořeno samostatné pracovní vlákno. Pro práci s vlákny byla zvolena knihovna `OpenMP`, pro překladač `gcc` ji lze zapnout přepínačem `-fopenmp`. Knihovna `OpenMP` byla sice primárně určena pro datový paralelismus na úrovni cyklů, ale od verze `3.0` (současná verze `4.5`) podporuje rovněž funkční paralelismus pomocí direktivy `task`. Navíc nabízí dobře čitelný zápis

a je běžně používanou knihovnou pro paralelizaci, jak zachycuje ukázka kódů 3.1.1.

Hlavní část programu rovněž naslouchá na systémové signály typu `SIGINT` a `SIGTERM`, díky čemuž je schopna správně ukončit komunikaci, zavřít otevřené sokety a zrušit virtuální rozhraní, pokud se uživatel rozhodne aplikaci ukončit například pomocí `Ctrl+C`.

```
1  plugin plugins[] = { [...] };
2
3  void muxStart(uint32_t endpoint, bool serverMode)
4  {
5      #pragma omp parallel num_threads(PLUGIN_COUNT)
6      #pragma omp single nowait
7      for (int i = 0; i < PLUGIN_COUNT; ++i) {
8          #pragma omp task
9          plugins[i].start(endpoint, serverMode);
10     }
11 }
```

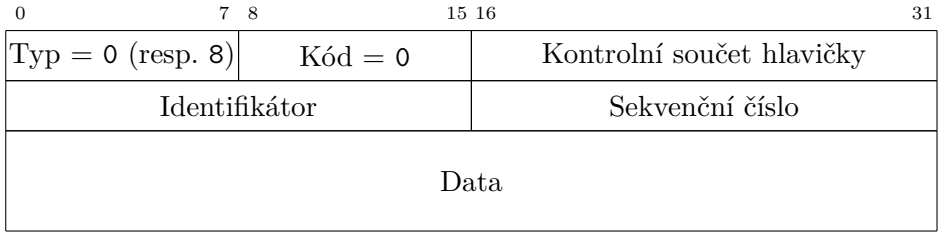
Ukázka kódu 3.1: Výňatek souboru `src/mux.c` znázorňující jednoduchost užití knihovny `OpenMP` pro správu vláken pluginů

Systém spuštění `PLUGIN_COUNT` vláken nejprve operačnímu systému sdělí, že následující kód je třeba spustit v `PLUGIN_COUNT` pomocí direktivy na řádku 5 – `num_threads()`, která zajistí spuštění kýženého počtu vláken i přestože neodpovídá dostupným prostředkům (počtu dostupných fyzických jader procesoru). Následující direktiva na řádku 6 (`single nowait`) zajistí, že smyčka na řádcích 7-10 bude vykonána pouze jedním vláknem a následně se provede spuštění požadovaného počtu pluginů, každý ve vlastním vlákně – pomocí direktivy `task`.

3.2 Plugin pro `ICMP` tunelování

Tunelování dat pomocí `ICMP` je technicky možné díky zprávám typu `0` a `8` (`echo reply`, resp. `echo request`), pro které je specifikována proměnlivá délka zpráv. Zprávy těchto typů jsou používány zejména nástrojem `ping` – například pro zjištění dostupnosti cíle, nebo pro informace o latenci spojení. Zdrojová stanice vyšle na cílovou stanici zprávu typu `echo request` a cílová stanice odpoví zprávou `echo reply`. Toto chování je zajištěno v [15, RFC1122]. Strukturu zmíněných `echo ICMP` zpráv znázorňuje následující diagram 3.1.

Protože `ICMP` pakety `echo reply`, (resp. `echo request`) na rozdíl od `UDP` a `TCP` paketů nenesou informaci o zdrojovém/cílovém portu, jsou pro sdružování souvisejících zpráv použity hodnoty *identifikátor* a *sekvenční číslo*. Dle [9, RFC 792] a [16, RFC 3022] mohou síťové prvky, jako například `NAT`, pro rozpoznání souvisejících zpráv tyto hodnoty používat. Specifikace se však nezmiňuje

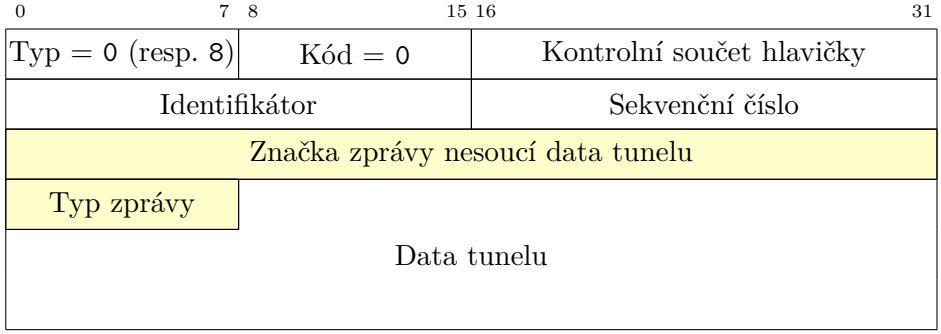


Obrázek 3.1: Diagram ICMP zprávy typu `echo request` (resp. `echo reply`)

o délce platnosti *identifikátoru*. Díky tomu je možno udržovat obousměrnou komunikaci, i pokud je jeden z účastníků omezen NAT, protože je možné klientovi „za NATem“ zaslat více paketů se shodnými hodnotami *sekvenční číslo* a *identifikátor*. Na základě hodnoty *identifikátor* je identifikována i žádost klienta o připojení a následná odpověď na výzvu serveru.

Systém, který dodržuje [15, RFC1122] se však bude snažit na příchozí ICMP *echo request* pakety reagovat. Toto chování je pro správnou funkci tunelu nežádoucí. V Linuxových distribucích je možné povolit ignorování příchozích zpráv ICMP echo změnou souboru `/proc/sys/net/ipv4/icmp_echo_ignore_all`.

Aby obě strany tunelu byly schopny identifikovat tok dat tunelu a rozeznat takové zprávy od jiných ICMP zpráv, je nutné „tunelové“ ICMP zprávy označit. Právě proto začíná každá taková ICMP zpráva stejnou sekvencí čtyř bajtů, která označuje ICMP zprávy, nesoucí data síťového tunelu. Komunikující protějšky kontrolují zdrojovou IP adresu označených ICMP zpráv, aby nebylo triviálně možné injektovat komunikaci do tunelu. Těchto dodatečných pět bajtů je potřebných pro správnou funkci tunelu a předávání servisních zpráv. Umístění těchto informací v ICMP paketu je vyznačeno na diagramu 3.2.



Obrázek 3.2: Diagram ICMP zprávy typu `echo request` (resp. `echo reply`) s vyznačenou hlavičkou dat tunelu

Značku zpráv tunelu má uživatel možnost změnit při kompilaci programu. Výchozí značkou je sekvence bajtů `0x63 0x76 0x75 0x74`, tedy CVUT.

3.2.1 Popis jednotlivých typů ICMP zpráv

Typ zprávy označuje jeden z typů, zachycených v ukázce kódu 3.2.1:

```
1  typedef enum ICMP_PACKET_TYPE
2  {
3          ICMP_CONNECTION_REQUEST,
4          ICMP_AUTH_CHALLENGE,
5          ICMP_AUTH_RESPONSE,
6          ICMP_CONNECTION_ACCEPT,
7          ICMP_CONNECTION_REJECT,
8          ICMP_NATPACKET,
9          ICMP_KEEPALIVE,
10         ICMP_DATA
11 } ICMP_PACKET_TYPE;
```

Ukázka kódu 3.2: Výňatek souboru `plugins/icmp/packet.h` definující typy ICMP zpráv

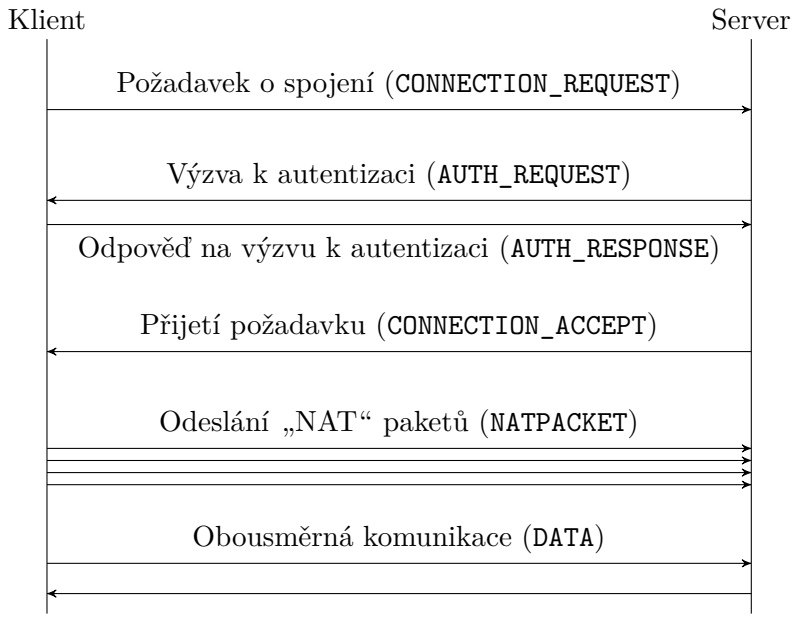
Zprávy typu `CONNECTION_REQUEST`, `AUTH_CHALLENGE`, `AUTH_RESPONSE`, `CONNECTION_ACCEPT`, `CONNECTION_REJECT` a `DATA` jsou implementovány přesně dle specifikace 2.5.1 *Popis základních servisních zpráv.*

Plugin navíc implementuje zprávu typu `NATPACKET`:

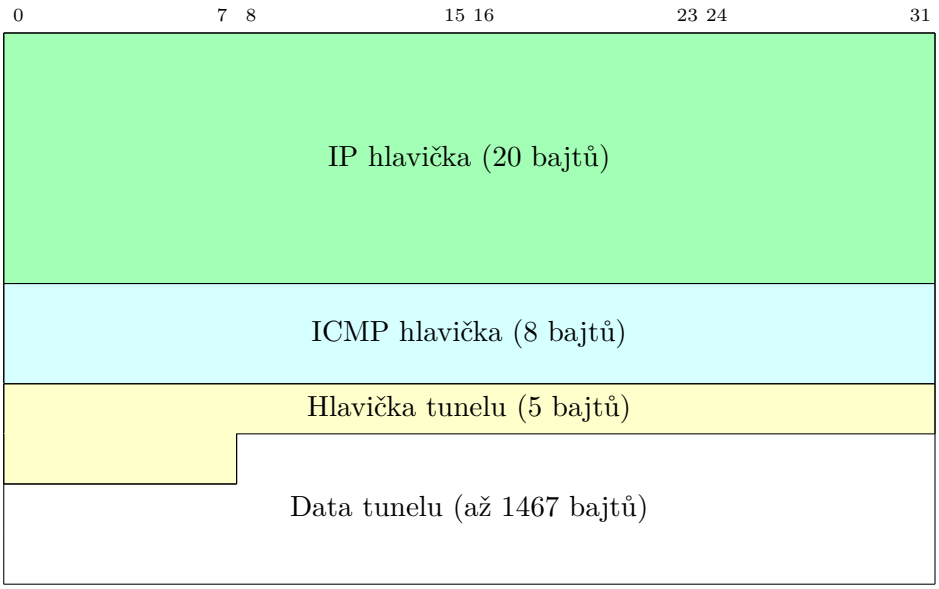
`NATPACKET` Tyto zprávy odesílá klient a přijímá server. Pokud by klientovi dorazila zpráva tohoto typu, bude ignorována. Server zprávu přijme a poznamená si *identifikátor* a *sekvenční číslo* `ICMP` zprávy. Tyto údaje následně server použije při odesílání zpráv klientovi. Jedná se o techniku překonání překážek, které způsobuje `NAT`.

Situace, kdy se klient připojuje k volnému server je zachycena na diagramu 3.3.

Celková velikost paketu nesoucího zprávu je standardně nastavena na 1500 bajtů – jedná se o `MTU` technologie *Ethernet*. To zahrnuje `IP` hlavičku (20 bajtů), `ICMP` hlavičku (8 bajtů) a hlavičku zpráv tunelu (5 bajtů). Pro přenášená data je tedy k dispozici až 1467 zbylých bajtů v jedné zprávě. Režie přenosu dat tunelem tvoří 21 bajtů. Vytvářené pakety se drží standardní hodnoty `MTU` aby bylo (pokud možno) zamezeno fragmentaci. Diagram paketu se všemi hlavičkami je znázorněn na obrázku 3.4.



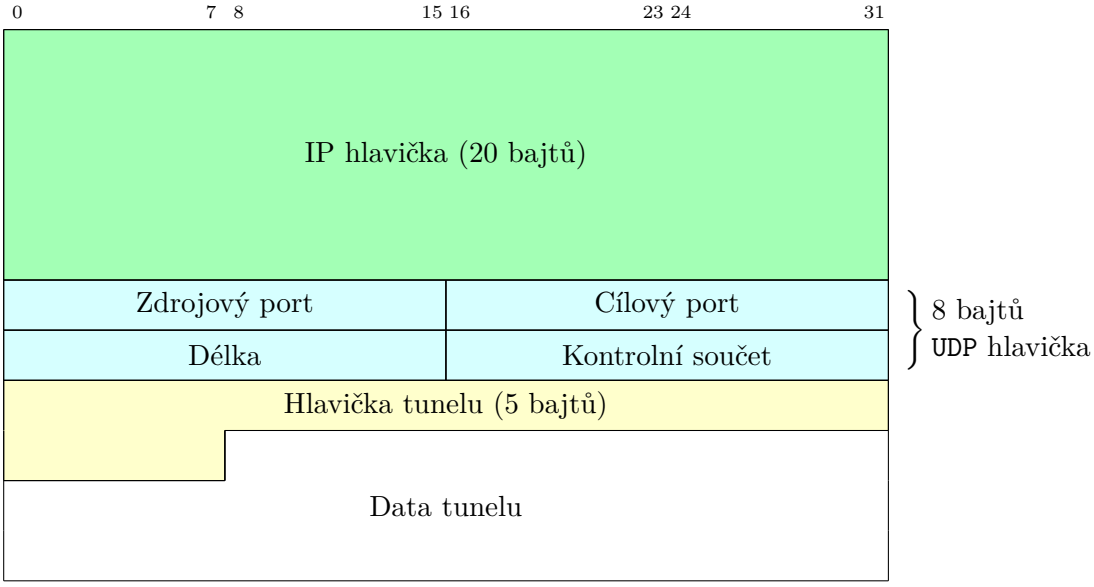
Obrázek 3.3: Sekvenční diagram úspěšného navázání ICMP tunelu



Obrázek 3.4: Diagram paketu včetně rozložení ICMP zprávy tunelu a hlaviček

3.3 Plugin pro UDP tunelování

Tunelování dat skrze UDP tunel je primárním principem technologie *OpenVPN*[8]. Jedná se o jednoduchý způsob tunelování dat sítí, kdy jsou data tunelu zapouzdřena do UDP datagramů. Protokol UDP nezajišťuje doručení zpráv v původním pořadí a neobsahuje ani mechanismy pro detekci ztracených nebo duplicitních zpráv. Tato zodpovědnost je ponechána na protokolech vyšších vrstev. Strukturu UDP datagramu znázorňuje diagram 3.5.



Obrázek 3.5: Diagram paketu obsahující UDP datagram s daty tunelu

Rozdílem oproti tunelování pomocí ICMP je volba portu pro komunikaci. Volbu portu závisí na předpokládané výjimce na firewallu *captive portálu*. Mezi běžné aplikace protokolu UDP patří zejména:

- *Domain Name System*, protokol pro překlad doménových jmen, port 53,
- *Network Time Protocol*, protokol pro synchronizaci času, port 123,
- *OpenVPN*, SW pro síťové tunely, port 1194,
- *Session Initiation Protocol*, protokol pro přenos signalizace, port 5060

Výše zmíněné služby jsou široce rozšířené a je tudíž možné, že pro ně na firewallu *captive portálu* bude existovat výjimka. Například IP telefony typicky nejsou schopny se ověřovat *captive portálu* a právě proto by pro takové zařízení mohla existovat výjimka ve firewallu. Má proto smysl provozovat UDP tunel

[8] *OpenVPN* samozřejmě podporuje i TCP tunelování

na takovém portu – tedy za účelem zvýšení šance nalezení bezpečnostních nedostatků firewallu *captive portálu*. Z výčtu jsem záměrně vynechal některé známé aplikace protokolu UDP, jako například DHCP – protože má silně lokální charakter a je velmi nepravděpodobné, že by neautentizovaným klientům bylo povoleno komunikovat s DHCP serverem napříč Internetem. Protokol DNS rovněž není vhodným kandidátem, protože pro DNS je v rámci této práce implementována sofistikovanější metoda tunelování dat, blíže popsaná v podkapitole 3.6 *Plugin pro DNS tunelování*.

Vhodnými kandidáty jsou tedy porty 123 (NTP) a 5060 (SIP). Službu *OpenVPN*, která do jisté míry plní podobný účel, lze pohodlně provozovat bok po boku softwarového řešení této diplomové práce. Není proto důvodu zbytečně okupovat výchozí port jiného software.

3.3.1 Popis jednotlivých typů UDP zpráv

Typ zprávy označuje jeden z typů, zachycených v ukázce kódu 3.3.1:

```
1  typedef enum UDP_PACKET_TYPE
2  {
3          UDP_CONNECTION_REQUEST,
4          UDP_AUTH_CHALLENGE,
5          UDP_AUTH_RESPONSE,
6          UDP_CONNECTION_ACCEPT,
7          UDP_CONNECTION_REJECT,
8          UDP_KEEPALIVE,
9          UDP_DATA
10 } UDP_PACKET_TYPE;
```

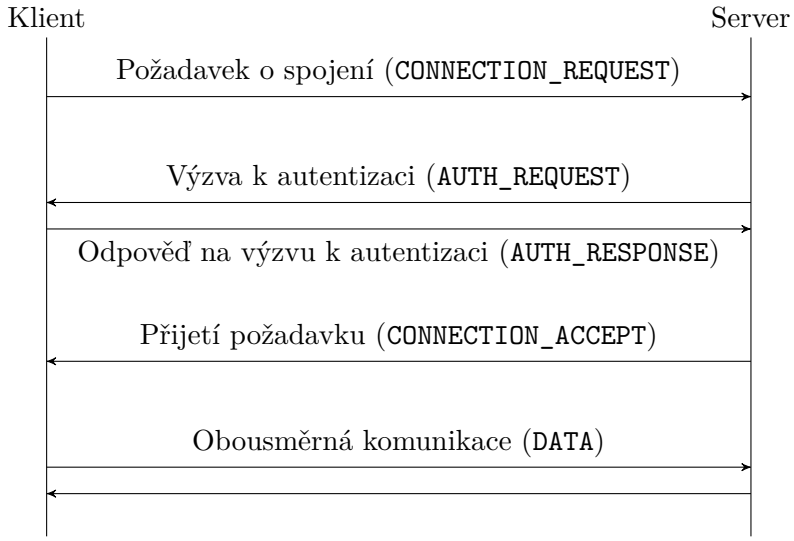
Ukázka kódu 3.3: Výňatek souboru `plugins/udp/packet.h` definující typy UDP zpráv

Z výňatku kódu je patrná analogie s výčtem zpráv 3.2.1 z podkapitoly *Plugin pro ICMP tunelování*. Jediným rozdílem oproti dříve popsaným typům zpráv je absence zpráv typu NATPACKET, protože protistrana (server) je schopen s klientem za NAT komunikovat díky překladu portů přímo. Klíčové jsou tím pádem KEEPALIVE zprávy, které slouží zejména k přesvědčení NAT zařízení o tom, že komunikace ještě neustala. Díky tomu může server komunikovat s klientem i skrze NAT.

Protože se však význam ostatních zpráv typu CONNECTION_REQUEST, AUTH_CHALLENGE, AUTH_RESPONSE,
CONNECTION_ACCEPT, CONNECTION_REJECT a DATA nijak nemění oproti specifikaci 2.5.1 *Popis základních servisních zpráv*, je jejich opětovný popis vynechán ve prospěch odkazu na původní specifikaci. Zprávy

Velmi podobný je také sekvenční diagram 3.6 komunikace klienta a serveru při pokusu o navázání spojení. V diagramu se nadále nevyskytují zprávy typu

NATPACKET.



Obrázek 3.6: Sekvenční diagram úspěšného navázání UDP/TCP tunelu

3.4 Plugin pro TCP tunelování

Protokol TCP je stěžejním členem transportní vrstvy ISO/OSI modelu. Poskytuje spolehlivý přenos dat, garantuje doručení dat ve stejném pořadí v jakém byly odeslány a detekci chyb při přenosu. Díky těmto vlastnostem je velice vhodný pro široké spektrum aplikací – na moderním Internetu se uživatelé nevědomky setkávají s TCP na každém kroku. Mezi nejrozšířenější aplikace TCP patří zejména protokol HTTP, protokoly pro přenos elektronické pošty (IMAP, POP3, SMTP) a další (SSH, FTP, telnet, ...).

Garance doručení, pořadí a detekce chyb s sebou přináší i některé negativní vlastnosti protokolu TCP. Pro aplikace, které vyžadují minimální latenci a rychlou komunikaci, není TCP vhodnou volbou. Zahájení TCP komunikace a případné opakované odesílání dat při ztrátě paketů nejsou ideální zejména pro aplikace v oblasti telefonie nebo živého přenosu obrazu a zvuku. Počítačové hry umožňující hru více hráčů po Internetu mohou rovněž vyžadovat co nejrychlejší provoz s minimální latencí a současně dokáží správně fungovat i při nedoručení části dat.

V takových případech je lepší zvolit jiný, jednodušší protokol transportní vrstvy (typicky UDP). Stejně k problému přistupují i autoři software *OpenVPN*, které primárně podporuje vytváření síťových tunelů pomocí UDP, ale je možné použít i TCP. To však není doporučeno a mělo by být používáno pouze pokud není možné spojení navázat pomocí UDP.

3.4.1 Potíže s TCP jako prostředkem pro tunelování

Jedním z důvodů nevhodnosti TCP jako technologie pro tunelování síťového provozu je fakt, že ne všechen tunelovaný provoz vyžaduje vlastnosti, které TCP nabízí. Implementace TCP obsahuje adaptivní mechanismy pro řízení zatížení linky tak, aby nedošlo k přetížení linky, popsané v [17, RFC 2001]. Klíčovým prvkem mechanismu je **exponenciální** zvyšování délky čekání před opakovaným vysláním nedoručeného segmentu dat. Takové chování však není žádoucí například při TCP komunikaci skrze TCP tunel. Pokud dojde ke ztrátě dat na TCP spojení tunelu, TCP dle implementace zvýší délku čekání před dalším opakovaným vysláním a připraví nedoručená data k opakovanému odeslání. Během této doby dochází k zablokování provozu tunelového spojení a TCP spojení aplikace komunikující skrze tunel nedostane včasné potvrzení protistrany – a rovněž dojde k zvýšení délky čekání před opakovaným vysláním data, která následně budou skrze tunel odeslána.

Pokud TCP spojení využívající tunel doposud nezvyšovalo délky vyčkávání, dojde k několika zvýšením a k zařazení několika opakovaných odeslání dat zatímco TCP spojení zajišťující tunel je stále zablokované. Každý pokus o opakované vyslání dat skrze tunel problém zhoršuje. Výsledkem je velmi snadno „zamrzající“ spojení napříč tunelem, zapříčiněné mechanismy garantující doručení zpráv a pořadí doručení. Tunelované spojení předpokládá standardní nestabilní linku bez garanci doručení – v případě TCP tunelu je však opak realitou. Tento důsledek tunelování TCP spojení pomocí TCP tunelu je znám jako *meltdown effect*[18].

3.4.2 Implementace TCP tunelu

Struktura tunelovacího nástroje popsaná v podkapitole 2.4 *Struktura softwarového řešení* umožňuje při implementaci *pluginů* pro jednotlivé tunelovací mechanismy využít již existující kód. Implementace *pluginu* pro TCP tunelování je proto velmi podobná implementaci *pluginu* pro UDP tunelování.

Rozdíly plynou především z rozdílného pojetí komunikace mezi UDP (samostatné zprávy) a TCP (orientace na samostatná spojení a jednotný proud dat). Oproti ICMP tunelování není klíčová identifikace komunikující protistrany, ale délka jednotlivých zpráv v proudu dat. I přes architekturu TCP založenou na konkrétním spojení (oproti UDP bezstavového soketu) zůstaly zachovány KEEPALIVE zprávy, které jsou užitečné zejména při navazování nového spojení – fáze autentizace. Protože z principu obcházení *captive portálu* je *plugin* navržen pro obsluhu pouze jednoho spojení, mohlo by docházet k blokování dostupného spojení jinými připojujícími se uzly v Internetu. Server tedy ignoruje KEEPALIVE zprávy až do chvíle, kdy se klient úspěšně autentizuje. Z pohledu serveru je klient před úspěšnou autentizací nedostupný, resp. již není na příjmu (ignorování KEEPALIVE zpráv) a bude odpojen, pokud se úspěšně neověří před vypršením maximální délky bez přijetí KEEPALIVE zprávy.

Sekvenční diagram 3.6 komunikace klienta a serveru při pokusu o navázání spojení je identický s pluginem pro UDP tunelování.

3.4.3 Volba portu pro provoz TCP tunelu

S přihlédnutím k nesmírné popularitě a rozšířenosti protokolu TCP se k maskování tunelového provozu nabízí celá řada TCP portů. Volbou může být port 80, resp 443 (HTTP, resp. HTTPS). S provozem cíleným na tyto porty však *captive portál* zpravidla manipuluje a nelze tak dopředu říci, jaká je šance na úspěšné obejití omezení *captive portálu*. Alternativní volbu představuje port 3128, který může mít ve firewallu výjimku za účelem provozu HTTP caching proxy (správce *hotspotu* může takovou službu provozovat za účelem šetření přenesených dat směrem z/do Internetu).

Možnou volbou je i port TCP/22, jehož užití s sebou však ponese nutnost změny portu SSH služby, který na serveru s velkou pravděpodobností je provozován.

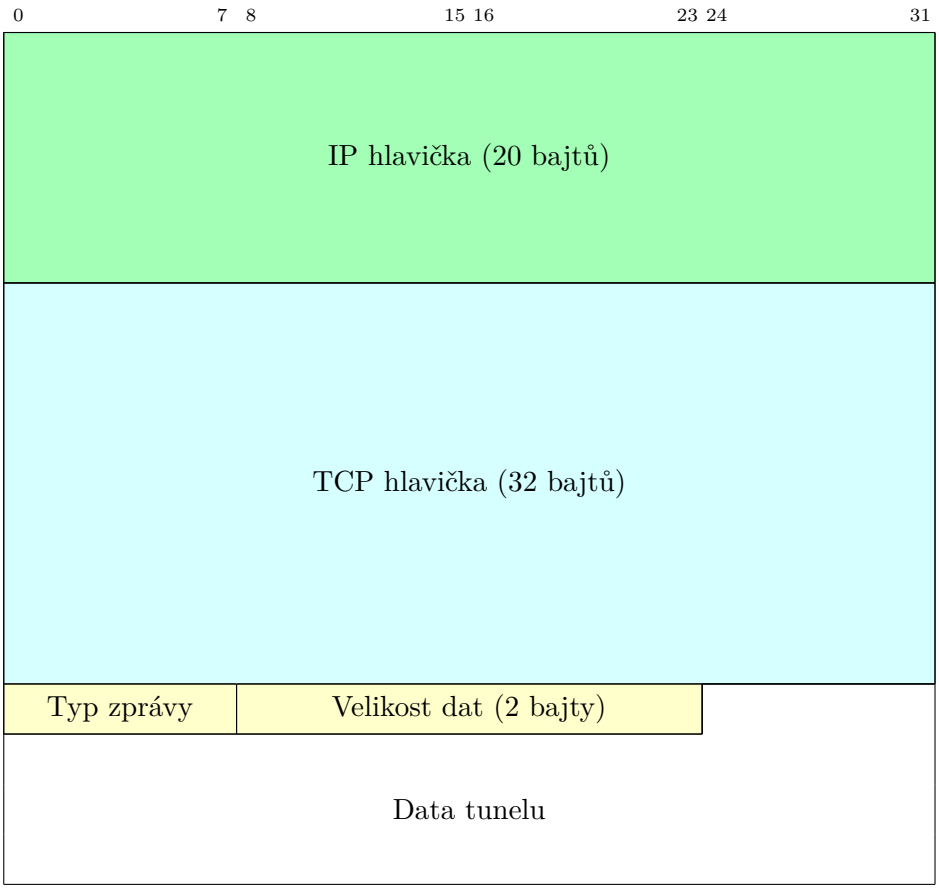
Není možné předem s jistotou říci, pro jaký (nebo zda-li vůbec nějaký) port má konkrétní *captive portál* ve firewallu nastavenou výjimku. Volba portu je uživatelsky nastavitelná a záleží tedy na zvážení uživatele software, kolik instancí TCP pluginu pro různé porty hodlá provozovat.

3.4.4 Návrh formátu datové zprávy

Z diagramu návrhu zprávy je dobře patrná výrazně větší režie TCP spojení (32 bajtů hlavičky oproti dosavadním 8 bajtům u ICMP a UDP). Protože TCP přistupuje k přenášeným datům jako k proudu dat, není zaručené, že jednotlivé zprávy budou mít vždy tento formát. Každá zpráva začíná jednobajtovým identifikátorem typu zprávy (servisní a její typ, nebo datová) a je následován velikostí zprávy (2 bajty). Velikost umožňuje specifikovat velikost až zprávy až 65535 bajtů, té by však nemělo být nikdy dosaženo, protože do tunelu data proudí z virtuálního síťového rozhraní, které má standardní MTU 1500 bajtů.

Velikost dat je přenášena ve dvou bajtech – nižších 8 bitů a vyšších 8 bitů samostatně pomocí maskování původní nezáporné šestnáctibajtové délky dat. Čtení dat z TCP proudu probíhá ve smyčce, která vždy:

1. Přečte jeden bajt (typ zprávy),
2. Přečte dva bajty a rekonstruuje velikost data,
3. Provede neblokující čtení dokud nepřečte přesně specifikované množství dat.



Obrázek 3.7: Diagram TCP zprávy s vyznačenou hlavičkou dat tunelu, délkou TCP hlavičky a délkou IP hlavičky

3.5 Plugin pro SCTP tunelování

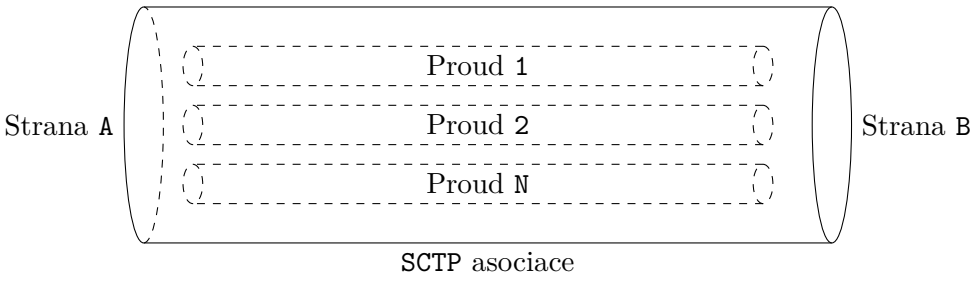
SCTP je protokolem transportní vrstvy ISO/OSI modelu, stejně jako protokoly TCP nebo UDP. Přebírá některé volitelné vlastnosti těchto protokolů, například volitelnou obdobu TCP garanci doručení paketů ve správném pořadí, nebo UDP způsob komunikace (oproti TCP proudové komunikaci). Navíc přidává funkce jako multihoming nebo víceproudovou komunikaci v rámci jednoho spojení. Jedná se o *relativně* mladý protokol, definovaný v [19, RFC 4960] v roce 2007.

Stejně jako u ICMP je možné, že implementace *captive portálu* nebude pamatovat na jiné protokoly než TCP a UDP. Právě proto je v rámce této práce implementován rovněž tunel pomocí SCTP protokolu. Jeho vlastnosti navíc umožňují efektivněji pracovat se spojením – zejména oproti TCP tunelu.

3.5.1 Víceproudovost

SCTP umožňuje v rámci jednoho spojení (*asociace*) komunikovat pomocí vícero nezávislých datových proudů. Tato vlastnost nachází praktické využití například v IP telefonii, kdy je možné v rámci jednoho spojení přenášet odděleně signalizaci a samotná hlasová data.

Pro účely této práce je tato vlastnost užitečná, protože kromě čistšího návrhu protokolu, který nemíchá servisní zprávy a data dohromady, tento přístup částečně řeší problém známý jako *head-of-line blocking*. Ten nastává zejména při zablokování datového proudu (například kvůli ztrátě paketu a čekání na opakované odeslání). Proud dat je tedy blokován první zprávou, která nebyla úspěšně doručena. Čekání na další postup brání prostupu jiných dat, zejména servisních zpráv. Oddělené datové proudy pro servisní zprávy a samotná data tunelu tento problém společně s 3.5.2 doručením v náhodném pořadí řeší problém *head-of-line blocking*.



Obrázek 3.8: Vztah proudů spojení a SCTP asociace

3.5.2 Doručení v náhodném pořadí

Protokol SCTP respektuje hranice zpráv. Díky tomu programátorovi nabízí možnost nevynucovat doručení zpráv ve stejném pořadí, v jakém byly ode-

slány. Tato vlastnost je pro implementaci tunelu velice vhodná, protože *nevnucuje* garanci pořadí doručení veškerému provozu a deleguje odpovědnost na protokoly vyšších vrstev.

3.5.3 Popis jednotlivých typů SCTP zpráv

Díky víceproudovosti transportního protokolu SCTP dochází ke kompletnímu oddělení dat tunelu a servisních zpráv pro řízení tunelu. Jednotlivé servisní zprávy jsou zachyceny v ukázce kódu 3.5.3:

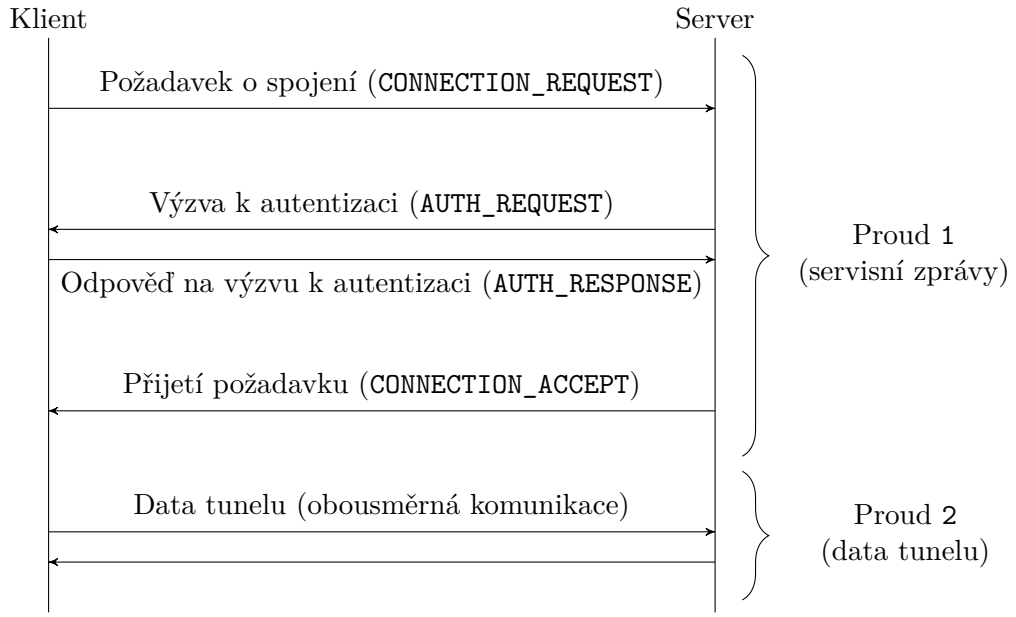
```
1 typedef enum SCTP_PACKET_TYPE
2 {
3         SCTP_CONNECTION_REQUEST,
4         SCTP_AUTH_CHALLENGE,
5         SCTP_AUTH_RESPONSE,
6         SCTP_CONNECTION_ACCEPT,
7         SCTP_CONNECTION_REJECT,
8         SCTP_KEEPALIVE
9 } SCTP_PACKET_TYPE;
```

Ukázka kódu 3.4: Výňatek souboru `plugins/sctp/packet.h` definující typy SCTP zpráv

Počet typů servisních zpráv je oproti ostatním implementovaným tunelům omezen na nejnutnější minimum. Jsou implementovány přesně dle specifikace popsané v podkapitole 2.5.1 *Popis základních servisních zpráv*. Protože se však význam ostatních zpráv nijak nemění, je jejich opětovný popis vynechán s odkazem na jejich specifikaci v podkapitole 2.5.1. Lehce odlišný je však sekvenční diagram 3.6 komunikace klienta a serveru při pokusu o navázání spojení. V diagramu se nadále nevyskytují zprávy typu DATA, protože data tunelu jsou přenášena samostatným datovým proudem.

3.5.4 Volba portu pro provoz SCTP tunelu

Pointou implementace SCTP pluginu je možná nedbalost správce *firewallu*, či nedokonalost *captive portálu*, která neomezuje provoz protokolu SCTP. Uživatel tedy není limitován snahou o maskování provozu tunelu jako jiné legitimní či neblokované činnosti. Nevelká rozšířenost protokolu SCTP však umožňuje obsadit známé TCP porty na SCTP soketech aniž by došlo k narušení jiných bežících služeb. Opět platí, že konkrétní volbu nelze s jistotou předem předpovědět a je tak na zvážení uživatele, jaké výjimky ve firewallu by mohlo daná síť mít.

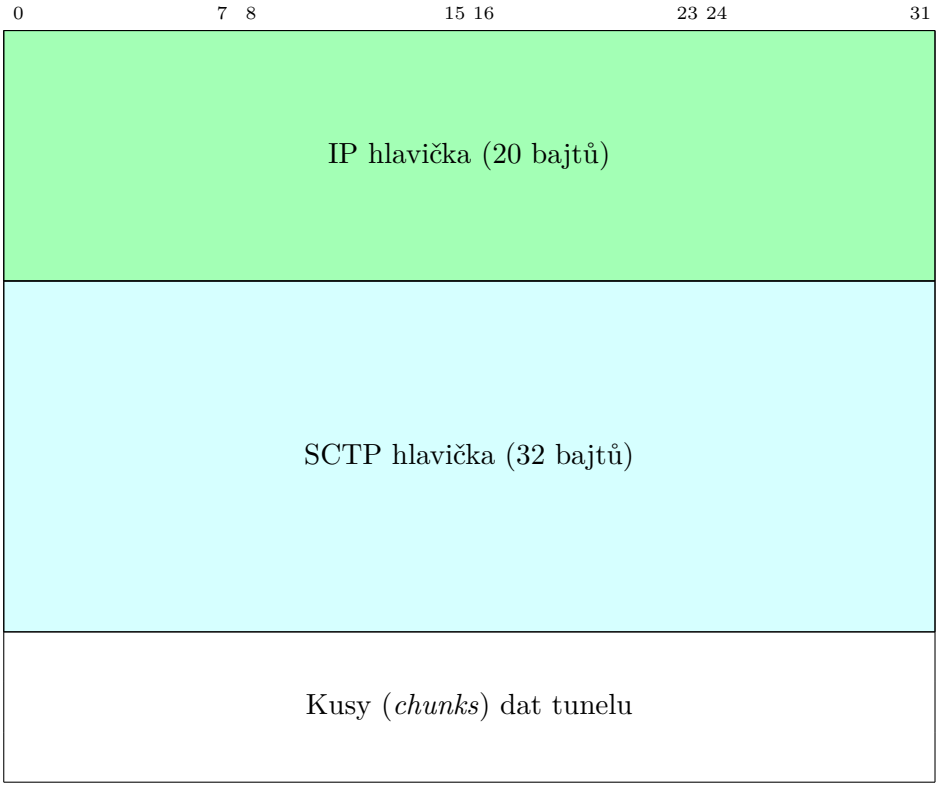


Obrázek 3.9: Sekvenční diagram úspěšného navázání SCTP tunelu

3.5.5 Návrh formátu datové zprávy

Protokol SCTP disponuje hlavičkou o velikosti 32 bajtů, velikostně menší, než TCP (38 bajtů), ale větší než ICMP a UDP (8 bajtů). Zato však disponuje možností odeslání několika zpráv (*chunks*) v jednom paketu (což zahrnuje i IP hlavičku) při zachování hranic zpráv. Protože SCTP umožňuje komunikovat pomocí oddělených a nezávislých datových proudů, není pro datové zprávy nutné specifikovat hlavičku dat tunelu, protože všechny servisní zprávy používají vlastní proud. „Střední“ velikost hlavičky je tedy vyvážena výhodami protokolu SCTP.

Protistrana tak může data rovnou zapisovat do virtuálního rozhraní a nemusí se zabývat čekáním na více zpráv a jejich následnému skládání (za předpokladu, že díky servisním zprávám již proběhla autentizace).



Obrázek 3.10: Diagram SCTP zprávy nezávislého datového proudu (tedy bez redundantní hlavičky dat tunelu) a s vyznačenou délkou SCTP hlavičky a délkou IP hlavičky

3.6 Plugin pro DNS tunelování

Protokol DNS je jedním z hlavních pilířů infrastruktury moderního Internetu. Umožňuje překládat doménová jména na IP adresy a poskytovat další související informace o konkrétní doméně, například pro poštovní servery.

Tunelování pomocí DNS staví na odlišném principu, než doposud všechny navržené a implementované pluginy. Největším rozdílem je, že oproti TCP, UDP a SCTP tunelům nespoléhá na navázání spojení se serverem v Internetu, ale podobně jako ICMP se snaží přimět okolní infrastrukturu, aby zprávy předala k cílovému serveru v Internetu a odpověď dopravila zpět. Základním mechanismem DNS tunelování jsou DNS servery, které umí zpracovat rekurzivní DNS požadavky.

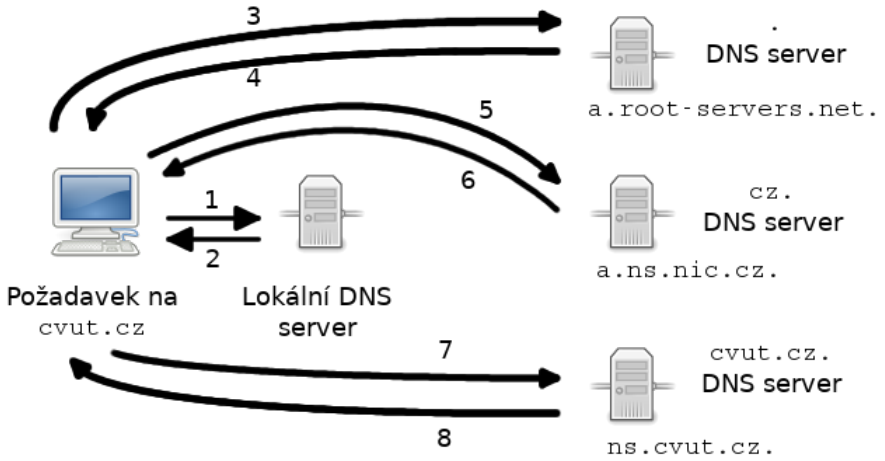
3.6.1 Principy DNS tunelování

Základním principem služby DNS je hierarchická struktura *jmenných serverů*[9], které jsou schopny odpovídat na DNS dotazy od klientských překladačů (*resolver*). Pokud jmenný server nezná odpověď na dotaz, může

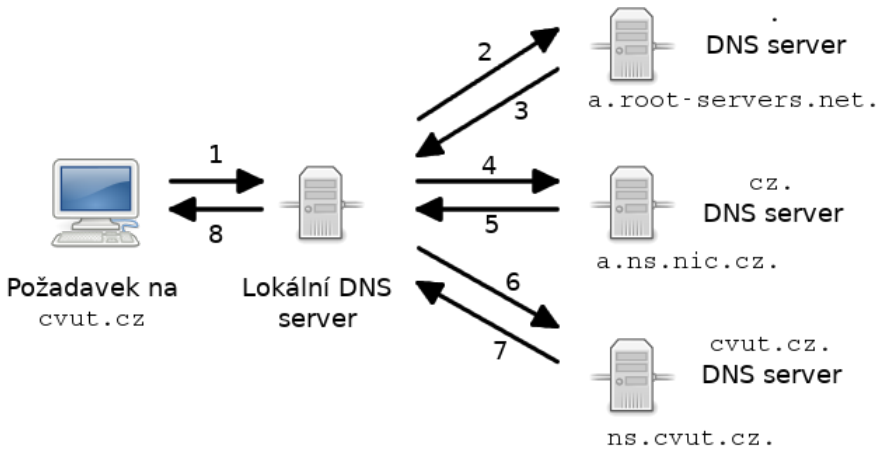
- klienta odkázat na jiný jmenný server (nerekurzivní chování),
- odpověď sám dohledat dotazem na další jmenné servery a vrátit klientovi (rekurzivní chování).

Pokud jmenný server je ochoten rekurzivně vyhodnocovat dotazy, může klient tento server přimět ke komunikaci s okolním světem. Klient může v tomto případě přesně *nasměrovat* rekurzivní DNS server do části Internetu, kterou klient ovládá (například provozuje svůj vlastní DNS server). V této pozici je tedy klient schopen komunikovat prostřednictvím rekurzivního DNS serveru s jiným svým systémem, dostupným přes síť Internet. Protože klient ovládá cílový DNS server, má tím pádem kontrolu i nad daty, která budou rekurzivním DNS serverem klientovi vrácena jako odpověď na jeho požadavek. Výsledkem je možnost obousměrné komunikace do Internetu pouze prostřednictvím dostupného DNS serveru ochotného zpracovávat rekurzivní dotazy.

[9] Pro potřeby této práce je v textu rovněž použito označení „DNS server“



Obrázek 3.11: Ukázka iterativního (nerekurzivního) DNS dotazu. Klient se DNS serverů dotazuje sám. Buď dostane odpověď, nebo je odkázán na jiný DNS server.



Obrázek 3.12: Ukázka rekurzivního DNS dotazu. Lokální DNS server prohledá svou *cache* a pokud v ní nenalezne odpověď na klientův dotaz, začne odpověď hledat dotazy na jiné DNS servery.

3.6.2 Překážky DNS tunelování

Protože ústřední součástí DNS tunelu je komunikace skrze prostředníka (rekurzivní DNS server), je pro DNS tunelování nutné zachovat specifikace DNS protokolu dle [20, RFC 1035] – například oproti pouhému provozu UDP (resp TCP) pluginu na portu UDP/53 (resp. TCP/53). Z toho plynou nutná omezení pro tunelování síťového provozu – nemožnost *plně* oboustranné komunikace, protože klientem ovládaný DNS server v Internetu nedokáže dopravit data ke klientovi, pokud nemá klientův DNS dotaz k zodpovězení.

Tuto překážku lze vyřešit metodou *polling*, kdy se klient opakovaně dotazuje serveru pro případ, že server má data, která je třeba dopravit ke klientovi (jako součást odpovědi). Toto řešení s sebou však nese výrazně sníženou prostupnost tunelového spojení. Klient typicky nemůže DNS serveru pokládat desítky – nebo dokonce stovky – dotazů za vteřinu. Mnohé DNS servery používají mechanismy pro limitování počtu požadavků jednoho klienta za určitý čas, jak jsem zjistil ve své bakalářské práci [21] – například DNSSEC servery českého sdružení *CZ.NIC, z.s.p.o.*[22] limitují dotazy na přibližně 10 dotazů za vteřinu.

Dalším problémem je *caching* – ukládání výsledků po specifikovanou dobu s cílem zrychlení odpovědí na dotazy a menší zátěž ostatních DNS serverů, spojená s nižším síťovým provozem. Tato, leč šlechetná, vlastnost je krajně nežádoucí pro tunelování dat pomocí DNS, protože odpovědi typicky nikdy nebudou stejné. Klient se tím pádem musí serveru pokaždé dotazovat na unikátní doménové jméno. Součástí DNS odpovědi sice může být atribut TTL nastaven na hodnotu 0 (tedy efektivně zakázání uložení do *cache*), nicméně nelze zaručit, že DNS server *captive portálu* bude atribut respektovat.

Limitována je i celková velikost požadavku/odpovědi. V původním návrhu protokolu [20, RFC 1035] je uvedena maximální velikost UDP zprávy 512 bajtů. Příčinou této limitace je snížení pravděpodobnosti fragmentace paketu po cestě na server a ze serveru zpět. Specifikace IPv4 [23, RFC 791] vyžaduje, aby každý účastník sítě byl schopen přijmout paket minimálně o velikosti 576 bajtů. Jedná se však o více než 30 let staré specifikace a schopnosti zařízení připojených k Internetu typicky nemají problém s velikostí přijímaných paketů větší, než 576 bajtů. Větší data (například pro přenos celé zóny) je nutné přenášet pomocí TCP.

Právě proto vzniklo v roce 1999 rozšíření *EDNS* specifikované v [24, RFC 6891], které mimo jiné umožňuje užití větší velikost UDP paketů – až 4096 bajtů. Díky rozsáhlé podpoře (demonstrace v ukázce kódu 3.6.2 je možné větší povolenou velikost paketů využít k lepší prostupnosti tunelu.

3.6.3 Detekce a prevence DNS tunelování

DNS tunelování generuje značně atypický síťový provoz, který je relativně snadné detekovat[12]. Míra atypičnosti závisí na „střídmosti“ softwarového

```
1  $ dig fit.cvut.cz @1.1.1.1
2  ;; Got answer:
3  ;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
4
5  ;; OPT PSEUDOSECTION:
6  ; EDNS: version: 0, flags:; udp: 1536
7  ;; QUESTION SECTION:
8  ;fit.cvut.cz.                IN      A
9
10 ;; ANSWER SECTION:
11 fit.cvut.cz.          797     IN      A     147.32.232.248
```

Ukázka kódu 3.5: Část výstupu nástroje `dig` demonstrující užití `EDNS` v praxi (velikost `UDP` paketu 1536 bajtů od DNS serveru `1.1.1.1` (provozovatel *Cloudflare* a *APNIC*)

řešení, tunel s nižší prostupností a vyšší latencí lze maskovat lépe, než tunel s maximální prostupností a minimální latencí. Na Internetu lze dohledat články, které správcům radí zahazovat na portu 53[10] `UDP` data větší, než 512 bajtů a `TCP` data větší než 1024 bajtů[25].

3.6.4 Kódování dat tunelu

Na rozdíl od ostatních implementovaných pluginů pro tunelování dat, `DNS` tunelování vyžaduje *transformaci* přenášených dat. Specifikace `DNS` [20, RFC 1035] totiž v části 3.5 jasně vymezuje maximální délku 63 znaků subdomény, maximálně 255 znaků celkem a povolené znaky doménových jmen – alfanumerické znaky (a-z, A-Z a 0-9) a pomlčka. V případě odpovědi je však k dispozici typ odpovědi `NULL`, který dle článku 3.3.10 specifikace [20, RFC 1035] smí obsahovat jakákoliv data s maximální délkou 65535 bajtů. Tento typ `DNS` záznamu však není příliš rozšířený a jeho nadměrné používání, poukazující na možné tunelové spojení, lze snadno strojově detekovat a blokovat.

Poznámka k `IDN` Specifikace povolených znaků v doménových jménech je poměrně striktní. Mohlo by se zdát, že některá platná doménová jména, jako například `háčkyčárky.cz` se specifikaci vymykají, nicméně taková doménová jsou (skrytě před uživatelem) fakticky překládána tak, aby původní specifikaci vyhovovaly. Zmíněná doména `háčkyčárky.cz` je tak fakticky doménou `xn-hkyrky-ptac70bc.cz` a nenabízí tudíž prostor pro efektivnější kódování dat do doménových jmen.

[10] paradoxně odkazovaná `iptables` pravidla filtrují provoz na portu 53 pouze pro protokoly `TCP` a `UDP` – `SCTP` tunel by na základě odkazovaných pravidel firewallem prošel.

3.6.4.1 Kódování dat v názvu domény Base32

I přesto, že specifikace povoluje jak malé, tak velké alfanumerické znaky, fakticky dochází k ignorování velikosti písmen, viz experiment 3.6.4.1. Fakticky je tedy pro každou část doménového jména k dispozici 26 znaků, 10 číslic a pomlčka[11], dohromady 37 různých znaků.

Kódování *Base32* je schopno v jednom znaku kódu uchovat 5 bitů informace původních dat. Vyžaduje k tomu alespoň 2^5 znaků abecedy, což povolený rozsah znaků splňuje.

Například, pokud pro přenesení zprávy *All your base are belong to us* na `DNS` server je možné tuto zprávu zakódovat do podoby
`ifwgyidzn52xeidcmfzwkidbojssaytfnrxw4zzaorxsa5ltbi.example.com`
přičemž subdoména je složena z tajné zprávy. Touto cestou lze přenášet informace ze sítě omezené *captive portálem*.

```
1 $ dig +short fit.cvut.cz A @8.8.8.8
2 147.32.232.248
3
4 $ dig +short fIT.CvUt.cZ A @8.8.8.8
5 147.32.232.248
```

Ukázka kódu 3.6: Demonstrace ignorování velikosti písmen v doménových jménech

3.6.4.2 Porušení specifikace omezující znaky domény

Na konferenci *Shakacon*[26] v roce 2009 prezentovali autoři `DNS` tunelovacího nástroje *Heyoka* své nálezy z oblasti podpory doménových jmen porušujících specifikace napříč různými `DNS` servery. Autoři zjistili, že některé `DNS` servery jsou schopny zpracovat i doménová jména, která obsahují nejen `ASCII` znaky mimo specifikace, ale rovněž netisknutelné znaky. Díky tomu byli schopni za jistých podmínek dosáhnout výrazně lepších vlastností tunelového spojení. Nedodržením specifikace se však autoři vystavují riziku komunikace prostřednictvím `DNS` serveru, který nebude `DNS` dotazům rozumět a tunelové spojení se nepodaří navázat. Vytvořené softwarové řešení je autory označeno jako *Proof of concept*, tedy jako *demonstrace principů*.

3.6.4.3 Přenos dat v TXT záznamu

Pro přenos dat z `DNS` serveru zpět ke klientovi lze využít `TXT` záznamy. Takový záznam může obsahovat řetězec délky až 255 bajtů. Je však přípustné, aby `TXT` záznam obsahoval takový řetězců více a tím pádem je možné pomocí `TXT` záznamu přenést v rámci jedné odpovědi více, než 255 bajtů. Odkazovaná

[11] Žádná část doménového jména nesmí začínat ani končit pomlčkou

aplikace *Heyoka*[26] pro zvýšení prostupnosti rovněž experimentuje s binárními daty v rámci TXT záznamů, což ovšem opět vnáší problém kompatibility DNS serverů zpracovávajících DNS odpověď.

Pro zachování vymezené sady znaků dle *RFC* je možné opět použít kódování Base32, schopné do jednoho znaku odpovědi uložit 5 bitů informace. TXT záznamy však dbají na velikost písmen a díky tomu je pro kódování k dispozici abeceda složená z malých a velkých znaků abecedy (52 znaků), číslice (10 znaků) a některé speciální znaky, jako například znak + (plus), nebo - (minus). Tímto způsobem lze dosáhnout abecedy o 64 různých znacích. Proto je možné k zakódování informace do TXT záznamu možné zvolit kódování Base64, které do jednoho znaku odpovědi zakóduje 6 bitů informace.

Například, pokud pro přenesení zprávy *All your base are belong to us* z DNS serveru jako odpověď v rámci záznamu TXT je možné tuto zprávu zakódovat do podoby QWxsIHlvdXIgYmFzZSBhcmUgYmVsb25nIHRvIHVz, která obsahuje pouze 40 znaků, namísto 50 v případě užití Base32.

3.6.4.4 Použití jiných typů DNS záznamů

Mimo záznam typu TXT a experimentální typ NULL, popsaný v podkapitole 3.6.4*Kódování dat tunelu*, je možné data ze serveru ke klientovi přenášet i pomocí jiných typů záznamů. Jedná se o běžněji používané DNS záznamy, na které se však váží různá omezení a nejsou proto ideálními kandidáty pro přenos dat. Jedná se zejména o typy A, AAAA, CNAME, NS a MX. Výhodou užití těchto DNS záznamů je nižší riziko detekce tunelu.

3.6.5 Implementace pluginu pro DNS tunelování

Vzhledem k rozsáhlé nabídce dostupných nástrojů pro tunelování DNS – od nástrojů pro tunelování po nástroje pro tichou exfiltraci dat – není v rámci této diplomové práce implementován plugin pro DNS tunelování. Namísto toho jsou vybrané dostupné nástroje analyzovány a srovnány s ostatními technikami obcházení *captive portálu*.

Kapitola 4

Testování

Tato kapitola je věnována testování implementovaných pluginů, srovnání s jiným dostupným software a porovnání vlastností vytvořených síťových tunelů.

4.1 Základní testovací prostředí

Jako klientská část pro testování implementovaného software a ostatních srovnávaných programů byl použit virtualizovaný operační systém *Ubuntu 17.10 x64* serverové verze s 8GB paměti, 4 jádry procesoru. Přístup do sítě byl záměrně omezen na hodnotu 10 Mbit/s, aby byl minimalizován efekt „úzkého hrdla“ v částech síťové infrastruktury, které nelze z pozice uživatele ovlivnit. Distribuce *Ubuntu* byla zvolena pro široký výběr a snadnou instalaci systémových balíčků, nutných k sestavení a spuštění softwarového řešení. Serverová verze distribuce *Ubuntu* byla zvolena pro minimalizaci podílu částí operačního systému na síťovém provozu (například systém automatických aktualizací, nebo jiná aktivita vyvinutá uživatelem).

4.2 Testování reálného provozu

Testování skrze reálnou síť bylo provedeno s pomocí kontejnerové služby provozované společností *DigitalOcean*. S pomocí této služby byla vytvořena serverová strana tunelu, umístěná ve Frankfurtském datacentru společnosti *DigitalOcean*. Jako testovací kontejner byl zvolen serverový obraz Linuxové distribuce *Ubuntu*, konkrétně *Ubuntu 16.04.4 x64*. Kontejner měl k dispozici dedikované 4GB pamětí a 2 jádra *vCPU*. Kontejner disponoval připojením do gigabitové sítě.

4.2.1 Příprava kontejneru pro testování

Pro sestavení softwarového řešení ze zdrojových kódů bylo třeba ve vytvořeném kontejneru doinstalovat některé závislosti projektu. Jednalo se o balíčky:

- `build-essential` – základní nástroje pro překlad zdrojových kódů
- `libssl-dev` – vývojová verze knihovny *OpenSSL*
- `libsctp-dev` – vývojové prostředí pro protokol `SCTP`
- `iperf` – nástroj pro testování síťové prostupnosti

4.2.2 Testování simulovaného stahování souboru pomocí `HTTP`

Tento test se skládal z testování simulace činnosti reálného uživatele – stahování dat pomocí protokolu `HTTP`, protože se jedná o jeden z nejtypičtějších scénářů uživatele Internetu. Za tímto účelem byl na serverové straně tunelu vytvořen binární soubor o velikosti 16MB s pomocí příkazu zachyceného v ukázce kódu 4.1.

```
1 $ dd if=/dev/random of=random.16 bs=1M count=16
2 16+0 records in
3 16+0 records out
4 16777216 bytes (17 MB, 16 MiB) copied, 0,0254478 s, 659 MB/s
```

Ukázka kódu 4.1: Vytvoření souboru binárních dat za účelem testování přenosu pomocí `HTTP`

K testování (stahování dat) byl použit příkaz `wget`, dostupný v distribuci *Ubuntu* pomocí stejnojmenného balíčku. Otestovány byly samostatně všechny implementované pluginy pro tunelování. Všechny uvedené hodnoty jsou průměrem deseti nezávislých měření. Jako kontrolní měření bylo provedeno stažení dat bez využití tunelu. Ukázka výstupu nástroje `wget` je zachycena v ukázce kódu 4.2. Jako `HTTP` server byl použit modul jazyka *Python* s názvem *SimpleHTTPServer*, spuštěn pomocí příkazu `python -m SimpleHTTPServer`. Server neprováděl žádnou kompresi přenášených dat.

Plugin	Délka přenosu	Průměrná rychlost přenosu
`ICMP`	15,2 vteřin	1,05 MB/s
`TCP`	91,8 vteřin	179,2 KB/s
`UDP`	15,1 vteřin	1,06 MB/s
`SCTP`	26,7 vteřin	631 KB/s
(žádný)	14,4 vteřin	1,12 MB/s

Tabulka 4.1: Tabulka zachycující naměřenou dobu přenosu dat pomocí protokolu `HTTP` s využitím různých tunelů

Z tabulky 4.1 plyne, že ideální tunelové spojení pro přenos dat pomocí `HTTP` je tunel na bázi `UDP` nebo `ICMP`. Naměřené hodnoty reflektují pluginu pro `TCP` tunelování reflektují problém popsaný v podkapitole 3.4.1 *Potíže s `TCP`*

jako prostředkem pro tunelování. Protokol HTTP pro komunikaci totiž využívá TCP, což v kombinaci s TCP tunelem neposkytuje optimální výkon. Délka přenosu dat s využitím SCTP tunelu se nachází ve středu spektra mezi nejvíce a nejméně vhodnými technikami tunelování pro HTTP přenosy. To je způsobeno relaxováním podmínek přenosu – implementovaný plugin pro datový proud nevyžaduje dodržení seřazení zpráv, ale (z principu SCTP) vyžaduje potvrzení doručení zpráv – oproti rychlejším UDP a ICMP tunelům. Protože stahovaná data jsou přenášena protokolem, který vynucuje potvrzování doručení zpráva zachování pořadí, dochází k propadu prostupnosti. Ta je mimo jiné způsobena TCP mechanismy pro řízení zahlcení linky, které pro správné dynamické přizpůsobení *potřebují* ztrátovost paketů, která však v tunelech pomocí SCTP a TCP nenastává.

```
--2018-04-06 17:13:35-- http://192.168.99.2:8000/random.16
Connecting to 192.168.99.2:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 16777216 (16M) [application/octet-stream]
Saving to: "/dev/null"

     0K ......... ......... ......... ......... ......... 0% 1,01K 17s
    50K ......... ......... ......... ......... ......... 0% 1,07M 16s
   100K ......... ......... ......... ......... ......... 0% 1,04M 15s
   150K ......... ......... ......... ......... ......... 1% 1,07M 15s
   200K ......... ......... ......... ......... ......... 1% 1,07M 15s
   250K ......... ......... ......... ......... ......... 1% 1,04M 15s

[...]

 16200K ......... ......... ......... ......... ......... 99% 1,07M 0s
 16250K ......... ......... ......... ......... ......... 99% 1,04M 0s
 16300K ......... ......... ......... ......... ......... 99% 1,07M 0s
 16350K ......... ......... ......... ....                 100% 1,04M=15s

2018-04-06 17:13:50 (1,05 MB/s) - '/dev/null' saved [16777216/16777216]
```

Ukázka kódu 4.2: Ukázkový výstup příkazu `wget`

4.2.3 Testování provozu pomocí nástroje `iperf3`

Nástroj *iperf* slouží k testování síťové prostupnosti. K testování byla použita nejnovější stabilní verze `iperf 3.1.3`. Délka testování byla ponechána na výchozí hodnotě 10 vteřin. Testování proběhlo pomocí protokolu TCP.

Nástroj `iperf3` funguje na základě zahlcení sítě daty pro přenos. Na velký nátlak dat nejlépe reaguje tunel pomocí UDP. V porovnání s testem stahování dat pomocí HTTP si v `iperf3` testu vede lépe implementace SCTP tunelu, kdy je – dle záznamů síťového provozu aplikací *Wireshark* – efektivněji využíván

princip *kousků* (anglicky *chunks*) dat, sdílející hlavičku paketu. Naopak horší vlastnosti při zahlcení vykazuje `ICMP` tunel, kdy při odesílání `ICMP` zpráv soket začal hlásit chyby typu *No buffer space available*.

Plugin	Průměrná rychlost přenosu
`ICMP`	959,5 KB/s
`TCP`	29,2 KB/s
`UDP`	1,01 MB/s
`SCTP`	851 KB/s
(žádný)	1,12 MB/s

Tabulka 4.2: Tabulka zachycující naměřenou dobu přenosu dat pomocí nástroje `iperf3` s využitím různých tunelů

```
  iperf3 -c 192.168.99.2
Connecting to host 192.168.99.2, port 5201
[  4] local 192.168.99.1 port 39950 connected to 192.168.99.2 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  1.23 MBytes 10.3 Mbits/sec  0   77.8 KBytes
[  4]   1.00-2.00   sec  1.12 MBytes 9.38 Mbits/sec  0    126 KBytes
[  4]   2.00-3.00   sec  1.37 MBytes 11.5 Mbits/sec  0    180 KBytes
[  4]   3.00-4.00   sec  1.37 MBytes 11.5 Mbits/sec  0    233 KBytes
[  4]   4.00-5.00   sec  1.37 MBytes 11.5 Mbits/sec  0    288 KBytes
[  4]   5.00-6.00   sec  1.30 MBytes 10.9 Mbits/sec  0    342 KBytes
[  4]   6.00-7.00   sec  1.37 MBytes 11.5 Mbits/sec  0    396 KBytes
[  4]   7.00-8.00   sec  1.37 MBytes 11.5 Mbits/sec  0    451 KBytes
[  4]   8.00-9.00   sec  1.37 MBytes 11.5 Mbits/sec  0    505 KBytes
[  4]   9.00-10.00  sec  1.30 MBytes 10.9 Mbits/sec  0    559 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  13.2 MBytes 11.0 Mbits/sec  0             sender
[  4]   0.00-10.00  sec  12.7 MBytes 10.67 Mbits/sec                receiver

iperf Done.
```

Ukázka kódu 4.3: Ukázkový výstup nástroje `iperf3`

4.2.4 Srovnání s dostupnými nástroji pro `DNS` tunelování

Jedním z nejpopulárnějších a nejaktivněji vyvíjených projektů pro tunelování pomocí `DNS` je projekt *iodine*. Dle oficiálního repozitáře zdrojových kódů[27] do projektu přispělo několik desítek vývojářů. Výsledné softwarové řešení podporuje řadu automatizovaných mechanizmů pro detekci optimálních parametrů síťového tunelu. Implementace rovněž využívá virtuálních síťových rozhraní typu *TUN*. Nástroj je schopen detekovat možnost `UDP` tunelování skrze port

53, kterou při spuštění ve výchozí podobě otestuje jako první. Pokud dostupná není, pokusí se využít některý z typů DNS odpovědí v pořadí od NULL, přes TXT, SRV, MX, CNAME a A. Pořadí je dáno známou prostupností vybraného typu záznamu[27]. Software automaticky rozpozná maximální přípustnou délku požadavků a aplikuje ji (pokud uživatel explicitně nespecifikuje jinak). Různé způsoby kódování dat do DNS provozu autoři označují jako *kodeky*.

4.2.4.1 Testování software ve výchozím nastavení na síti bez omezení

Prvním krokem pro otestování software je stejná sada testů jako pro softwarové řešení vyvinuté pro účely této diplomové práce. Testovací prostředí je popsáno v kapitole 4.1 *Základní testovací prostředí*.

V případě DNS tunelování je však důležité brát ohled na asymetričnost spojení. Zatímco pro tok dat od serveru ke klientovi existuje více způsobů, pro tok dat směrem od klienta na server tomu tak není. Právě proto se *iodine* snaží nejprve navázat spojení pomocí UDP tunelu na portu používaném službou DNS. Následující tabulka 4.3 a tabulka 4.4 zachycují naměřené časové hodnoty pro tunel pomocí UDP pluginu a software *iodine*. Metodika měření je stejná jako v podkapitole 4.2.2 *Testování simulovaného stahování souboru pomocí HTTP* a 4.2.3 *Testování provozu pomocí nástroje iperf3*.

Pro testování byla nutná úprava DNS záznamů veřejně dostupné domény. Za tímto účelem jsem použil svou doménu cernac.cz, respektive nově vzniklou subdoménu t1.cernac.cz a její nameserver, zajištěný softwarem *iodine* na adrese t1ns.cernac.cz. Software *iodine* byl spuštěn s výchozími parametry dle uživatelské příručky[27].

Nástroj	Délka přenosu	Průměrná rychlost přenosu
UDP plugin	15,1 vteřin	1,06 MB/s
nástroj *iodine*	15,2 vteřin	1,05 MB/s

Tabulka 4.3: Tabulka zachycující naměřenou dobu přenosu dat pomocí protokolu HTTP s využitím implementovaného UDP tunelu a stejné funkcionality software *iodine*

Plugin	Průměrná rychlost přenosu
UDP	1,01 MB/s
nástroj *iodine*	1,01 MB/s

Tabulka 4.4: Tabulka zachycující naměřenou dobu přenosu dat pomocí nástroje iperf3 s využitím implementovaného UDP tunelu a stejné funkcionality software *iodine*

Z tabulek vyplývá, že obě implementace UDP tunelování dosahují stejných výsledků z hlediska měření prostupnosti. Jedná se opět o symetrické spojení, proto jsou uvedeny údaje pouze v jednom směru spojení, testovány však byly oba směry – se stejnými výsledky.

4.2.4.2 Testování software s vynuceným použitím DNS

Zajímavější je testování software *iodine* na síti, které nepovoluje přímý UDP tunel k cílovému serveru s protějškem tunelu. Toto chování bylo simulováno pomocí přepínače -r, který software přiměje ignorovat možnost přímého UDP spojení. Dle oficiální stránky projektu je neoptimálnějšího spojení dosaženo s užitím záznamů typu NULL a kodeku Raw, případně Base128. Pro účely testování bylo navázáno spojení s těmito parametry a aplikovány stejné testovací metody. Protože se jedná o asymetrické spojení (prostupnost jednoho směru spojení se liší od opačného směru), jsou uvedeny údaje pro každý směr zvlášť.

Směr	Délka přenosu	Průměrná rychlost přenosu
Stahování ze serveru	544,3 vteřin	36,1 KB/s
Nahrávání na server	577,6 vteřin	23,7 KB/s

Tabulka 4.5: Tabulka zachycující naměřenou dobu přenosu dat pomocí protokolu HTTP v obou směrech využitím software *iodine* – DNS záznamy typu NULL

Z tabulky 4.5 lze vyčíst, že rychlosti přenosu v ideálních podmínkách nejsou srovnatelné s jinými analyzovanými technikami tunelování. Pro srovnání lze uvést fakt, že průměrné YouTube video v nejnižší dostupné kvalitě 144p vyžaduje prostupnost zhruba 32KB/s[28]. Oficiální webové stránky fakulty, umístěné na doméně fit.cvut.cz, dosahují po sečtení všech závislostí velikosti 2,9MB. Načtení celé webové stránky by tedy vzhledem k průměrné rychlosti stahování trvalo více, než 80 vteřin.

Jiné volby kódování dat, typu DNS záznamů pro komunikaci a dalších parametrů dosahují dle oficiálních informací[27] horších vlastností. Vzhledem k dosaženým rychlostem v optimálních podmínkách nebyly neoptimální konfigurace nástroje *iodine* dále testovány.

Směr	Průměrná rychlost přenosu
Stahování ze serveru	37,2 KB/s
Nahrávání na server	25,5 KB/s

Tabulka 4.6: Tabulka zachycující naměřenou dobu přenosu dat pomocí nástroje iperf3 v obou směrech využitím software *iodine* – DNS záznamy typu NULL

4.3 Testování *captive portálů*

Vytvořené softwarové řešení a vybrané řešení pro DNS tunelování je dále testováno na dostupných *captive portálech*. Avšak jak bylo uvedeno v úvodní kapitole – řešení pro *captive portál* jsou velmi rozmanitá a mnohdy dodaná výrobcem specifického zařízení. Testování tedy proběhlo na vybraných *open-source* řešeních.

4.3.1 *WiFiDog*

Software *WiFiDog* je open-source řešení pro *captive portál*, dostupné od roku 2004. Je vytvořen v jazyce *C* a založen na funkcionalitě `iptables`, které jsou vyžadovány pro správný chod *captive portálu*. Díky volbě jazyka a minimálním systémovým požadavkům je možné používat software nejen na většině Linuxových distribucích, ale rovněž samostatných zařízeních (dedikované směrovače a přístupových bodech WiFi sítí). *WiFiDog* je proto distribuován jako balíček pro platformu *OpenWRT* (a deriváty), která je primárně určena právě pro zmíněná dedikovaná zařízení. Součástí softwarového balíčku je rovněž autentizační server, založen na technologiích PHP a PostgreSQL (případně MySQL).

Výchozí konfigurační soubor `wifidog.conf` zakazuje všechnu komunikaci neautentizovaným uživatelům, ale explicitně povoluje TCP a UDP porty 53 a 67. Díky tomu je možné komunikovat směrem do Internetu pomocí těchto portů. Mimo tyto porty je také veškerý uživatelský provoz před ověřením uživatele směrován pomocí `iptables` na přihlašovací stránku *captive portálu* (port TCP/80).

```
1  [ ... ]
2  # Used for unvalidated users
3  #
4  # XXX The redirect code adds the Default DROP clause.
5  FirewallRuleSet unknown-users {
6      FirewallRule allow udp port 53
7      FirewallRule allow tcp port 53
8      FirewallRule allow udp port 67
9      FirewallRule allow tcp port 67
10 }
11 [ ... ]
```

Ukázka kódu 4.4: Část výchozího konfiguračního souboru captive portálu WiFiDog

Toto softwarové řešení je tedy možné obejít prostým UDP, nebo případně TCP, tunelem. V úvahu přichází i DNS tunelování, které však v tomto případě poskytuje nejhorší vlastnosti tunelu.

Doporučené volby Vzhledem k výrazně lepším vlastnostem `UDP` tunelu oproti `TCP` tunelu lze doporučit tunelování pomocí pluginu `-p udp:53`, případně `-p udp:67`. Jedná se však o doporučení na základě výchozího konfiguračního souboru projektu. *Captive portal* může být v konkrétní veřejné síti provozován se zcela jiným konfiguračním souborem. Protože ale řešení `WiFiDog` nijak nezasahuje do `DNS` provozu, je dost dobře možné, že `UDP` tunelování skrze port 53 bude plošně dostupné v sítích řízených softwarem *WiFiDog*.

4.3.2 *ChilliSpot*

ChilliSpot je open-source řešení pro *captive portál*, dostupné od dubna roku 2004. Poslední zveřejněná verze pochází ze září roku 2006 (verze `1.1`). Projekt je vytvořen v jazyce *C* a založen na funkcionalitě `iptables`, které jsou vyžadovány pro správný chod *captive portálu*. Stejně jako řešení *WiFiDog* je vhodným kandidátem pro použití na dedikovaných zařízeních (WiFi přístupové body, směrovače). Disponuje (pro *captive portál*) typickým webovým rozhraním, ale pro správnou funkčnost požaduje, aby uživatelé měli ve svém prohlížeči otevřené speciální okno, které o aktivitě uživatele periodicky informuje *captive portál*. Toto řešení je krajně nevhodné pro celou řadu zařízení, zejména mobilní telefony a tablety.

ChilliSpot neautentizovaným uživatelům výrazně omezuje možnosti komunikace po síti. Ve výchozím nastavení blokuje veškerý síťový provoz a nepovoluje klientům komunikovat s vlastními `DNS` servery. Díky tomu není možné jako neautentizovaný uživatel navázat tunelové spojení pomocí `TCP`, `UDP` ani `SCTP` pluginů. Pluginu pro `ICMP` tunelování je ve funkčnosti dále bráněno blokováním příchozích `ICMP echo-reply` zpráv. `HTTP` požadavky jsou podrobeny inspekci a neověřenému uživateli je při pokusu o přístup mimo *captive portál* odpovězeno `HTTP` kódem `302 Moved Temporarily` společně s nasměrováním na přihlašovací formulář *captive portálu*.

Jedinou možností, jak komunikovat s okolním světem je prostřednictvím `DNS` serverů, o kterých byl klient informován v rámci `DHCP` odpovědi. S pomocí těchto `DNS` serverů je možné navázat tunelové spojení pomocí `DNS` tunelování. Jedním z často kladených dotazů na oficiální webové prezentaci projektu[29] je povolení přístupu k jiným `DNS` serverům (například pro klienty se statisticky nastaveným `DNS` serverem). Řešením tohoto problému je dle vývojářského týmu projektu *ChilliSpot* zapnutí volby `UAM AnyDNS` doplněné o spuštění lokálního `DNS` cache serveru a úpravu `iptables` pravidel pro směrování veškerého `UDP/53` provozu na tento lokální `DNS` server. Pokud však správce *hotspotu* **pouze** zapne konfigurační volbu `UAM AnyDNS` bez další úpravy `iptables`, bude **všem** klientům umožněno provozovat jakoukoliv komunikaci po Internetu na portu `UDP/53`. S přihlédnutím na náročnost provozu vlastního `DNS` cache serveru existuje pravděpodobnost scénáře, kdy síťový správce nechce, nebo nemůže takový server provozovat, ale současně je nutné respektovat staticky konfigurované `DNS` servery klientů. Doporučený konfigurační soubor na ofi-

ciálních webových stránkách projektu *OpenWRT* **doporučuje** `UAM AnyDNS` zapnout, ale o doplnění `iptables` pravidel se již nezmiňuje[30].

Doporučené volby Jedinou možností, jak komunikovat skrze *captive portál ChilliSpot* s výchozí konfigurací je pomocí `DNS` tunelování. *ChilliSpot* totiž ve výchozím stavu nezpracovává `DNS` dotazy. Pokud je navíc povolena funkci *AnyDNS* a `UDP/53` provoz není násilně směrován na konkrétní server, je možné využít `UDP` plugin pro tunelování (`-p udp:53`). V opačném případě zbývá jediná možnost – `DNS` tunelování pomocí specifikovaných nebo vynucených `DNS` serverů.

4.3.3 CoovaChilli

Projekt *CoovaChilli* se označuje jako nástupce projektu *ChilliSpot*, přestože projekt *ChilliSpot* je stále deklarován jako aktivní. S původním projektem tedy sdílí drtivou většinu zdrojových kódů a tím pádem i funkcionalit. Oproti původní implementaci však přidává některé pokročilé vlastnosti jako inspekci `DNS` provozu a zahazování `DNS` odpovědí, které obsahují jiné záznamy než-li `A`, `CNAME`, `SOA` nebo `MX`. (funkce `DNSPARANOIA`). Doporučené volby pro tunelování vzhledem k „neprůstřelnosti“ jiného provozu, než-li `DNS` proto zůstávají stejné, jako u software *ChilliSpot*.

4.4 Závěry z testování

V kapitole *Testování* byla zhodnocena prostupnost implementovaných tunelových řešení a porovnána s nejsofistikovanějším dostupným řešením pro `DNS` tunelování. Propastné rozdíly v naměřených hodnotách poukazují na fakt, že tunelování pomocí plnohodnotného spojení dosahuje výrazně lepších vlastností, než-li spojení svázané restrikcemi a limitacemi konkrétního nosného protokolu (v tomto případě `DNS`).

Zkoumaná řešení byla dále otestována na vybraných existujících *open-source* řešeních pro *captive portál*. Dle tvrzení správců jednotlivých projektů a dokumentace populární distribuce *OpenWRT* patří vybraná řešení k nejpopulárnějším volbám pro realizaci *captive portálu*. Z úspěšného navázání tunelů skrze testované *captive portály* však nelze vyvodit tak široké závěry, jak by se dle údajné popularity projektů mohlo zdát. Reálně nasazovaná řešení jsou velmi odlišná a často proprietární a **vždy** závisí na konfiguraci konkrétního *hotspotu*.

Závěr

Cílem této diplomové práce byl návrh, implementace a otestování protokolu pro obcházení limitací *captive portálů*. Tento cíl byl naplněn návrhem softwarového řešení s důrazem na snadnou rozšířitelnost pomocí struktury pluginů. Byly implementovány čtyři samostatné pluginy pro tunelování síťového provozu na základě různých technik pro obcházení *captive portálu*. Jednotlivé techniky byly otestovány jak s pomocí specializovaných nástrojů pro měření prostupnosti, tak simulací reálné činnosti uživatele (`WWW` provoz). V poslední fázi testování byla implementace společně s jiným dostupným řešením otestována na reálných implementacích *captive portálu*.

Čtenářům této práce byl poskytnut náhled do problematiky technik řízení síťového přístupu pomocí *captive portálu*, který – jak bylo v rámci práce demonstrováno – není ideálním řešením této problematiky. Praktickým přínosem pro čtenáře může být vyvinutý software pro obcházení omezení *captive portálů*, či odkazovaná současná řešení stejného účelu. Zejména situace `DNS` tunelování je zajímavou, díky existenci `open-source` nástrojů pro tichou exfiltraci dat, nebo pro řízení botnetu.

Práce dále nabízí přehled nejběžnějších metod provozování síťových tunelů skrze *captive portál*, což může čtenář-správce sítě využít společně s vyvinutým software pro otestování stavu provozované sítě.

Možným rozšířením této práce je implementace dalších technik tunelování dat skrze *captive portál*, například pomocí dalších méně známých protokolů transportní vrstvy. Avšak podporu takových protokolů v koncových sítích nelze bezpodmínečně očekávat. Možná je i implementace `DNS` tunelování, pro které je však v současné době dostupná řada komunitou vyvíjených *open-source* nástrojů.

Osobním přínosem této práce je hlubší porozumění problematice síťových tunelů a některých méně známých protokolů transportní vrstvy `ISO/OSI` modelu, jako například protokolu `SCTP`. Dalším přínosem byla samotná implementace konceptů a technik pro obcházení *captive portálů* – kdy teoreticky jednoduchý koncept skýtá řadu překážek v praktické implementaci.

Literatura

[1] Kumari, W.; Gudmundsson, O.; Ebersman, P.; aj.: Captive-Portal Identification Using DHCP or Router Advertisements (RAs). RFC 7710 (Proposed Standard), Prosinec 2015. Dostupné z: https://tools.ietf.org/html/rfc7710

[2] Lauer, O.: *Porovnání systémů pro pokročilou správu připojení k síti*. Bakalářská práce, České vysoké učení technické v Praze, 2017.

[3] Smitka, J.: *Systém pro řízení přístupu do kolejní sítě ZČU*. Bakalářská práce, Západočeská univerzita v Plzni, 2016.

[4] Facebook: Get Facebook Wi-Fi for Your Business [online]. 2013, [cit. 2018-02-20]. Dostupné z: https://www.facebook.com/business/facebook-wifi

[5] Nintendo Co., Ltd.: Consolidated Sales Transition by Region [online]. Duben 2016, [cit. 2018-02-23]. Dostupné z: https://www.nintendo.co.jp/ir/library/historical_data/pdf/consolidated_sales_e1603.pdf

[6] Wireless Broadband Alliance: WISPr 2.0 [online]. Duben 2010, [cit. 2018-02-23]. Dostupné z: https://bitbucket.org/tamias/pywispr/downloads/WBA-WISPr2.0v01.00.pdf

[7] Google Chromium: Certificate Transparency in Chrome - Change to Enforcement Date [online]. Duben 2017, [cit. 2018-02-23]. Dostupné z: https://groups.google.com/a/chromium.org/forum/#!msg/ct-policy/sz_3W_xKBNY/6jq2ghJXBAAJ

[8] Felt, A. P.: Twitter [online]. Leden 2015, [cit. 2018-02-23]. Dostupné z: https://twitter.com/__apf__/status/551083956326920192

[9] Postel, J.: Internet Control Message Protocol. RFC 792 (Internet Standard), Září 1981. Dostupné z: https://tools.ietf.org/html/rfc792

[10] Laliberte, M.: Lessons from DEFCON 2016 – Bypassing Captive Portals [online]. Srpen 2016, [cit. 2018-03-15]. Dostupné z: https://www.secplicity.org/2016/08/26/lessons-defcon-2016-bypassing-captive-portals/

[11] Farnham, G.: Detecting DNS Tunneling [online]. Únor 2013, [cit. 2018-03-18]. Dostupné z: https://www.sans.org/reading-room/whitepapers/dns/detecting-dns-tunneling-34152

[12] Rosa, Z.: *Detekce síťových tunelů v počítačových sítích*. Bakalářská práce, České vysoké učení technické v Praze, 2014.

[13] Pennarun, A.: sshuttle: where transparent proxy meets VPN meets ssh [online]. https://github.com/apenwarr/sshuttle, Březen 2016, [cit. 2018-04-29].

[14] OpenSSL Cryptography and SSL/TLS Toolkit [online]: Manual page of RAND_bytes() [online]. 2018, [cit. 2018-04-23]. Dostupné z: https://www.openssl.org/docs/man1.1.0/crypto/RAND_bytes.html

[15] Braden (Ed.), R.: Requirements for Internet Hosts - Communication Layers. RFC 1122 (Internet Standard), Říjen 1989. Dostupné z: https://tools.ietf.org/html/rfc1122

[16] Srisuresh, P.; Egevang, K.: Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (Informational), Leden 2001. Dostupné z: https://tools.ietf.org/html/rfc3022

[17] Stevens, W.: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001 (Proposed Standard), Leden 1997. Dostupné z: https://www.rfc-editor.org/rfc/rfc2001.txt

[18] Titz, O.: Why TCP Over TCP Is A Bad Idea [online]. Duben 2001, [cit. 2018-04-20]. Dostupné z: http://sites.inka.de/bigred/devel/tcp-tcp.html

[19] Stewart (Ed.), R.: Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), Září 2007. Dostupné z: https://www.rfc-editor.org/rfc/rfc4960.txt

[20] Mockapetris, P.: Domain names - implementation and specification. RFC 1035 (Internet Standard), Listopad 1987. Dostupné z: https://www.rfc-editor.org/rfc/rfc1035.txt

[21] Černáč, M.: *Studie kvaliti parametrů digitálních certifikátů na českém Internetu*. Bakalářská práce, České vysoké učení technické v Praze, 2016.

[22] CZ.NIC, z. s. p. o.: Otevřené DNSSEC Validující Resolvery [online]. 2018, [cit. 2018-04-20]. Dostupné z: https://www.nic.cz/odvr/

[23] Postel, J.: Internet Protocol. RFC 791 (Internet Standard), Září 1981. Dostupné z: https://www.rfc-editor.org/rfc/rfc791.txt

[24] Damas, J.; Graff, M.; Vixie, P.: Extension Mechanisms for DNS (EDNS(0)). RFC 6891 (Internet Standard), Duben 2013. Dostupné z: https://www.rfc-editor.org/rfc/rfc6891.txt

[25] Luci, D.: How DNS Tunneling Works [online]. 6 2018, [cit. 2018-04-20]. Dostupné z: http://inside-out.xyz/technology/how-dns-tunneling-works.html

[26] Revelli, A.: Playing with Heyoka: spoofed tunnels, undetectable data exfiltration and more fun with DNS packets [online]. In *Shakacon 2009*, 2009, [cit. 2018-04-20]. Dostupné z: http://heyoka.sourceforge.net/heyoka-shakacon2009.pdf

[27] Ekman, E.: iodine. https://github.com/yarrick/iodine, Březen 2012, [cit. 2018-04-29].

[28] Tam, C.: [online]. http://www.clifftam.com/much-data-youtube-video-use/, Únor 2015, [cit. 2018-04-29].

[29] ChilliSpot developer team: ChilliSpot FAQ [online]. 6 2007, [cit. 2018-04-20]. Dostupné z: http://www.chillispot.org/FAQ.html#mozTocId304167

[30] OpenWRT FAQ: ChilliSpot [online]. 1 2009, [cit. 2018-04-20]. Dostupné z: https://wiki.openwrt.org/oldwiki/wireless.hotspot.chillispot

Příloha A

Uživatelská příručka

V této příloze se nachází informace pro obsluhu programu, požadavků pro správný běh a instrukce k případnému překladu zdrojových kódů do formy spustitelné aplikace. Popsána je i ukázka použití výsledných programů.

A.1 Požadavky pro prostředí

Softwarové řešení je připravené pro Linuxové prostředí s podporou virtuální síťových rozhraní typu `TUN`. Je vyžadováno, aby bylo základní `TUN` rozhraní přístupné pomocí standardní cesty `/dev/net/tun`. Pro podporu `SCTP` pluginu je klíčová podpora `SCTP`. Stav podpory je možné ověřit příkazem `lsmod | grep sctp`. Konfigurace IP adresy síťového rozhraní je v následujících ukázkách demonstrována příkazem `ifconfig`, který však lze nahradit jiným nástrojem, například `ip`.

A.2 Požadavky pro překlad

Softwarové řešení je celé složeno ze zdrojových kódů jazyka *C*. Pro překlad zdrojových kódů je tedy nutný překladač a dostupné vývojové verze knihoven, které program pro svůj běh vyžaduje. Jedná se o knihovny *OpenSSL* a případnou knihovnu zajišťující podporu protokolu `SCTP`. Ve vybraných distribucích (*Debian*, *Ubuntu* a další deriváty) jsou tyto knihovny dostupné formou balíčků

- `build-essential` – základní nástroje pro překlad zdrojových kódů
- `libssl-dev` – vývojová verze knihovny *OpenSSL*
- `libsctp-dev` – vývojové prostředí pro protokol `SCTP`

Otestovány byly následující verze balíčků:

- `libssl-dev:amd64`, `1.0.2g-1ubuntu13.5`

- `libsctp-dev:amd64`, `1.0.17+dfsg-1`

Otestovaným překladačem je `gcc` verze `7.2.0 (Ubuntu 7.2.0-8ubuntu3.2)`.

A.3 Překlad zdrojových kódů

Pro usnadnění překladu zdrojových kódů obsahuje softwarový projekt soubor `CMakeLists.txt`, který zajistí detekci softwarových závislostí a vytvoří lépe známý `Makefile`. Pro zpracování souboru `CMakeLists.txt` je nutné využít nástroj `cmake`, verze alespoň `3.9`. Vygenerování `Makefile` lze provést spuštěním `cmake .` z adresáře, kde se soubor `CMakeLists.txt` nachází. Po vygenerování `Makefile` je možné spustit samotný překlad zdrojových kódů pomocí příkazu `make`. Demonstrace užití příkazů je zachycena v ukázce kódu A.1

```
1  $ cmake .
2  -- The C compiler identification is GNU 7.2.0
3  -- Check for working C compiler: /usr/bin/cc
4
5  [ ... ]
6
7  -- Generating done
8  -- Build files have been written to: /tmp/code
9
10 $ make
11 Scanning dependencies of target code
12 [  3%] Building C object CMakeFiles/code.dir/src/auth.c.o
13 [  6%] Building C object CMakeFiles/code.dir/src/keyfile.c.o
14
15 [ ... ]
16
17 [ 96%] Building C object CMakeFiles/code.dir/plugins/udp/udp.c.o
18 [100%] Linking C executable code
19 [100%] Built target code
```

Ukázka kódu A.1: Ukázka překladu zdrojových kódů aplikace

Výsledkem překladu projektu je soubor `code`, který slouží jako klientská i serverová část pro navázání a správu tunelových spojení.

A.4 Popis přepínačů

Sestavený spustitelný program podporuje několik různých přepínačů. Jejich seznam lze vypsat použitím přepínače `-h`, zachyceno v ukázce kódu A.2.

Přepínač `-v` slouží k vypsání verze programu (příloha diplomové práce obsahuje software ve verzi `1.0.0`). Po vypsání verze program svou činnost ukončí.

```
$ ./code -h
./code v1.0.0
  -v                  print program version and exit
  -k                  keyfile filename (required for auth)
  -l                  list compiled plugins and exit
  -h                  print help and exit
  -t                  test client connectivity to server and exit
  -p [<name[:port]>,...] use plugin on port, use comma to specify more
       plugins
  -s                  run in server mode
  -c <server>         run in client mode, connect to server ip/
      hostname
```

Ukázka kódu A.2: Výpis podporovaných přepínačů aplikace

Přepínač `-k` slouží pro specifikaci tzv *keyfile*. Jedná se o soubor se sdíleným tajemstvím, s pomocí kterého se mohou klienti autentizovat serveru. Přepínač `-k` je doplňkový, nepovinný. Pokud však server nepoužívá *keyfile*, budou akceptována spojení od všech klientů.

Přepínač `-l` slouží k vypsání seznamu názvů a verzí pluginů, se kterými byl program sestaven a jsou tím pádem k dispozici. Jména (identifikátory) pluginů v prvním sloupci výpisu jsou důležité, protože s jejich pomocí jsou specifikovány pluginy, které budou použity pro tunelové spojení.

```
$ ./code -l
There are a total of 4 plugins:
NAME   VERSION
icmp   1.0.0-icmp
udp    1.0.0-udp
tcp    1.0.0-tcp
sctp   1.0.0-sctp
```

Ukázka kódu A.3: Výpis jmen a verzí podporovaných pluginů

Přepínač `-h` slouží k výpisu nápovědy k použití programu – seznamu přepínačů a krátkých popisků jejich činnosti. Výpis všech podporovaných přepínačů je vyobrazen v ukázce kódu A.2.

Přepínač `-t` lze použít k `testování` konektivity společně s přepínačem `-c`. Pokud bude uživatelem tento přepínač specifikován, software se pokusí navázat tunelové spojení a o výsledky informuje uživatele. Pokud se spojení podaří navázat tunel bude následně uzavřen (přepínač pouze `testuje` možnost spojení). U pluginů na základě samostatných zpráv/datagramů (`ICMP`, `DNS`)

připadá v úvahu situace, kdy při opakovaném testování dostupnosti server nestihne ukončit předchozí spojení a může tak nesprávně informovat uživatele o dostupnosti protějšku. Z tohoto důvodu je doporučeno opakované testování provádět v několikavteřinových intervalech – v závislosti na konstantách `TIMEOUT` jednotlivých pluginů.

Přepínač `-c` a `-s` Povinný přepínač `-c` nebo `-s` specifikuje, zda-li je program spouštěn jako klient (`-c`), nebo server (`-s`). Pro navázání spojení je nutné specifikovat právě jeden z těchto přepínačů. Specifikovat oba přepínače není dovoleno.

Přepínač `-p` Povinný přepínač `-p` je nutné použít pro specifikaci alespoň jednoho pluginu pro komunikaci. Je přípustné specifikovat více pluginů oddělených čárkou. Plugin je specifikován svým názvem (identifikátor), které je možné dohledat ve výpisu podporovaných pluginů (přepínač `-l`). Pokud plugin podporuje specifikaci dalších parametrů (například název portu pro komunikaci), je možné specifikovat tyto parametry za názvem pluginu, odděleného dvojtečkou. Příklad použití je zachycen v ukázce kódu A.4.

```
1 # ./code -s -p udp:4444,icmp,sctp:1234
```

Ukázka kódu A.4: Ukázka spuštění serverové části s více pluginy a jejich parametry

A.5 Příklad použití

Typickým scénářem použití je spuštění serverové části na dostupném stroji připojeného do Internetu a následný pokus o navázání tunelového spojení na tento server ze sítě omezené *captive portálem*.

Serverová část je po spuštění kódu z A.5 připravena, nabízí spojení s pomocí pluginů `ICMP`, `UDP` na portu 4444 a `SCTP` na portu 1234. IP adresa virtuálního rozhraní je `192.168.123.1`. Postup pro připojení klientské části zachycuje výpis kódu A.6.

Ověření funkčnosti tunelu lze docílit například s pomocí služby `SSH`. Není tak možné učinit nástrojem `ping`, protože pro použití `ICMP` pluginu je nutné zakázat systémové odpovědi na `echo request ICMP` zprávy. Toho lze na vybraných Linuxových distribucích (Debian, Ubuntu a další deriváty) docílit příkazem `echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all`

Po úspěšném navázání spojení je možné začít tunel využívat. Například k vytvoření lokální `SOCKS` proxy na portu 8080 pro potřeby komunikujících aplikací příkazem `ssh -D 8080 uzivatel@192.168.123.2`.

```
1  // Spusteni serverove casti
2  # echo tajneheslo > keyfile
3  # ./code -k keyfile -s -p udp:4444,icmp,sctp:1234 &
4  TUN interface created: tun0
5  Running in server mode ...
6  [udp:4444] Starting worker thread (1.0.0-udp)
7  [icmp:0] Starting worker thread (1.0.0-icmp)
8  [sctp:1234] Starting worker thread (1.0.0-sctp)
9
10 // Nastaveni IP adresy virtualniho rozhrani
11 # ifconfig tun0 192.168.123.1
```

Ukázka kódu A.5: Ukázka spuštění serverové části s více pluginy a jejich parametry

```
1  // Otestovani dostupnosti z klientske casti aplikace
2  # echo tajneheslo > keyfile
3  # ./code -k keyfile -c hostname-nebo-ip-adresa.cz -p udp:4444,icmp,sctp:1234
       -t
4  Testing connectivity options to hostname-nebo-ip-adresa.cz ...
5  Testing availability of 3 plugins:
6  Testing 1.0.0-udp:4444 ..... unavailable or already connected
7  Testing 1.0.0-icmp:0 OK
8  Testing 1.0.0-sctp:1234 unavailable or already connected
9
10 // Pripojeni s pomoci dostupneho tunelu
11 # ./code -k keyfile -c dip-test -p icmp &
12 TUN interface created: tun0
13 Connecting to server dip-test ...
14 [icmp:0] Starting worker thread (1.0.0-icmp)
15
16 // Nastaveni IP adresy virtualniho rozhrani
17 # ifconfig tun0 192.168.123.2
```

Ukázka kódu A.6: Ukázka spuštění klientské části s otestováním pluginů a následném připojení s pomocí jednoho z nich

Příloha B

Seznam použitých zkratek

DNS Domain Name System

GSM Global System for Mobile Communications

HTTP Hypertext Transfer Protocol

HTTPS HTTP Secure

ICMP Internet Control Message Protocol

IP Internet Protocol

ISM Industrial, Scientific and Medical radio bands

MAC Media Access Control

MITM Man-in-the-middle

MTU Maximum transmission unit

NAC Network Access Control – řízení síťového přístupu

NAT Network Address Translation

RADIUS Remote Authentication Dial-In User Service

SIP Session Initiation Protocol

SOCKS Socket Secure

SSH Secure Shell

TCP Transmission Control Protocol

TTL Time to live

UDP User Datagram Protocol

VLAN Virtual local area network

VoIP Voice over IP

VPN Virtual private network

WISPr Wireless Internet Service Provider roaming

WPA Wi-Fi Protected Access

WWW World wide web

XML Extensible markup language

Příloha C

Obsah přiloženého CD

- `src`
  - `code` ..... zdrojové kódy implementace
    - `plugins` ..... zdrojové kódy jednotlivých pluginů
    - `src` ..... zdrojové kódy hlavní části programu
    - `CMakeLists.txt` ..... soubor pro sestavení `Makefile`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `attachments` ..... přílohy použité v práci
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF