VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

POBOČKOVÁ VOIP ÚSTŘEDNA ASTERISK A JEJÍ NÁSTAVBY

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Ondřej Melichar

AUTHOR

VEDOUCÍ PRÁCE — Ing. Nikola Papež

SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Ondřej Melichar ***ID:*** 168043

Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Pobočková VoIP ústředna Asterisk a její nástavby

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte open source PBX Asterisk a seznamte se s možnostmi, kterými disponuje. Zaměřte se na distribuci AsteriskNOW a další nástavby. Pro tyto ústředny pak vytvořte fungující zásuvný modul, který bude reagovat na síťové útoky. K realizaci těchto útoků využijte rovněž i hardwarový tester Spirent Avalanche. Popište odlišnosti ústředen mezi sebou v rámci jejich konfigurace, rozhraní, zabezpečení a odolnosti vůči útokům. Srovnejte také funkčnost a kompatibilitu vašeho vytvořeného modulu.

DOPORUČENÁ LITERATURA:

[1] MADSEN, Leif, JIM VAN MEGGELEN a AND RUSSELL BRYANT. Asterisk: The Definitive Guide. 3rd ed Sebastopol, CA: O'Reilly Media, 2011. ISBN 978-0596517342.

[2] DWIVEDI, Himanshu. Hacking VoIP: Protocols, Attacks, and Countermeasures. San Francisco: No Starch Press, 2009. ISBN 978-1-59327-163-3.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 21.5.2018

Vedoucí práce: Ing. Nikola Papež
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Diplomová práce rozebírá možnosti open-source pobočkové ústředny Asterisk, popisuje její funkce a porovnává ji s několika jejími nástavbami. Je zde vysvětlen i pojem SIP stack a také jsou zde konkrétně uvedeni dva jeho zástupci. Dále jsou v práci vysvětlena bezpečnostní rizika technologie VoIP a jsou popsány a později realizovány jednotlivé možné útoky. V rámci testování je zkoumána i možnost vytvoření a realizace vlastního modulu, jeho následné implementace, přenositelnosti mezi jednotlivými nástavbami a jeho funkce.

KLÍČOVÁ SLOVA

VoIP, PBX, Asterisk, SIP, RTP, nástavby, AsteriskNOW, útoky, srovnání, moduly, PJSIP

ABSTRACT

This master's thesis delves into the possibilities of the open-source Private Branch Exchange Asterisk, elaborates on its features and compares it with several other distros. The term SIP stack is explained here with the mention of two of its representatives. Further in the thesis, the security risks of the VoIP technology are explained, and specific attacks are described and then realized. As a part of the testing process, the possibilities of a custom module and its following implementation are explored, as well as the portability between the individual distros and its proper functioning.

KEYWORDS

VoIP, PBX, Asterisk, SIP, RTP, distros, AsteriskNOW, attacks, comparison, modules, PJSIP

MELICHAR, O. *Pobočková VoIP ústředna Asterisk a její nástavby*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2018. 97 s., 1 s. příloh. Diplomová práce. Vedoucí práce: Ing. Nikola Papež

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma Pobočková ústředna Asterisk a její nástavby jsem vypracoval samostatně pod vedením vedoucího Ing. Nikoly Papeže a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji uživatelům z online komunity Asterisk za jejich cenné praktické rady a zejména vedoucímu diplomové práce Ing. Nikolu Papežovi za jeho účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce. Dále bych rád poděkoval společnosti CBL s.r.o. za výborné pracovní prostory a za stálý a spolehlivý přísun kávy. V poslední řadě děkuji společnosti Biomedica za účinnost jejich doplňku stravy Ascoffin.

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Výzkum popsaný v této diplomové práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

S rozvíjející se dostupností internetového připojení tradiční je analogové telefonní vedení využíváno v praxi čím dál tím méně. Toto platí zejména ve firemním, resp. komerčním sektoru. Internetové (datové) připojení dnes nabízí unifikovaný komunikační kanál, přes který lze realizovat telefonní hovory, čímž odpadá potřeba zajištění další kabelové trasy, která by byla vyhrazena striktně pro připojení do veřejné telefonní sítě.

Technologie, kdy je telefonní hovor realizován pomocí internetového připojení je označována jako VoIP (Voice over Internet Protocol). Právě této technologie využívají pobočkové ústředny (PBX). Pobočkové ústředny umožňují v rámci lokální sítě realizovat telefonní hovory bez jakýchkoliv poplatků a to neomezeně. Samy pak mohou být připojeny přípojkou do veřejné telefonní sítě. Zavedení pobočkových ústředen značně snížilo náklady na poplatky telefonním společnostem, a z tohoto důvodu jsou implementovány čím dál tím více.

Komunikace pomocí přepínání paketů s sebou ale přináší také jistá bezpečnostní rizika. Jelikož se jedná o veřejnou síť, je potřeba zabezpečit integritu a bezpečí přenášených dat.

Diplomová práce se věnuje pobočkové ústředně Asterisk a nástavbám, jež jsou na ní založeny. V teoretické části je rozebrána architektura ústředny, číslovací plán, je zde vysvětlen pojem *SIP stack* a jsou zde zmíněny nejznámější konfigurátory této ústředny. Pod pojmem *konfigurátor* si lze představit formu určité softwarové nástavby nad ústřednou Asterisk, která její funkci rozšiřuje a nabízí nástroje (i grafické) na ovládání této ústředny. Pro porozumění dalšího obsahu práce jsou stručně popsány nejdůležitější síťové protokoly, které v této problematice figurují. Práce se dále věnuje důležitému tématu, které s touto oblastí souvisí, a to bezpečnosti technologie VoIP. Jsou zkoumány jednotlivá bezpečnostní rizika, resp. útoky, jejichž princip je v teoretické části popsán.

Praktická část se věnuje nástavbám ústředny Asterisk, externím zásuvným modulům a zkoumá jejich přenositelnost. Za pomoci určené metodiky jsou jednotlivé konfigurátory srovnány a vyhodnoceny. Je vysvětlena architektura modulů a jsou popsány jejich povinné části, které jsou pro funkčnost klíčové. V rámci diplomové práce je sestrojen zásuvný modul, který dovede analyzovat příchozí hlavičku SIP protokolu a reagovat na něj. Součástí testování je testovací systém Spirent Avalanche, na kterém byly vytvořeny vlastní scénáře pro testování bezpečnosti. V závěru práce následuje shrnutí získaných výsledků.

1 TELEKOMUNIKAČNÍ SYSTÉMY

V úvodní kapitole je uvedeno stručné seznámení s přenosem hlasu po telekomunikační síti. Pro uskutečnění nejen hlasové komunikace mezi uživateli je vždy zapotřebí určité množství strukturovaných sítí. Obecně je dělíme na telekomunikační sítě přístupové (Access networks – AN) a sítě transportní (Transport networks – TN). Jedná se nejen o sítě určené explicitně k přenosu hlasu (Public Switched Telecommunication Networks), či sítě datové (IP), ale také třeba hybridní (konvergovaná), kde jedna síť navazuje na druhou, resp. kde jsou tyto síťové architektury propojeny. V následující podkapitole bude krátce popsána architektura PSTN, její vývoj, a princip. Obecný popis a motivace k modernizaci čili přechodu na VoIP telefonii je obdobně popsána v podkapitole 1.1.

1.1 Veřejná telefonní síť

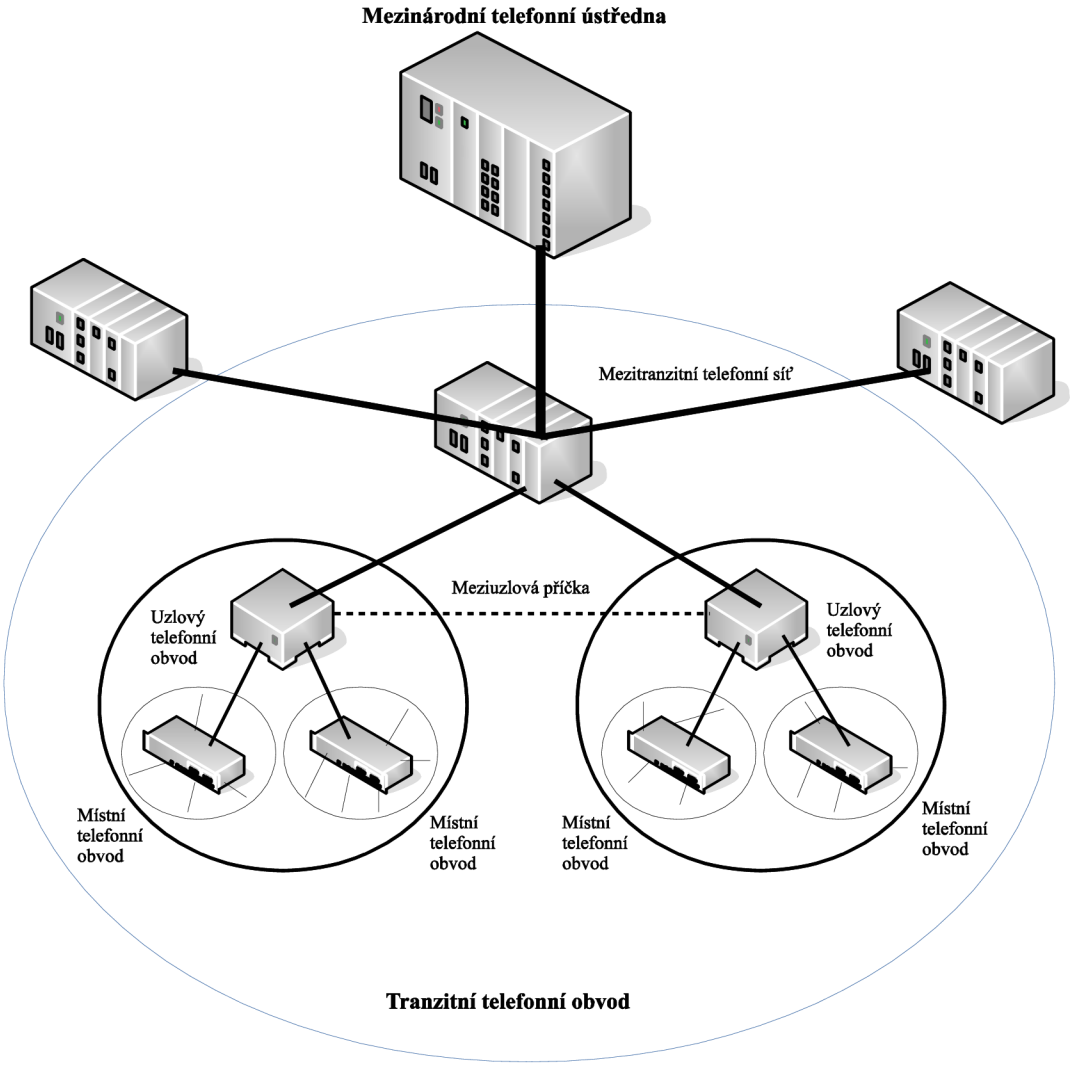
Nejstaršími telefonními přístroji, resp. telefonní službou jsou analogové telefony POTS (Plain Old Telephone Service). První telefonní přístroje měly tzv. místní baterii (MB) což značí, že byly napájeny lokálně. Tato baterie byla zařazena do obvodu mikrofonu. Přístroje s místní baterií měly taktéž induktor, který generoval tzv. vyzváněcí proud a posílal signál spojovatelce v tehdy ještě manuálně obsluhovaných ústřednách. Nynější analogové telefonní přístroje již ovšem místní baterii nemají. Mají naopak tzv. ústřední baterii (UB) či anglicky Central Battery (CB). Jsou tedy napájeny z ústředny. Výjimkou jsou telefony pro vojenské účely, které jsou stále napájeny místní baterií, jež jsou ovšem již v digitálním provedení [20].

U telefonů s ústřední baterií je princip takový, že telefonní přepínač v ústředně telefonní společnosti přivádí do telefonu napětí -48 V. Díky tomuto napětí je telefon schopen vytočit danou klapku. Dráty vedoucí z telefonního přístroje se tradičně značí jako a-vodič a b-vodič. Jakmile ovšem tyto vodiče opustí koncový telefonní přístroj, dostanou nové označení. Spojení mezi telefonem a místní ústřednou nazýváme místní smyčkou. Telefonní číslo vytočené účastníkem se po přivedení do ústředny pomocí právě místní smyčky přepojí k telefonnímu přepínači (switchboard). Tento přepínač, jak již název napovídá, slouží k propojení koncových účastníků, respektive jejich koncových zařízení (telefonních přístrojů). Odposlechem tónů generovaných klávesnicí telefonu nebo interpretací pulsů generovaných rotační číselnicí může přepínač rozpoznat vytočené číslo. Na základě čísla je pak hovor patřičně nasměrován [27].

Původně byly hovory v ústřednách spojovány manuálně. Tuto činnost prováděly operátorky [27]. Nynější telefonní ústředny již samozřejmě obsahují přepínače digitální, které jsou mezi sebou propojeny hierarchickým způsobem. Telefonní přístroj koncového účastníka se nejprve spojí s místní ústřednou, která se následně spojí s ústřednou jí nadřazenou a tak dále. Pokud bychom se zaměřili na strukturu pevné telefonní sítě v České Republice, zjistili bychom, že zde je síť rozdělena na několik úrovní. Pro další popis sítě v ČR je třeba si definovat správnou terminologii. Účastníkem se rozumí koncový uživatel, do jehož domu vede tzv. účastnické vedení. Toto vedení obvykle bývá tvořeno párem vodičů. Pokud se jedná o ISDN přípojku (digitální), jsou zde vodiče čtyři (dva páry).

Samotné účastnické vedení je ukončeno tzv. koncovým bodem sítě, do něhož lze připojit libovolné koncové telekomunikační zařízení. Již bylo zmíněno, že koncové zařízení účastníka je připojeno do nejbližší místní ústředny, z které je napájeno. Právě místní ústředny (MÚ/MTÚ) jsou nejnižší úrovní v hierarchickém žebříčku [28]. Každá tato místní ústředna spravuje území (obvod) o průměru cca 5 km. Toto území nazýváme místním telefonním obvodem (MTO). V současnosti jsou ovšem místní ústředny nahrazeny tzv. koncentrátory (RTU – Remote Subscriber Unit). Koncentrátory neprovádí žádné spojování, jen sbírají a seskupují účastnická vedení směrem k tzv. uzlovým telefonním ústřednám. Uzlové telefonní ústředny (UTÚ) zabírají druhou příčku v hierarchii telefonní sítě ČR. Každá uzlová telefonní ústředna obsahuje zhruba 40 zmíněný koncentrátorů. Analogicky na místní ústředny, se území, které pokrývá UTÚ nazývá jako uzlový telefonní obvod (UTO). Jednotlivé uzlové telefonní ústředny jsou mezi sebou propojeny tzv. meziuzlovými příčkami. V současnosti jsou UTÚ nahrazeny tzv. řídícími telefonními ústřednami HOST, kterých je v ČR aktuálně cca 140 [28].

O úroveň výše nad uzlovými telefonními ústřednami jsou telefonní ústředny tranzitní (TTÚ), kterých je v současnosti na celém území ČR 8. Obdobně jako u MTÚ a UTÚ se i zde oblast, kterou tato ústředna obsluhuje, nazývá Tranzitní telefonní obvod (TTO). K propojení tranzitních ústředen slouží mezitranzitní telefonní sítě. Mezitranzitní telefonní síť je strukturována tak, aby vždy mezi každými dvěma tranzitními ústřednami bylo spojení, které vede maximálně přes tři další tranzitní ústředny. Dálkové tranzitní ústředny následně napojují zmíněné tranzitní ústředny do ústředny mezinárodní (MTÚ), která propojuje telefonní sítě národní do mezinárodní [28]. Samostatnou kategorií jsou pak pobočkové ústředny, kterým je věnována kapitola 1.3. Výše popsaná hierarchie je znázorněna na Obr. 1.



Obr. 1: Struktura telefonní sítě

1.2 Přechod k VoIP

Voice over Internet Protocol, zkráceně VoIP, je poměrně moderní technologie. Často je nazýván i jinak – Voice over BroadBand (VoBB), Internetová telefonie, IP telefonie aj. S touto technologií se v nynější době setká téměř každý, ale vždy tomu tak nebylo.

S příchodem internetu došlo k mnoha modernizacím na poli komunikace a informací. Na úvod je třeba si definovat, co pojem VoIP či Internetová telefonie vlastně znamená. Jedná se o zasílání datových jednotek tzv. paketů přes IP síť. Tato technologie z roku 1995 vznikla původně jako náhrada telefonního spojení na velké vzdálenosti, neboť poplatky za volání do zahraničí byly tehdy značné. VoIP funguje na principu přenosu hlasu mezi IP adresami, což značí, že je třeba hlas v analogové podobě nejprve přeměnit na digitální formu, a poté rozdělit na části o vhodné velikosti k přenosu po dané síti. Těmto částem (paketům) je přidělena směrovací informace. Pakety jsou po přenosu sítí poskládány tak nejlépe, jak to jen jde (používá se termín Best effort).

VoIP by nemohlo existovat bez tří telekomunikačních milníků, které mu předcházeli. Jde o telefon samotný, internet, a IP protokol. Internet poprvé spatřil světlo světa v padesátých letech minulého století, nicméně nejednalo se o internet, jaký ho známe nyní. Byl původně vynalezen společností Advanced Research Projects Agency Network (ARPANET), a šlo o proprietární komunikační síť amerického ministerstva obrany. Sloužila ke komunikaci mezi ministerstvem a armádou. Síť byla v následných letech pronajímána různým společnostem, což vedlo v osmdesátých letech k rozšíření internetu k veřejnosti. S nástupem počítačů (PC) se mohl uživatel připojit do internetu pomocí tzv. dial-up neboli vytáčeného připojení, což bylo účtováno podle času stráveného na internetu. V roce 1989 byl vynalezen protokol HTTP a později také URL, což položilo základy World Wide Webu (WWW) tak, jak ho známe dnes. Adresování bylo zajištěno tehdy novým IP protokolem [23].

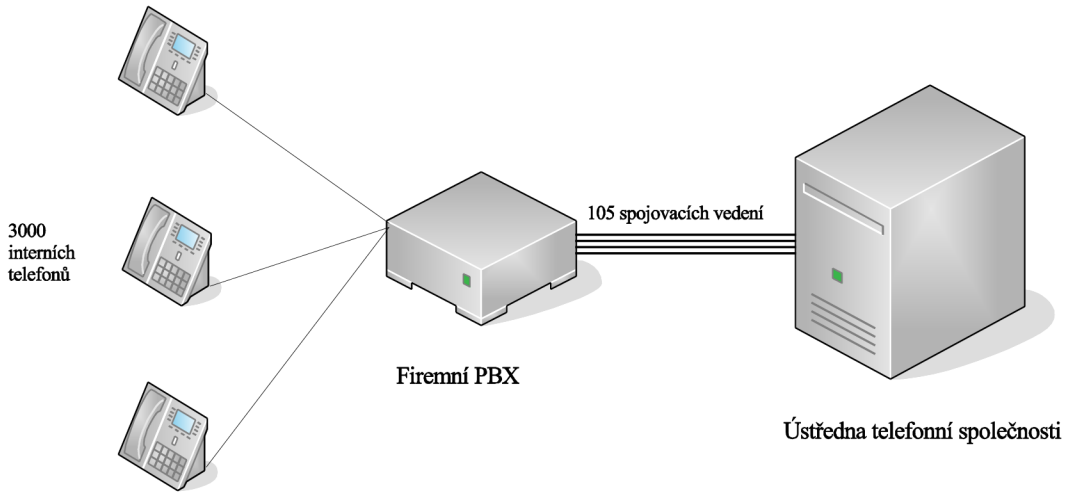
Samotné VoIP bylo spuštěno v roce 1995 firmou VocalTec. Původní název byl InternetPhone. Uživatelé mohli pomocí počítače, mikrofonu a sluchátek volat pomocí softwarového telefonu, který musel být na obou stanicích ve stejné verzi. V této dekádě se o VoIP začaly zajímat hlavně firmy, které by mohli nový digitální přenos využít.

Na konci dvacátého století se na trhu objevil první VoIP switch, resp. software. S tím přišly dokonce tři různé firmy. Tento fakt přispěl k tomu, že v roce 2003 bylo 25 % všech hlasových hovorů vedeno přes VoIP. Dostupnost širokopásmové Ethernetové služby znamenala zlepšení kvality hovoru a s připojením obecně byl menší problém než-li tomu bylo u vytáčeného připojení. Výrobci telekomunikačního hardware začali vyrábět zařízení, které by umělo VoIP hovory přepínat. S předchozí technologií tomu bylo tak, že CPU na účastníkově PC muselo přeměnit datové pakety tak, aby byly srozumitelné pro PTSN veřejnou telefonní síť. Nyní se o tuto funkci staral externí hardware. S postupným vývojem přišly na scénu pobočkové ústředny, které byly a jsou i v současnosti využívané společnostmi a firmami a které pracují s dalšími protokoly na vrstvách architektury ISO/OSI [23].

1.3 Pobočkové ústředny

Pobočková ústředna (PBX – Private Branch Exchange) je neveřejná soukromá ústředna sloužící pouze pro určitý okruh uživatelů. Zjednodušeně by se dalo říci, že pomocí ní je určitá soukromá entita (firma, škola, úřad) připojena k veřejné telefonní síti. Pobočkové telefonní ústředny nyní existují ve formě IP PBX s použitím protokolu IP s komunikací po IP síti a konvenční PBX využívající multiplex s časovým dělením (TDM) komunikující po telefonních linkách. K těmto ústřednám je vždy zapotřebí určitý konfigurační či ovládací software. Rozlišujeme pobočkové ústředny proprietární či open-source. Zatímco komunikace s ostatními entitami (venkovní komunikace) zcela závisí na přenosovém médiu a prvcích kompletně mimo režii PBX, interní komunikace a provoz jsou na ní úplně závislé.

Hlavní motivací pro zavedení PBX do firem byla snaha o zjednodušení a po ekonomické stránce dostupnější komunikaci mezi jednotlivými zaměstnanci či odděleními společnosti. Jako příklad je možno uvést společnost s třemi tisíci zaměstnanci, z nichž každý vlastní telefonní přístroj. V případě absence PBX bychom se dostali do situace, kdy je třeba tři tisíce spojovacích vedení k místní ústředně. Soukromá entita ovšem nemusí mít všechny interní přístroje připojeny přímo k místní telefonní ústředně, ale k vlastnímu telefonnímu přepínači. Ten je připojen k místní telefonní ústředně pomocí spojovacích vedení viz Obr. 2. Těch samozřejmě už nemuselo být tři tisíce, ale např. 105. Statisticky si totiž můžeme domyslet (a samozřejmě spočítat), že nikdy nebudou využívány všechny linky současně, ale pouze určité procento z nich [1].



Obr. 2: Analogová pobočková ústředna

Takto tomu bylo u pobočkových ústředen analogových, které využívali připojení do veřejné telefonní sítě PSTN. S modernizací telekomunikačních technologií a rozvojem internetu ovšem přišly jiné typy ústředen.

Obecně dělíme pobočkové ústředny na:

- analogové (3. generace),
- digitální (4. generace),
- VoIP (5. generace),
- kombinované.

Generaci pobočkové ústředny určuje způsob spojování. Výrobci zpravidla vlivem konkurence nabízí všechny možná rozhraní. V současnosti se setkáváme již zřídka kdy s jiným typem pobočkové ústředny, než 5. (VoIP). U digitálních a VoIP ústředen je umožněno připojení analogových finančně již dostupných koncových zařízení, například pomocí analogových karet [1].

Práce je věnována pobočkové ústředně Asterisk, což je ústředna páté generace, a proto generacím starší již další text věnován nebude.

Jak již bylo řečeno, PBX ústředny dělíme na open-source, a ústředny s uzavřeným kódem, popř. open-source s uzavřeným kódem, což je většinou proprietární nástavba či konfigurační rozhraní nějaké společnosti. V poslední době se do popředí dostávají zejména ústředny s otevřeným kódem. Toto označení znamená, že vlastní zdrojový kód ústředny je k dispozici všem uživatelům s možností libovolných úprav. Tato vlastnost přináší možnost vytvořit si zásuvný modul nebo úpravu, kterou daná firma potřebuje. Naprogramovat takový modul ale není ve většině případech jednoduchá záležitost. Přináší to s sebou totiž nutnost porozumět architektuře dané ústředny a nutnost znalosti daného programovacího jazyka/prostředí. Obecně vývoj jakéhokoliv open-source projektu bývá většinou nekoordinovaná záležitost, protože vývoj je rozdělen mezi mnoho uživatelů. Ústředny tohoto typu mohou být instalovány na běžné PC či servery. Jak bylo zmíněno dříve, doplnit je třeba pouze hardware, který je nutný k vazbě na jiné telekomunikační rozhraní, např. analogové karty. Hlavními zástupci této kategorie jsou ústředny FreeSWITCH, YATE a nejrozšířenější Asterisk.

PBX ústředny druhé kategorie, tedy s uzavřeným kódem s sebou přináší jednoúčelový hardware, který je ovládán pomocí proprietárního softwaru bez možnosti modifikace. Tyto ústředny v počtu instalací stále dominují hlavně díky kvůli spolehlivosti a podpoře. Proprietární PBX jsou ovšem dražší a jejich výrobci mají propracovanou licenční politiku. Mezi proprietární ústředny patří PBX od společností Cisco, Siemens, Panasonic či Alcatel-Lucent. Srovnání výhod a nevýhod těchto dvou kategorií je uvedeno v Tab. 1 a Tab. 2 [1].

Open-source PBX	
Výhody	**Nevýhody**
možnost přidání vlastních modulů	(ne)stabilita projektu
nižší náklady	bez podpory
potenciálně neomezené množství funkcí	obecně náchylnější proti útokům
	složitější obsluha

Tab. 1: Výhody a nevýhody open-source PBX

Komerční PBX	
Výhody	**Nevýhody**
oficiální podpora	vysoké náklady
garance funkčnosti	není umožněno upravovat kód
díky aktualizacím bezpečnější	pouze omezená funkčnost

Tab. 2: Výhody a nevýhody komerční PBX

Kromě zmíněných výhod a nevýhod je třeba u pobočkových ústředen páté generace vzít v potaz i celkový princip VoIP telefonie a otázky, které je nutné u této technologie nějakým způsobem řešit. Jednou z takových otázek je digitalizace, resp. kódování hlasu. Zatím co u technologie POTS (PSTN) je na hovor použita modulace PCM, 64 kb/s na jeden hovor, u technologie VoIP je možností více. Ve světě IP telefonie se používá mnoho kodeků, které ovšem nemusí podporovat každá ústředna či koncové zařízení. Různé podporované kodeky a možnosti různých zařízení je třeba také řešit. Musí existovat mechanismus, který zajistí komunikaci tak, aby ji obě strany byly schopny správně přijmout. K tomuto jsou využity různé protokoly, avšak nejčastěji SIP, resp. SDP, kterému se věnuje kapitola 3.1.1, resp. 3.1.2.

2 POBOČKOVÁ ÚSTŘEDNA ASTERISK

V této kapitole je podrobněji rozebrána PBX Asterisk. Konkrétně stručný úvod do historie následující architekturou, tzn. moduly, souborová struktura, číslovací plán, verze aj.

2.1 Vznik

Počátek vývoje Asterisku se datuje k roku 1994. Jednalo se o tehdy pouze vedlejší projekt zakladatele nynější společnosti Digium, Marka Spencera. Původně vlastnil společnost nazývající se „Linux Support Services“, nicméně po určité době zjistil, že zákazníci a partneři mají větší zájem právě o tehdy ještě nejmenný projekt, Asterisk. Posun nastal seznámením Marka Spencera s vývojářem hardwaru Jimem Dixonem. Spolu se jim povedlo získat dostatečné finance na to, aby roku 1999 vydali první verzi. Zajímavostí byla politika ohledně vlastnictví práv na zdrojový kód. S postupným rozrůstáním společnosti přicházelo více a více zaměstnanců – vývojářů, a každý z těchto vývojářů byl nucen podepsat souhlas s tím, že veškerý napsaný kód bude náležet společnosti Digium a tudíž nebude moct být bez souhlasu použit jinde. Digium tím také mělo otevřenou možnost poskytnutí licence na použití kódu jiným společnostem, jako jsou 3COM nebo NTT [6].

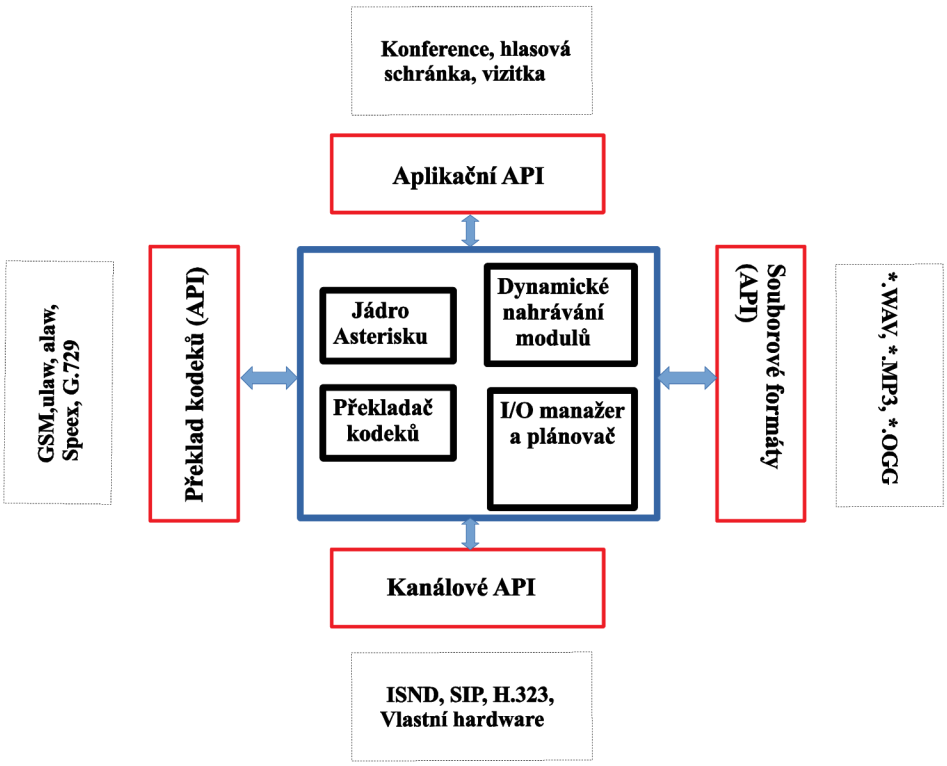
Hlavním zdrojem příjmů společnosti Digium je možnost placené technické podpory a prodeje kompatibilního hardwarového příslušenství. Myšlenkou Asterisku je ovšem open-source, tudíž ať už si uživatel platí podporu či nikoliv, funkčnost ústředna zůstává stejná.

2.2 Architektura

Asterisk je svým charakterem pobočkovou ústřednou značně se lišící od ústředen tradičních, jelikož všechny příchozí kanály obsluhuje stejně.

V tradičních PBX existuje logický rozdíl mezi stanicí (účastnickou sadou) a trunkem (propoj do vnější sítě). Z toho vyplývá, že nelze například připojit externí bránu na port stanice a směrovat na ni externí hovory, aniž by účastník musel napřed vytočit příslušnou klapku. Obecně je velmi obtížné umožnit nějakému vnějšímu prvku (např. externí brána) přístup do interního nastavení a funkcí ústředny[6].

Asterisku se tato omezení ovšem netýkají. Princip je zde takový, že všechen provoz, ať už dovnitř či vně jde přes nějaký komunikační kanál. Existují samozřejmě různé druhy kanálů. Číslovací plán Asterisku (viz. kapitola 2.3) obsluhuje všechny kanály podobně, což znamená, že účastník na konci libovolného trunku může spadat pod správu Asterisku a tím pádem být obsluhován stejně jako by patřil do interní sítě. Struktura ústředny je vyobrazena na Obr. 3 [6].



Obr. 3: Architektura Asterisku

2.2.1 Moduly

Asterisk je pobočková ústředna založená na modulech. Modulem rozumíme určitou přídavnou část, která nám po kompilaci a implementaci do ústředny přidá jistou funkcionalitu. Modulem může být třeba ovladač kanálu (např. *chan_sip.so*), nebo prostředek pro připojení externí technologie (např. *func_odbc.so*). Konfiguračním souborem, který určuje, které moduly a jak budou nahrány, je soubor *modules.conf*, jenž je uložen v adresáři */etc/asterisk/*. Modulová architektura nám v Asterisku umožňuje provést spuštění ústředny v režimu bez jakýchkoliv modulů. V tomto režimu ústředna nemá žádné funkce, což se na první pohled jeví jako bezúčelné, nicméně v konzolovém prostředí lze postupně libovolné moduly nahrávat. Tato vlastnost se využívá při výkonovém ladění ústředny [6].

Na ilustraci v Obr. 3 lze spatřit architekturu Asterisku. Vidíme zde, že pro zaváděné moduly jsou definovány čtyři aplikační programovací rozhraní (API), která slouží k oddělení protokolů a hardware. Díky tomuto modulovému systému nemusí samotné jádro Asterisku řešit detaily jako je např. používaný kodek volajícího. Dle [22] rozlišujeme celkem čtyři API (Application Programming Interface):

a) Kanálové API (Channel API) – toto rozhraní se stará o typ příchozího spojení. Zjišťuje, jestli se jedná o VoIP spojení, ISND PRI či jinou technologii.

b) Aplikační API (Application API) – stará se o úlohy, které souvisí s různými funkcemi ústředny jako je např. konference, hlasová pošta aj.

c) API překladu kodeků (Codec Translation API) – zavádí moduly kodeků sloužící pro podporu odlišných způsobů kódování a dekódování jako je např. G.711a, G.711u aj.

d) API souborových formátů (File Format API) – toto rozhraní ovládá R/W proces (čtení a zápis) odlišných souborových formátů sloužící k ukládání dat do souborového systému.

2.2.2 Souborová struktura

Asterisk je považován za rozsáhlý a komplexní systém, který pracuje s mnoha různými prostředky. Jsou to prostředky různými způsoby využívající souborový systém. Pro porozumění je nutné se s touto strukturou seznámit a zjistit, kde je který konfigurační soubor uložen nebo kde jsou uloženy různá automaticky generovaná data, ať už dočasná nebo trvalá, jako jsou například logy [6].

Konfigurační soubory byly již částečně zmíněny. Jedná se o soubory jako jsou *extensions.conf*, *sip.conf*, *modules.conf* a mnoho dalších, jenž definují parametry pro určité kanály, prostředky, moduly a různé funkce, které mohou být využívány. Tyto soubory jsou ve výchozím nastavení v adresáři */etc/asterisk/*. Složka s moduly už byla popsána dříve. Do tohoto adresáře uživatel běžně nevstupuje, nicméně správce ústředny samozřejmě musí vědět, kde se nachází. Při aktualizaci Asterisku se nekompatibilní moduly ze složky automaticky nesmažou a musí tedy být smazány dodatečně.

Další částí souborové struktury je tzv. spool. Spool je místo, kam se v Linuxu ukládají soubory, s kterými se často pracuje, nebo které čekají na obsloužení jiným procesem. Řadí se sem například požadavky na tisk, nebo emaily čekající na odeslání. V Asterisku se do spoolu ukládají například záznamy hovorů nebo hlasové zprávy. Výchozím adresářem je */var/spool/asterisk*.

Nedílnou součástí ústředen jsou logy. Slouží jako záznam provedených úkonů ústředny a mohou být užitečné při diagnostice. Asterisk je schopen generovat různé druhy logů podle toho, v jakém režimu je spuštěn. Výchozí adresář je umístěn v */var/log/asterisk*.

Poměrně odlišnou filozofii má pak tzv. číslovací plán (dial plan), o němž blíže pojednává kapitola 2.3. Všechny komunikační kanály musí tímto číslovacím plánem projít. Je zde uložen tzv. call-flow skript, který určuje, jak budou obslouženy hovory. Číslovací plán může být napsán třemi způsoby [6]:

- za použití tradiční syntaxe číslovacího plánu Asterisk v */etc/asterisk/extensions.conf,*
- za použití Asterisk Extension Logic (AEL) v */etc/asterisk/extensions.ael,*
- za použití skriptovacího jazyka LUA v */etc/asterisk/extensions.lua.*

2.3 Číslovací plán

Číslovací plán (Dial plan) je jednou z nejdůležitějších součástí ústředny Asterisk. Jeho koncept spočívá ve čtyřech jeho součástech – kontext, klapky, priorita a aplikace.

2.3.1 Kontext

Číslovací plán dělíme na určité sekce, které se nazývají kontexty. Každá entita (pobočka, kanál, trunk) musí být přiřazena právě k jednomu z kontextů. Kontext reprezentuje skupinu poboček (klapek), jenž byly do kontextu přiřazeny v konfiguračním souboru *sip.conf*. Na základě např. prefixu je využito pravidlo s nejlepší shodou. Kontexty slouží mimo jiné také k tomu, aby mezi sebou jednotlivé části číslovacího plánu neinteragovali. Klapka, která je přidělena k jednomu kontextu je úplně oddělena od ostatních klapek, pokud to ovšem v kontextu není explicitně nastaveno. Jako příklad uveďme třeba situaci, kdy se dva účastníci pod stejným Asterisk serverem dovolají přes stejnou klapku jinam. Toto zařizuje právě kontext.

Kontext se utvoří vložením zvoleného jména kontextu do hranatých závorek.

```
[MujKontext_1]
```

Všechny instrukce, vložené za definici kontextu spadají pod daný kontext až do té doby, než je definován nějaký další. Na počátku souboru číslovacího plánu jsou ve výchozí podobě definovány dva kontexty, jmenovitě *general* a *globals*. Ve skutečnosti se ovšem nejedná o kontexty, ale pouze o obecné nastavení číslovacího plánu. Z toho vyplývá, že nelze vytvořit kontexty s obdobným jménem [6].

Kontexty je také potřeba zmínit kvůli jejich dopadu na bezpečnost. Díky správnému používání kontextů, je možno nastavit různým účastníkům různá práva jako je třeba volání do zahraničí a podobně.

2.3.2 Klapka

Ve světě telefonie značí obyčejně slovo klapka nějaký numerický identifikátor, který, když vytočen, zavolá na určité koncové zařízení. V Asterisku mají klapky o něco propracovanější, jelikož zde zastupují unikátní posloupnost kroků (každý z těchto kroků obsahuje nějakou aplikaci), skrze které Asterisk obslouží daný hovor.

V každém kontextu je možné definovat libovolný počet klapek. Každá z klapek může být aktivována buď příchozím hovorem, nebo zadání příslušné posloupnost znaků na číselníku. Jakmile je klapka aktivována, následují ony definované kroky. Klapky mají v číslovacím plánu standardizovanou syntaxi [6].

Standartní syntaxe pro určitou klapku začíná zkratkou *exten*, po které následuje šipka tvořená znakem pro rovnost a znakem „větší než“.

```
exten=>
```

Jako další následuje jméno (nebo číslo) klapky. Vytáčení pomocí jména nebo emailové adresy je jedna z věcí, která dělá Asterisk tak zajímavým. Každý krok procesu vytáčení se tedy skládá ze tří komponent:

- jméno nebo číslo klapky,
- priorita (každá klapka může mít v kontextu přiřazeno více instrukcí (příkazů), které jsou provedeny postupně právě dle priority,
- aplikaci, resp. příkaz, který se v daném kroku provede.

Jednotlivé komponenty jsou odděleny čárkami:

```
exten=> jméno,priorita,příkaz()
```

Jednoduchým příkladem by byla následující řádka. Ta uvádí situaci pro klapku 007, kde je příkaz *Hangup()* proveden s prioritou 1.

```
exten=> 007,1,Hangup()
```

2.3.3 Priorita

Už bylo zmíněno, že každá klapka může mít v kontextu více kroků. Kroky jsou seřazeny dle priorit. Priority jsou číslovány sekvenčně, počínaje jedničkou, a každá z nich spouští určitou aplikaci Asterisku resp. provádí příkaz. Jako příklad je uvedena situace, kdy s prioritou jedna je hovor přijat, poté následují tři další libovolné příkazy a následně je hovor ukončen [6].

```
exten=> 007,1,Answer()
exten=> 007,2,Příkaz1()
exten=> 007,3,Příkaz2()
exten=> 007,4,Příkaz3()
exten=> 007,5,Hangup()
```

Číslování priorit se ovšem ve starých verzích Asterisku setkalo s jistými problémy. V případě, že jistá klapka měla např. 15 kroků, a konfigurátor potřeboval přidat příkaz mezi kroky 3 a 4, musel celý seznam posunout a manuálně přečíslovat. Ve verzi 1.2 se objevilo řešení. Bylo zde zavedeno použití znaku *n* což značí anglické *next*. Pokaždé, když se Asterisk s tímto znakem setká, vezme hodnotu předchozí priority a přidá jedničku. Tímto byla pozdější editace konfigurace značně usnadněna. Použití by tedy bylo následující [6].

```
exten=> 007,1,Answer()
exten=> 007,n,Příkaz1()
exten=> 007,n,Příkaz2()
exten=> 007,n,Příkaz3()
exten=> 007,n,Hangup()
```

Pozdější verze přinesla ještě jedno zjednodušení. Existuje totiž ještě alternativní zápis výše uvedeného kódu, který konfigurátoru ušetří několik znaků kódu. Pokud se kroky týkají stejné klapky, je možno použít *same,* po kterém již nebude třeba klapku specifikovat.

```
exten=> 007,1,Answer()
     same=> n,Příkaz1()
     same=> n,Příkaz2()
     same=> n,Příkaz3()
     same=> n,Hangup()
```

2.3.4 Aplikace

Aplikacemi rozumíme příkazy nebo úkony, které Asterisk provede v daném kroku (např. *Answer()*). Každá taková aplikace nějak pracuje s konkrétním komunikačním kanálem, může se jednat například o přehrání zvuku, čekání na klávesový vstup od účastníka, vyhledání položky v databázi, vytočení hovoru, zavěšení apod. V předchozích příkladech byly použity aplikace *Answer()* a *Hangup()*. Tyto úkony ke svojí funkci nepotřebují žádný další argument. To ovšem pro většinu jiných aplikací neplatí. Argument se aplikaci předává umístěním mezi závorky (platí např. pro *Playback()*).

Úkon *Playback()* slouží k přehrání dříve nahraného zvukového souboru. Není zde očekáván vstup od účastníka, avšak argument musí být přiřazen konfigurátorem. Využije se například při tvorbě IVR (Interactive Voice Response) neboli Interaktivní hlasové odezvy, kde jednotlivé hlášky je možno namluvit a přiřadit jako argument. Asterisk již standardně má ve své distribuci zabudovány profesionálně namluvené hlášky, které může konfigurátor využít. Tyto zvukové soubory je možno zvolit při instalaci Asterisku, poté jsou umístěny do adresáře */var/lib/asterisk/sounds*. Pro správné použití aplikace *Playback* je nutné umístit mezi závorky jméno zvukového souboru bez koncovky. Pokud bychom tedy chtěli přehrát pozdrav, jehož zvukový záznam jsme uložili do patřičného adresáře se zvuky, vložili bychom název tohoto soubor mezi závorky [6].

```
Playback(Pozdrav)
```

Samozřejmě, pokud se soubor nachází mimo výchozí adresář se zvuky, je možno aplikaci předat jako argument patřičnou cestu k souboru.

2.4 Verze

Asterisk je projektem, jemuž se dostává hojného množství aktualizací. Dle aktuální metodologie označování verzí existují dva druhy verzí Asterisku: Long Term Support (LTS) a Standard. Rozdíl mezi těmito druhy spočívá v době podpory. Verze LTS je plně podporována po dobu čtyř let s jedním rokem zabezpečujících záplat navíc. Verze Standard jsou podporovány po dobu jednoho roku s jedním rokem zabezpečujících

záplat navíc. Aktualizace pro oba druhy verzí vychází zhruba každých šest až osm týdnů. Další kategorií jsou pak verze Certified (certifikované). Tyto verze, které mohou být stabilnější variantou zmíněných dvou kategorií, se liší větším počtem testování a zároveň jsou vydávány méně často – typicky dvakrát až čtyřikrát do roka. Certifikované verze Asterisku jsou tedy téměř identické verzím LTS na kterých jsou založené. Aktuálně podporované verze Asterisku je možno spatřit v Tab. 3 [6].

Verze	Typ	Četnost aktualizací
Asterisk 15	Standard	jednou za 6–8 týdnů
Asterisk 14	Standard	jednou za 6–8 týdnů
Certifikovaný Asterisk 13	LTS	2 - 4x ročně
Asterisk 13	LTS	jednou za 6–8 týdnů
Certifikovaný Asterisk 11	LTS	2 - 4x ročně
Asterisk 11	LTS	jednou za 6–8 týdnů

Tab. 3: Aktuální podporované verze Asterisku

2.5 SIP stack

V kapitole 2.2 byla architektura Asterisku. Vyplívá z ní fakt, že Asterisk funguje jako mezilehlá entita, která propojuje jednotlivé telefonní systémy (např. VoIP) a aplikace, které tyto systémy využívají. K jednotlivým funkcím se přistupuje přes odlišná API, jenž komunikují s jádrem přes tzv. kanály. Totožně funguje spolupráce jádra ústředny s protokoly jako jsou SIP či IAX.

O správnou spolupráci s těmito protokoly se stará tzv. *channel driver* (volně přeloženo jako *kanálový ovladač*) neboli *stack*. Těchto stacků obsahuje Asterisk několik, neboť protokolů, které mohou komunikovat s Asteriskem je také několik. Jedná se o stacky *chan_iax2*, *chan_dahdi* a zejména pak *chan_sip*, kterým se tato práce zabývá, protože právě tento stack řídí komunikaci s protokolem SIP.

2.5.1 Původní stack chan_sip

Stack *chan_sip* je pro Asterisk stackem nativním, a jeho vznik se datuje k roku 2002 jen pár měsíců poté, co vzniklo tehdy nové doporučení RFC 3261. V této době ještě nebyl protokol SIP nejvýraznějším protokolem na poli VoIP. Jednalo se o navržení jednokanálové zároveň o jediný samostatně pracující zásuvný modul, jehož zdrojový kód čítá přes 40 000 řádků kódu. V době vzniku byla tato implementace jednoznačně dostačující, nicméně s narůstající expanzí telekomunikační technologií již tento stack přestával být strukturálně dostačující a velmi neflexibilní.

Fakt, že je celá implementace spolupráce se SIP protokolem navržena do jednoho samostatného modulu, mimo jiné také znamená, že je do jisté míry omezena modulární povaha celého Asterisku v oblasti SIP, a tedy i jeho flexibilita. Tyto nedostatky ještě umocnil posun SIP protokolu do popředí VoIP systémů, a proto započala snaha

o inovaci, konkrétně o vytvoření sady modulů, která by jednokanálový stack skládající se z jediného modulu nahradila.

2.5.2 Stack res_pjsip

Stack s názvem res_pjsip je založen na knihovně PJSIP a poprvé se objevil v Asterisku verze 12. Skládá se z několika modulů, které mezi sebou navzájem komunikují a předávají si informace. Značně se zde tedy zvýšil princip modularity. Demonstrativně lze uvést příklad, kdy lze například libovolně aktivovat či deaktivovat jeden z PJSIP modulů, např. *res_pjsip_exten_state*, který monitoruje stav zaregistrovaných koncových stanic, zatím co všechny ostatní moduly si zachovají funkčnost a nezávislost [19].

Zejména tento fakt nahrává vývojářům. Koncept je rozdělen do několika samostatných modulů, a proto je mnohem jednodušší provádět úpravy či přidávat nové moduly do již stávajícího mechanismu i z hlediska četnosti a případné identifikace chyb při vývoji. Z hlediska koncových uživatelů tento stack nevyžaduje jakoukoliv změnu, nicméně z pohledu administrátora je nutno přejít na nový zápis v konfiguračních souborech a také na novou syntaxi v příkazovém řádku Asterisku.

V konfiguračních souborech se změna stacku projeví nejzřetelněji. Místo souboru s klapkami *sip.conf* je použit soubor *pjsip.conf*, jenž má znatelně odlišnou strukturu.

```
[vychozi]
type=transport
protocol=udp
bind=0.0.0.0

[100]
type = endpoint
context = internal
disallow = all
allow = ulaw
aors = 100
auth = auth100

[100]
type = aor
max_contacts = 3

[auth100]
type=auth
auth_type=userpass
password=mojeheslo
username=100
```

V demonstrativní ukázce lze spatřit strukturu definice klapky v konfiguračním souboru *pjsip.conf*. Jedná se o nejnutnější konfigurační minimum. Definice klapky je rozdělena na několik sekcí. V hlavičce souboru je sekce *vychozi*, která platí pro všechny klapky a definuje se zde použitý protokol na transportní vrstvě. Lze si povšimnout, že každá sekce je definována parametrem *type*. V sekci definici klapky 100 se definuje typ *endpoint*, což značí, že se jedná o koncového uživatele. Dále je potřeba definovat, do kterého kontextu danou klapku zařadit parametrem *context*, které kodeky

zakázat, a které povolit. Dále se zde odkazuje na sekci, v které se definuje proces autentizace, a v poslední řadě skupina *aors*, neboli *address of records*. Tento parametr sděluje ústředně, kde je koncový bod k zastižení. Tato sekce může obsahovat více koncových bodů (endpoint). Právě v sekci 100 s nastaveným *type = aor* vidíme, že skupina může mít 3 přidružené koncové body. Toto v praxi znamená, že jedna klapka může být registrována na několika zařízeních, což je z hlediska implementace SIP takřka revoluční. V praxi se i dnes setkáváme se situací, kdy fyzická osoba potřebuje od administrátora několik SIP účtů, neboť se nelze stejnou klapkou přihlásit na více zařízeních [19]. Toto PJSIP eliminuje. Aktuální verze PJSIP ve stacku *res_pjsip* je v Asterisku 2.7. Kromě zmíněných odlišností a výhod obsahuje *res_pjsip* také značné bezpečnostní vylepšení.

2.6 Distribuce

V dnešní době existuje spousta menších i větších projektů věnujících se pobočkovým ústřednám, které se zakládají na Asterisku. Tyto projekty si zakládají na jednoduché obsluze díky vlastnímu grafickému rozhraní. Samotné distribuce jsou šířeny ve formě obrazu disku (.iso soubor), který již obsahuje nějakou distribuci operačního systému. Následuje krátký popis jednotlivých distribucí, kterým se práce v praktické části věnuje. Tyto distribuce jsou podrobněji rozebrány právě v praktické části.

2.6.1 AsteriskNOW

AsteriskNOW je úplná a přizpůsobená distribuce operačního systému Linux s nainstalovaným Asteriskem. Konkrétně se jedná o přizpůsobenou verzi systému CentOS. Není zde ovšem využito klasického provedení Asterisku. Tato verze má administrativní grafické rozhraní FreePBX. AsteriskNOW je určen pro firemní konfigurátory hledající elegantní, a přitom jednoduché VoIP řešení pro firmy, či studenty a další. Projekt má svou vlastní komunitu uživatelů. Firma Digium nabízí komerční podporu i pro tento produkt. Za zmínku stojí fakt, že tato produktová větev nedostává tak časté aktualizace, jako klasický Asterisk [13].

2.6.2 FreePBX

FreePBX je jednou z nejrozšířenějších distribucí Asterisku. Zakládá si na jednoduchém rozhraní a k jejímu ovládání není třeba žádných programovacích schopností. Hlavní rozdíl mezi FreePBX a AsteriskNOW je v tom, kdo se stará o vývoj, resp. přizpůsobení operačního systému k PBX. Zatímco u AsteriskNOW se jedná o společnosti Digium, u FreePBX se o tuto činnost vždy starala společnost Schmooze. Nedávno ovšem došlo k pohlcení této společnosti firmou Sangoma, a projekt nyní tedy spadá pod její správu. Dalším rozdílem je možnost úpravy distribucí. AsteriskNOW je plně open-source, zatímco u FreePBX jsou jakékoliv modifikace zakázány [13].

2.6.3 Elastix

Další distribucí je Elastix. Jedná se o open-source sjednocenou komunikační platformu, která používá operační systém centOS. Zakladateli jsou Edgar a José Landívar ze společnosti Palosanto Solutions. Prvotní verze byla zveřejněna roku 2006. Výhodou byl fakt, že nebylo třeba nic kompilovat. Po instalaci Elastixu dojde k detekci všech přídavných karet a k následné integraci do systému. Poté je možno tuto kartu konfigurovat pomocí webového rozhraní. Kromě klasických funkcí obsažených v IP-PBX podporuje Elastix také fax, e-mail či messaging s anti-spamovou ochranou [5].

2.6.4 AstLinux

AstLinux je velmi originálním projektem. Od ostatních se liší zejména svoji minimalistickou architekturou. Jedná se o projekt, který není pod správou nějaké větší společnosti, nýbrž pouze o malý osobní projekt Kristiana Kielhofnera, který jej vytvořil a stará se o něj. Hlavní myšlenkou této nástavby je především kompaktnost. Účelem bylo vytvořit platformu, který dokáže fungovat na přenosném médiu. Nepotřebuje jako úložiště pevný disk, ale stačí mu paměť flash. Je ideální volbou pro tzv. embedded (jednoúčelové) systémy.

3 BEZPEČNOST VE VOIP

Každá modernizace s sebou nese jistá rizika – nové hrozby. Technologie VoIP je nejen kvůli potenciální informační hodnotě, ale také hlavně díky své rozšířenosti častým terčem všemožných útoků. Tyto útoky můžeme jistým způsobem kategorizovat podle dopadu na danou oběť. Záleží na tom, jestli po získání přístupu k síti útočník zamýšlí odposlouchaná data pouze monitorovat nebo také modifikovat. V následující kapitole bude probrána bezpečnostní rizika a možnosti zabezpečení VoIP komunikace a protokolů, které s touto komunikací souvisí.

3.1 VoIP protokoly

Aby bylo možné pochopit podstatu bezpečnostních rizik, je potřeba pochopit základní principy komunikace ve VoIP. V této oblasti figuruje vícero protokolů, nicméně v rámci teoretického rozboru zde budou popsány protokoly nejznámější a nejvýznamnější.

3.1.1 Session Initiation Protocol

Protokol SIP (Session Initiation Protocol) je protokolem aplikační vrstvy referenčního modelu ISO/OSI. Byl oficiálně vydán roku 1999 organizací IETF (Interenet Engineering Task Force) v doporučení RFC 2543. Jeho nástup byl však pomalý. V té době byl ve VoIP využíván protokol H.323, nicméně v roce 2002 v novém vydání RFC 3261 již H.323 předčil a dostal se do popředí [1], [8].

Protokol SIP je protokolem signalizačním. Zajišťuje inicializaci, modifikaci a terminaci multimediální relace jako je nejen přenos hlasu, ale také i obrazu. Tato relace může probíhat mezi dvěma a více účastníky. Lze jej také použít pro posílání zpráv (např. OfficeSIP). Jedná se o textově založený protokol a je proto podobný protokolům SMTP (Simple Mail Transfer Protocol) či HTTP (Hypertext Transfer Protocol). SIP je pouze protokolem signalizačním a nepřenáší tedy žádná multimediální data. K tomu je použit nejčastěji protokol RTP. SIP pracuje na portu 5060 a je přenášen transportním protokolem UDP. Mezi základní vlastnosti a schopnosti tohoto protokolu patří [8]:

- lokalizace uživatele – při inicializaci relace je volaný účastník vyhledán,
- zjištění schopností – před začátkem relace je stanoveno, jaké vlastnosti mají jednotlivá komunikující zařízení (např. kodeky),
- stav uživatele – možnost přenosu informací o dostupnosti uživatele,
- nastavení relace – možnost nastavení používaných prostředků při zahájení relace,
- změna relace – možnost správy relace, tzn. modifikace, přesměrování či terminace.

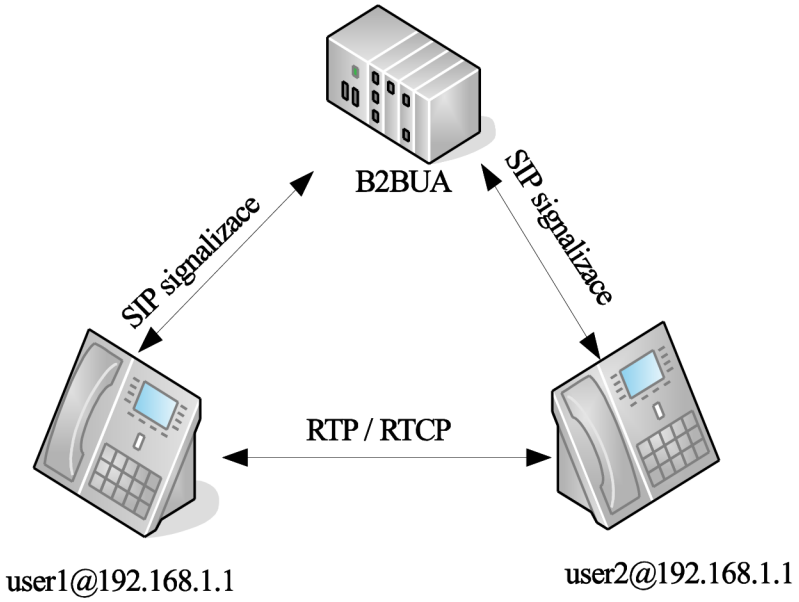
Jak bylo zmíněno, SIP je protokolem, který byl inspirován protokolem jiným, starším. Mimo jiné lze tento fakt pozorovat na adresním schématu, které určuje a odlišuje každý SIP účet. Je použito adresní schéma podobné emailovým adresám. Tento identifikátor

se nazývá SIP URI (Uniform Resource Identifier).

Architektura protokolu SIP se skládá z následujících síťových elementů [8]:

- User Agent Client (UAC) – posílá žádosti a přijímá odpovědi,
- User Agent Server (UAS) – přijímá žádosti a posílá odpovědi,
- Proxy Server – mezilehlý síťový prvek, jehož funkcí je přeposílání žádostí na cílový UAS nebo odpovědí na daný UAC a zajišťuje tedy směrování a mj. také autentizaci a autorizaci účastníků,
- Redirect Server – jedná se o typ UAS, jehož úkolem je posílání SIP zpráv 3XX obsahující informaci o konkrétního uživatele.

V PBX Asterisk je implementován tzv. B2BUA neboli Back-to-Back User Agent, který není definován RFC 3261. Jeho funkcí je generování SIP žádostí jako UAC a odpovědí jako UAS. Hovor mezi dvěma účastníky je tvořen dvěma dialogy viz Obr. 4.



Obr. 4: Komunikace s B2BUA

Komunikaci mezi jednotlivými síťovými elementy zajišťují SIP zprávy. Tyto zprávy lze rozdělit na žádosti, odpovědi, a později přidané zprávy rozšiřující. Žádosti a rozdělení odpovědí dle na třídy dle [1] lze spatřit v Tab. 4 a Tab. 5.

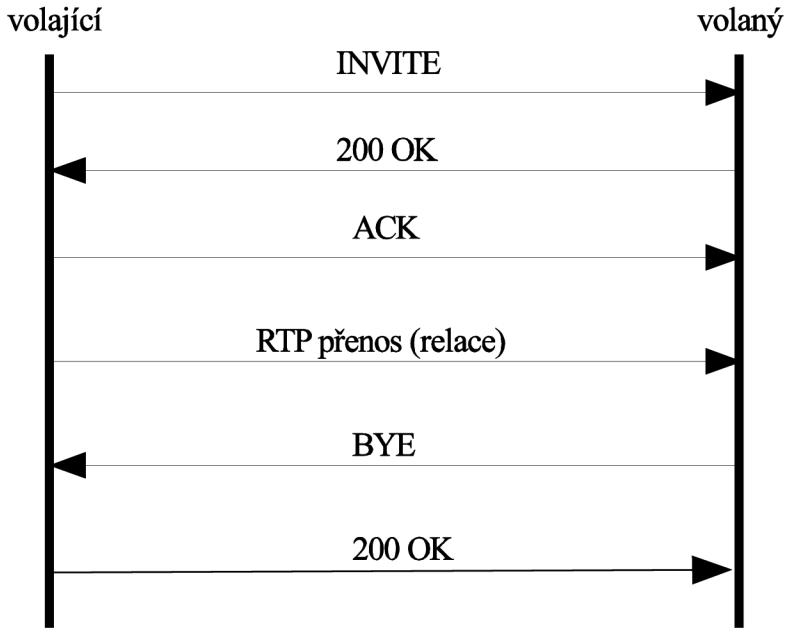
SIP žádosti	
Zpráva	*Funkce*
INVITE	žádost o spojení či modifikaci existující relace
ACK	potvrzení poslední odpovědi
REGISTER	žádost o registraci či zrušení registrace
CANCEL	žádost o ukončení při sestavování spojení
BYE	ukončení probíhajícího spojení

Tab. 4: SIP žádosti

SIP odpovědi	
Třída	*Funkce*
1XX	informační (180 - Ringing)
2XX	potvrzení (200 - OK)
3XX	přesměrování (302 - Moved Temporarily)
4XX	negativní odezva (404 - Not Found)
5XX	chyba serveru (502 - Bad Gateway)
6XX	globální selhání (600 - Busy Everywhere)

Tab. 5: Třídy SIP odpovědí

Pro názornost je na Obr. 5 zobrazen základní tzv. SIP call flow diagram, který ilustruje, jaké zprávy jsou posílány při inicializaci, průběhu a ukončení hovoru přes VoIP telefonii.



Obr. 5: SIP call flow

3.1.2 Session Description Protocol

Session Description Protocol je protokolem tunelovaným. Je zapouzdřen do protokolu SIP/SAP/RTSP a posílá se tedy spolu s nimi. Jedná se o protokol, který popisuje danou relaci. Zajišťuje přenos detailů o přenášených datech, ať už se jedná o streamování videa nebo VoIP. SDP je definován v doporučení RFC 4566 organizací IETF. Protokol je textově orientovaný a používá znakovou sadu UTF-8. Jeho obsah je tvořen několika řádky, jejichž struktura je následující.

```
<typ>=<hodnota>
```

SDP může být rozdělen na celkem tři části. První část popisuje detaily relace a nazývá se „Session Description“. Druhá část se nazývá „Time Description“ která popisuje, jak dlouho je relace aktivní a počet opakování. Třetí částí je „Media Description“, která popisuje typ a detaily multimediálního obsahu. Položky označené hvězdičkou jsou nepovinné.

```
(Session Description)
v= (verze protokol)
o= (původce relace)
s= (název relace)
i= (informace o relaci)*
u= (URI původce)*
e= (emailová adresa původce)*
p= (telefonní číslo)*
c= (informace o typu připojení)
b= (požadovaná šířka pásma)*
z= (nastavení časové zóny)*
k= (šifrovací klíč)*
a= (doplňující popis)*
(Time description)
t= (definuje začátek a konec relace)
r= (počet opakování relace)*
(Media description)
m= (název multimediálního obsahu a transportní adresa)
i= (označení multimediálního obsahu)*
c= (informace o typu připojení)
b= (informace o šířce pásma)*
k= (encryption key)*
a= (doplňující popis)*
```

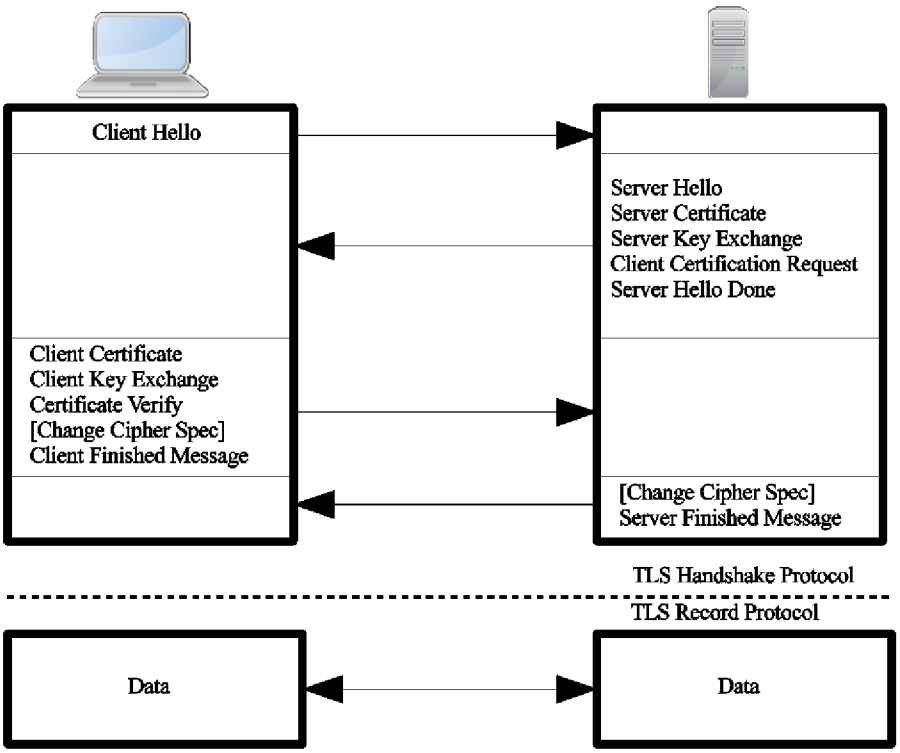
Ve VoIP je SDP protokol klíčový pro stanovení příslušného kodeku pro přenos mezi dvěma SIP entitami a pro přenos informací o daném multimediálním obsahu. SIP telefony ale obyčejně podporují více než jeden kodek. Během SIP zprávy INVITE je tedy také zaslána SDP zpráva, která obsahuje všechny podporované kodeky. SIP entity dbají na pořadí kodeků uvedených v SDP zprávě. Každému podporovanému kodeku je totiž přiřazena priorita. Dle této priority je pak zvolen patřičný kodek.

3.1.3 Transport Layer Security

Protokoly SIP a souběžně také SDP jsou textově orientované. Znamená to, že příslušné zprávy jsou posílány v běžné znakové sadě a jsou tedy při odchytu čitelné všemi. Abychom dosáhli ochrany těchto zpráv, používáme protokol TLS (Transport Layer Security). TLS je nástupcem protokolu SSL (Secure Socket Layer). SSL byl vytvořen společností Netscape za účelem šifrování komunikace mezi webovým prohlížečem a web serverem [24].

Jako většina ostatních protokolů, které jsou využívány protokolem SIP, je i TLS pod správou organizace IETF a je definován v doporučení RFC 5246. Zpočátku nebyly TLS a SSL tak odlišné, ovšem od roku 1999, kdy bylo TLS definováno v doporučení RFC 2246, došlo ke značnému vývinu tohoto protokolu za účelem použití nejen pro web, ale také pro zabezpečení real-time protokolů jako je právě SIP [24].

TLS využívá jak symetrickou, tak asymetrickou kryptografii. Symetrická je použita při autentizaci a asymetrická pro šifrování samotných dat (SIP zprávy). Pro symetrické šifrování jsou vygenerovány jedinečné klíče založeny na tzv. tajemství (secret), které je zajištěno protokolem TLS Handshake. Je nutno zdůraznit, že těmito daty nejsou myšlena data multimediální. Integritu celého spojení zajišťuje protocol TLS Record. SIP zabezpečený protokolem TLS označujeme jako SIPS či SIP secure. TLS je protokolem využívaným v aplikacích klient/server a zabraňuje odposlechu. Průběh komunikace při spojení TLS je znázorněn na Obr. 6.



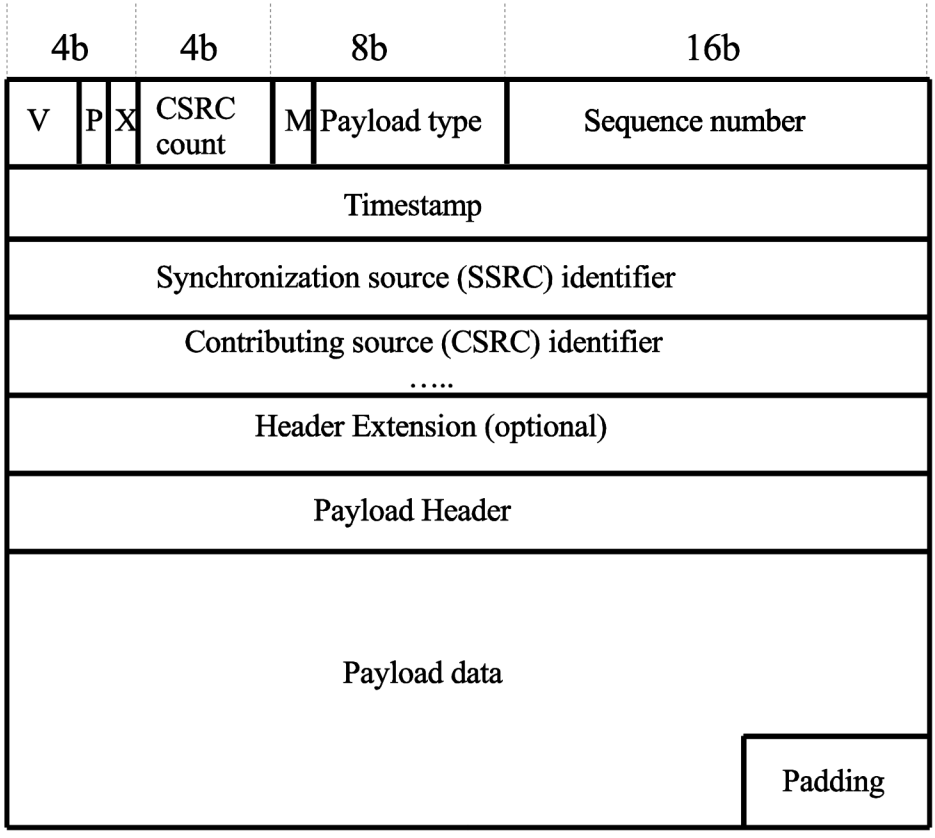
Obr. 6: Průběh TLS komunikace

3.1.4 Real-time Transport Protocol

Služby jako FTP či HTTP využívají pro přenos protokol transportní vrstvy TCP. Tento protokol ovšem vzhledem k jeho vlastnostem není ideálním kandidátem pro použití v případech, kdy se pracuje s multimediálním obsahem v reálném čase jako je třeba právě VoIP, video na vyžádání či videokonference. Lze konstatovat, že je určen zejména pro typ spojení bod-bod. RTP (Real-time Transport Protocol) tok obsahuje samotný multimediální payload (obsah). Tento obsah je přenesen pomocí RTP za spolupráce s protokolem transportní vrstvy UDP, který označujeme jako nespolehlivou síťovou službu. RTP je definován v doporučení RFC 3550 a je spravován organizací IETF.

Z toho vyplývá, že není garantováno doručení paketu, správné pořadí paketu, nebo konstantní rychlosti doručování. Tento problém je vyřešen informačními a režijními políčky v RTP hlavičce. Například díky sekvenčnímu číslování a časovému razítku dokáží aplikace pracující s protokolem RTP zrekonstruovat správné pořadí paketů, nebo detekovat jitter. K tomu slouží adaptivní jitter buffer.

Kontrolním protokolem je RTCP (RTP Control Protocol), který je zde použit pro monitorování kvality služby, sběr statistik, či pro kontrolu samotného přenosu. RTCP vyhodnocuje přijatá RTP data a následně, pokud je potřeba, pošle požadavek odesílateli na přizpůsobení spojení. RTP hlavička a její části lze spatřit na Obr. 7.



Obr. 7: RTP hlavička

Níže jsou shrnuty jednotlivé části RTP hlavičky dle [8].

- V – Version (2b) – značí verzi protokolu. Aktuálně je vždy nastavena na 2 dle doporučení RFC 3550.
- P – Padding (1b) – informuje o počtu doplňkových bytů (tzv. výplň) na konci pole užitečných dat,
- E – Extension (1b) – rozšíření, pokud je bit nastaven na 1, RTP je doplněno rozšířením,
- CSRC count – Contributing source count (4b) – počet přispívajících zdrojů rozšiřující základní hlavičku,
- M – Marker (1b) – je využíván pro oznámení nějaké konkrétní důležité události např. poslední paket snímku,
- Payload Type (7b) – specifikuje typ užitečného obsahu, který je určen profily definovanými v RFC 3551,
- Sequence Number (16b) – sekvenční číslo, podle nějž přijímač může detekovat pakety ztracené či mimo pořadí; toto číslo se s každým paketem inkrementuje o 1,
- Timestamp (32b) – jedná se o časové razítko, které značí okamžik prvního oktetu užitečného obsahu v paketu. Tento údaj je nutno generovat náhodně vždy, kdy komunikace není šifrována (z důvodu bezpečnosti),
- Synchronization Source (SSRC) Identifier – identifikátor zdroje synchronizace identifikuje všechny účastníky RTP relace. Pro každého účastníka se vygeneruje číslo, které pak uživatel používá v každém paketu. Kolizi zde detekuje protokol RTCP,
- Contributing Source (SSRC) Identifier – pole identifikátoru přispívajícího zdroje; obyčejně je generátorem RTP dat jeden zdroj, ovšem v případě použití více zdrojů a tedy následném použití mixeru je každému zdroji takových dat přidělen identifikátor,
- X – Header Extension – pole rozšířené hlavičky následuje po pevné hlavičce; samotné rozšíření je signalizováno bitem X,
- Payload Header – hlavička zátěže je povinné pole, poskytuje obecnou informaci o zátěži,
- Payload Data – samotný užitečný obsah

3.2 SIP VoIP útoky

V kapitole 3.1.1 byla stručně rozebrána problematika protokolu SIP. Také bylo zmíněno, že jelikož jde o nejrozšířenější signalizační protokol dnešní doby, riziko útoku právě za použití protokolu SIP je v praxi velmi vysoké. Je všeobecně známé, že protokol SIP má několik slabých míst, které by potenciální útočník mohl využít. Některé z těchto slabin jsou uvedeny v samotném doporučení RFC 3261. V této kapitole budou zmíněny některé známé typy útoků, jejichž podstata je popsána v doporučení RFC 4475. Konkrétní modifikované útoky budou realizovány v praktické části pomocí testovacího systému Spirent Avalanche. Ta se zaměřuje zejména na útoky následující:

- Missing Required Headers (absence povinných položek v hlavičce),
- Negative Content Length (záporná hodnota délky užitečného obsahu),
- RFC 2543 INVITE (INVITE dle syntaxe doporučení RFC 2543).

3.2.1 Missing Required Headers

Formáty SIP zpráv jsou, jak již bylo zmíněno, definovány doporučeními. SIP hlavička má pevně danou architekturu, která musí být pro správný chod relace dodržována. V praxi ovšem toto není vždy dodržováno. Různé VoIP entity mohou v situacích, kdy nejsou veškeré náležitosti komunikace dodrženy, reagovat odlišně.

Prvním útokem je útok *Missing Required Headers*. Tento typ útoku vyšle zprávu SIP INVITE, která ovšem neobsahuje některé z následujících položek:

- Call-ID,
- From,
- To,

Jedná se o položky, které identifikují hovor, odesílatele a adresáta. Entita, která obdrží takovou zprávu, by měla dle [16] po analýze hlavičky odpovědět zprávou *400 Bad Request*.

3.2.2 Negative Content Length

Druhý útok, který bude realizován v praktické části práce, je útok *Negative Content Length*. U tohoto typu útoku je vyslána zpráva SIP INVITE, která má v poli *Content-Length* negativní hodnotu. Toto pole definuje délku užitečného obsahu. V případě, že jsou SIP zprávy posílány přes protokol UDP (což je obvyklé), by ústředna měla odpovědět chybou. poslaný paket zahodit a čekat na další. Pokud ovšem nastane situace, že zprávy jsou posílány přes TCP protokol, ústředna by měla okamžitě ukončit spojení.

Stejné chování by ústředna měla vykazovat i poté, co místo negativní hodnoty do tohoto pole vložíme hodnotu nečíselnou – tedy například slovo nebo jen libovolné písmeno [16]:

```
Content-Length: osmnáct
```

3.2.3 RFC 2543 INVITE

Dalším typem testování ústředny je *INVITE with RFC 2543 Syntax,* která, jak již název napovídá, odpovídá syntaxi, jenž je uvedena ve starším doporučení RFC 2543. Doporučení pro SIP – RFC 3261 – by mělo být zpětně kompatibilní se svým předchůdcem, a zprávy se starší syntaxí by tedy měly být přijímány. To ovšem opět v praxi není vždy pravda. Rozdíly zpráv INVITE dle jednotlivých doporučení jsou následující [16]:

- V poli *Via* není žádný parametr *branch*,
- V poli *From* není parametr *tag*,
- Není explicitně definována délka pole *Content-Length.*

Jedná se tedy o plnohodnotnou zprávu, a ústředna by ji měla přijmout.

4 ÚVOD DO PRAKTICKÉ ČÁSTI

Praktická část práce se skládá z několik sekcí. Na úvod je za účelem přehlednosti uvedena tabulka se stručným seznamem všech testovaných konfigurátorů ústředny spolu s obsaženou verzí Asterisku a také jejich jádra (kernely). Dalším krokem je představení pracovní stanice, na kterou budou jednotlivá prostředí instalována, resp. její hardwarové specifikace a softwarové vybavení. Poté je popsána metodika testování těchto jednotlivých konfigurátorů pobočkové ústředny Asterisk. Následuje popis testovacího systému, který je využíván v rámci testování, viz kapitola 4.4.

4.1 Testované konfigurátory

Předmětem testování jsou konfigurátory pobočkové ústředny Asterisk. Tyto nástavby se liší v několika parametrech. Každý konfigurátor je pod správou určitě společnosti, která si k implementaci dané verze Asterisku zvolila vhodný operační systém s konkrétním kernelem. Je testována i verze čistého Asterisku jako reference pro grafické nástroje. Přehled testovaných objektů je uveden v Tab. 6.

Název	Verze	Verze Asterisku	Operační systém	Kernel OS
Asterisk	-	14.16.0	Debian 9	4.9.0-3
AsteriskNOW	10.13.66-17	13.12.1	SHMZ 6.6 (CentOS)	2.6.32-642.6.2.el6
AstLinux	1.3.1.	13.17.2	AstLinux (Gentoo)	3.16.47
Elastix	2.5.0	11.13.0-0	CentOS 5.1	2.6.18-371.1.2.el5
FreePBX	14.0.1.19	15.0.0	Sangoma Linux 7 (CentOS)	3.10.0-693.2.2.el7

Tab. 6: Testované konfigurátory

4.2 Pracovní stanice

Kromě verze kernelu (jádra) mají na výkon ústředny vliv hardwarové parametry stanice. Jednotlivé komponenty serveru (virtualizovaného systému), jsou vypsány v Tab. 7.

Komponent	Typ
Procesor	Intel Core i5-3570 @ 3.40 GHz
RAM	1 GB
Architektura	64-bit

Tab. 7: Hardwarové parametry serveru

V průběhu praktického testování jednotlivých konfigurací bude využíváno několika základních softwarových nástrojů:

- Wireshark – Síťový nástroj pro analýzu provozu na síti, resp. jednotlivých paketů. Lze také použít k přehrání audia po zachycení RTP streamu.
- WinSCP – Open-source SFTP, FTP a SCP klient pro Windows. Slouží pro přenos souborů mezi lokálním a vzdáleným počítačem.
- Putty – open-source SSH a Telnet klient,
- Zoiper – SIP softphone pro VoIP komunikace přes zvolenou pobočkovou ústřednu,
- VirtualBox – slouží jako nástroj pro virtualizaci operačních systému,
- Spirent Testcenter – slouží jako prostředí pro tvorbu a konfiguraci scénářů pro tvorbu útoků, které provádí testovací systém Spirent Avalanche.

Během pozdější implementace metodiky testování může být použito i dalšího specifického softwaru a například linuxových balíčků, které jsou ovšem popsány přímo v daném kroku testování.

4.3 Metodika testování

Struktura zvolené metodiky se skládá z celkem tří samostatných kapitol. Úvodní kapitola se zaobírá testovanými konfigurátory a bude zde proveden jejich rozbor. Následující kapitola se věnuje zásuvnému modulu, který byl vytvořen v rámci širšího testování zmíněných konfigurátorů pobočkové ústředny Asterisk. Poslední kapitola patřící do spektra praktického testování se zaobírá síťovými útoky využívající protokolu SIP.

Pro instalaci jednotlivých konfigurátorů je využito virtualizačního prostředí programu VirtualBox. Instalace bude realizována pomocí obrazů instalačních CD (.iso souborů). Jako první proběhne instalace čistého prostředí Asterisk, jehož obraz disku bude stažen z oficiálních stránek společnosti Digium. Bude se jednat o referenční instalaci, podle které pak budou hodnoceny jednotlivé nástavby. U každé takové nástavby budou zkoumány různé možnosti nastavení a úprav, jako jsou SIP účty či kontexty. Bude taktéž zkoumána propracovanost a intuitivnosti grafického prostředí a modulů daného projektu. Testovaní jednotlivých konfigurátorů bude provedeno v rámci samostatné kapitoly 5, resp. 8.

Jako nástroj pro širší testování ústředen bude nastudována problematika vlastních zásuvných modulů a následně bude vytvořen vlastní modul, který bude moci být nahrán do pobočkové ústředny Asterisk. Tento modul bude na úvod popsán teoreticky, následně proběhne implementace takového modulu do jednotlivých nástaveb a bude sledována jeho správná funkčnost tak, jak je této kapitole věnující se modulu popsána. Pro popis vytvořeného modulu je vyčleněna kapitola 7.

Každá jednotlivá nástavba bude také podrobena jistému typu útoku. Útoky, resp. jejich scénáře budou vytvořeny a provedeny pomocí testovacího systému Spirent Avalanche, který je popsán v podkapitole 4.4. Konkrétní typy útoků, zejména jejich princip a podstata jsou popsány taktéž v samostatné kapitole 6.

4.4 Testovací systém Spirent Avalanche

Zařízení Spirent Avalanche nabízí ucelený způsob podrobného testování sítě na čtvrté až sedmé vrstvě referenčního modelu OSI. Tester dokáže generovat reálný provoz, který zvládne poslat do i velmi obsáhlé sítě (až miliony uživatelů nebo velké množství aplikačních serverů) [12].

Testovací systém Spirent Avalanche je součástí sady, v které je mimo fyzického hardwaru také softwarová výbava. Mezi tyto aplikace patří následující nástroje:

- Spirent Testcenter Layer 4-7,
- Spirent Testcenter Result Analyzer 4.33,
- Spirent Testcenter Application 4.33,

V rámci testování je využíváno nástroje Spirent Testcenter Layer 4-7. V této aplikaci je možno vytvářet scénáře zahrnující testování síťové infrastruktury, cloudu, virtuálních prostředí, webových aplikací nebo Triple Play služeb, které zajišťují QoS (Quality of Service) a QoE (Quality of Experience) a to zejména z hlediska kapacity, výkonu a bezpečnosti [4]. Pomocí vytvořených scénářů lze na portech testeru simulovat funkce aplikačních serverů nebo generovat reálný síťový provoz. Tento provoz může být unicast, multicast či VoD, lze tedy generovat jak audio, tak i video, ale také i menší datové zátěže [12]. Mezi podporované protokoly patří: TCP, UDP, SSLv2, SSLv3, TLSv1, SSLVPN Cisco Anyconnect™, HTTP, HTTPS, FTP (Active/Passive), DNS, wTELNET, SMTP, POP3, IMAP4, RADIUS, MM4, ICMP, CIFS, BitTorrent, Gnutella, MSN, Yahoo, SKYPE, SQL, MYSQL, Oracle, SMB, NFS, Remote Desktop, Exchange, LDAP, SIP, SIPS, RTP, RTCP, PPPoX, 802.1Q VLAN tagging, Q-in-Q tagging a jiné [4].

5 TESTOVANÉ NÁSTAVBY

5.1 Asterisk

Prvním z testovaných objektů je čistá instalace pobočkové ústředny Asterisk ve verzi 14.6. Jedná se o standardní verzi bez jakéhokoliv grafického rozhraní. Právě na takové instalaci tvůrci zakládají svá grafická rozhraní a snaží se optimalizovat jimi zvolenou distribuci operačního systému na určitou verzi Asterisku.

5.1.1 Instalace

Pro čistou instalaci pobočkové ústředny Asterisk byla zvolena metoda instalace ze zdroje. Tato metoda má výhodu v tom, že v průběhu procesu instalace je možno zvolit moduly, které chceme do ústředny nainstalovat. Tohoto faktu se využívá například tehdy, pokud je cílem instalace výkonnostní testování ústředny.

Před samotnou instalací je nutno vybavit operační systém, na který bude ústředna instalována, patřičnými balíčky [6]. Debian využívá balíčkovací systém *apt* podobně, jako tomu je u linuxové distribuce Ubuntu. Jako první je potřeba nainstalovat balíček *build-essential*, který v sobě obsahuje nástroje nutné pro kompilační proces – nástroj *make*, kompilátory *C/C++* a *gcc*.

```
apt-get install build-essential
```

Dalšími nutnými balíčky jsou: *libxml2, libxml2-dev, openssl, libssl-dev, libncurses5, libncurses5-dev, libnewt0.52, libnewt-dev, libsqlite3-dev*. Do terminálu tedy vložíme následující sekvenci příkazů [6].

```
apt-get install libxml2
apt-get install libxml2-dev
apt-get install openssl
apt-get install libssl-dev
apt-get install libncurses5
apt-get install libncurses5-dev
apt-get install libnewt0.52
apt-get install libnewt-dev
apt-get install libsqlite3-dev
```

Poté je nutno systém restartovat. Jakmile jsou všechny knihovny a balíčky nainstalovány, je možno přejít k instalaci Asterisku. Ústřednu lze stáhnout z oficiálních stránek na adrese *https://downloads.asterisk.org/pub/telephony/asterisk,* kde si uživatel může vybrat verzi. Jsou zde verze aktuálně testované, které mohou být nestabilní, ale také současné stabilní verze či verze předešlé. V rámci testování je využito verze 14.6.

```
wget https://downloads.asterisk.org/pub/telephony/asterisk/asterisk-
14-current.tar.gz
```

Často se kromě vlastní ústředny stahují taky dva dodatečné podpůrné balíčky – DAHDI a LIBPRI. Knihovna DAHDI umožňuje Asterisku komunikaci s analogovými a digitálními telefony a telefonními linkami včetně připojení do Veřejné telefonní sítě (PSTN). DAHDI (Digium Asterisk Hardware Device Interface) je soubor ovladačů a nástrojů pro většinu analogových karet jako jsou například karty právě od společnosti Digium. Ovladače DAHDI jsou nezávislé na Asterisku a mohou být využívány i dalšími aplikacemi. LIBPRI umožňuje Asterisku komunikaci DAHDI s ISDN hardwarovými rozhraními, jako jsou např. karty E1, BRI, T1 [6]. Tyto knihovny jsou k dispozici opět v online repozitáři Asterisku. V práci není zmíněných knihoven využito, neboť pro testovací účely nebyly vyžadovány, nicméně v případě potřeby je možné je dodatečně nainstalovat.

```
wget https://downloads.asterisk.org/pub/telephony/libpri/libpri-
current.tar.gz
wget https://downloads.asterisk.org/pub/telephony/dahdi-linux/dahdi-
linux-complete-current.tar.gz
```

Nyní jsou staženy všechny potřebné soubory a je možno přejít k samotné instalaci. Stáhnutím výše uvedených dat jsou získány formáty souborů typu *.tar.gz*. Tento typ souboru je označován jako *tarball*. Tarbally je ovšem první nutno extrahovat do patřičné složky tak, jak je uvedeno níže.

```
cd /usr/local/src
tar -zxvf libpri-current.tar.gz
tar -zxvf dahdi-linux-complete-current.tar.gz
tar -zxvf asterisk-14-current.tar.gz
```

Po dokončení procesu jsou extrahovaná data ve zvoleném umístění a pro každý tarball je vytvořena složka dle jeho názvu. Po přepnutí do patřičné složky je možno začít s kompilací a instalací [6].

```
#instalace a kompilace knihovny DAHDI
cd dahdi-linux-complete-current
make
make install
make config

#instalace a kompilace knihovny LIBPRI
cd ..
cd libpri-current
make
make install
```

V tomto kroku jsou splněny všechny předpoklady pro instalaci samotného Asterisku.

Před spuštěním kompilace je ovšem doporučeno pomocí příkazu *./configure* spustit kontrolu systému a připravit kód k procesu kompilace. Tato kontrola může trvat až několik minut a výstupem je buď v kladném případě vykreslení loga Asterisku do terminálu a v případě záporném oznámení, že kontrola nebyla úspěšná a je nutno doinstalovat některé balíčky.

```
cd asterisk-14-current
./configure
```

Následně je potřeba instalační soubory zkompilovat pomocí příkazu *make*. Kompilační proces trvá několik minut. Na konci úspěšné kompilace by se měla objevit následující zpráva:

```
+--------- Asterisk Build Complete ---------+
+ Asterisk has successfully been built, and +
+ can be installed by running:              +
+                                           +
+                make install               +
+-------------------------------------------+
+--------- Asterisk Build Complete ---------+
```

Zbývá zkompilovaný build nainstalovat. To je provedeno příkazem *make install*. Výstupem je zpráva o úspěšné instalaci.

```
+---- Asterisk Installation Complete -------+
+                                           +
+    YOU MUST READ THE SECURITY DOCUMENT    +
+                                           +
+ Asterisk has successfully been installed. +
+ If you would like to install the sample   +
+ configuration files (overwriting any      +
+ existing config files), run:              +
+                                           +
+                make samples               +
+-------------------------------------------+
+---- Asterisk Installation Complete -------+
```

Nyní je instalace ústředny Asterisk hotova. Instalační průvodce ještě na závěr nabízí instalaci ukázkových sample souborů, která je ale pouze dobrovolná. Ústřednu lze spustit následujícím způsobem:

```
asterisk -cv
```

Parametr *c* značí, že chceme spustit ústřednu a přepnout se do její konzole (CLI – příkazový řádek). Parametr *v* je zkratka slovíčka *verbosity* neboli volně přeloženo *podrobnost*. Tento parametr určuje, kolik debugovacích informací se bude vypisovat do

konzole Asterisku během jeho chodu. Tuto podrobnost lze nastavovat podle toho, kolik informací je zrovna potřeba.

5.1.2 Konfigurace SIP účtů

Předpokladem pro uskutečnění VoIP hovoru jsou účty SIP. Tyto účty se u Asterisku nastavují pomocí souboru *sip.conf*. Jedná se o účty, kterými se uživatelé přihlašují k ústředně, ale také o účty, kterými se ústředna přihlašuje k jiným ústřednám a tvoří tak trunk. V rámci testování budou vytvořeny dva SIP účty, mezi kterými bude probíhat telefonní hovor. Možná struktura tohoto konfiguračního souboru je následující [21]:

```
[general] ;nejde o účet, nýbrž o část, která určuje určitá společná
nastavení pro všechny účty

bindport=5060 ;port na kterém naslouchá SIP
srvlookup=yes ; značí podporu DNS
qualify=yes ;značí, jestli ústředna monitoruje status klientů
disallow=all ;zakáže všechny kodeky
allow=alaw ;souvisí s předchozím, povoluje kodek alaw
dtmfmode=auto ;určení tónové volby DTMF
language=cz ;výchozí jazyk
allowguest=no ;povoluje hovory pro neregistrované klienty
context = kontext1 ;určuje kontext, do kterého spadá uživatel


;registrace ústředny jako klienta
[poskytovatel] ;název účtu k poskytovateli
name= poskytovatel; registrační jméno
secret=heslo123 ;heslo
type=peer ;má vliv na autentizaci, peer - servery, user - klient,
 friend - oba
fromuser=101 ;parametr fromuser v protokolu SIP (pokud je vyžadován)
fromdomain=ab.cz ;parametr fromdomain v protokolu SIP (pokud je
vyžadován)
host=asterixpbx.cz ;IP adresa či DNS název ( dynamic - dynamicky
přidělená registrací)
canreinvite=no ;značí, jestli RTP pakety tečou mimo ústřednu
context = public ; určuje kontext, do kterého spadá uživatel

[ucet1] ;uživatelský účet - název (nebo číselná klapka)
name=ucet1 ;registrační jméno
secret=heslo_ucet1 ;heslo
type=friend ;friend - obousměrná autorizace
host=dynamic ;dynamicky získaná IP hosta pomocí registrace
canreinvite=no
nat=yes ;značí, jestli se klient nachází za NATem
context = local ; určuje kontext, do kterého spadá uživatel
```

Je zřejmé, že pro jednoduchý hovor mezi dvěma uživateli není nutno, aby jednotlivé sekce konfiguračního souboru obsahovaly úplně všechny položky. V rámci testování je konfigurační soubor *sip.conf* následující:

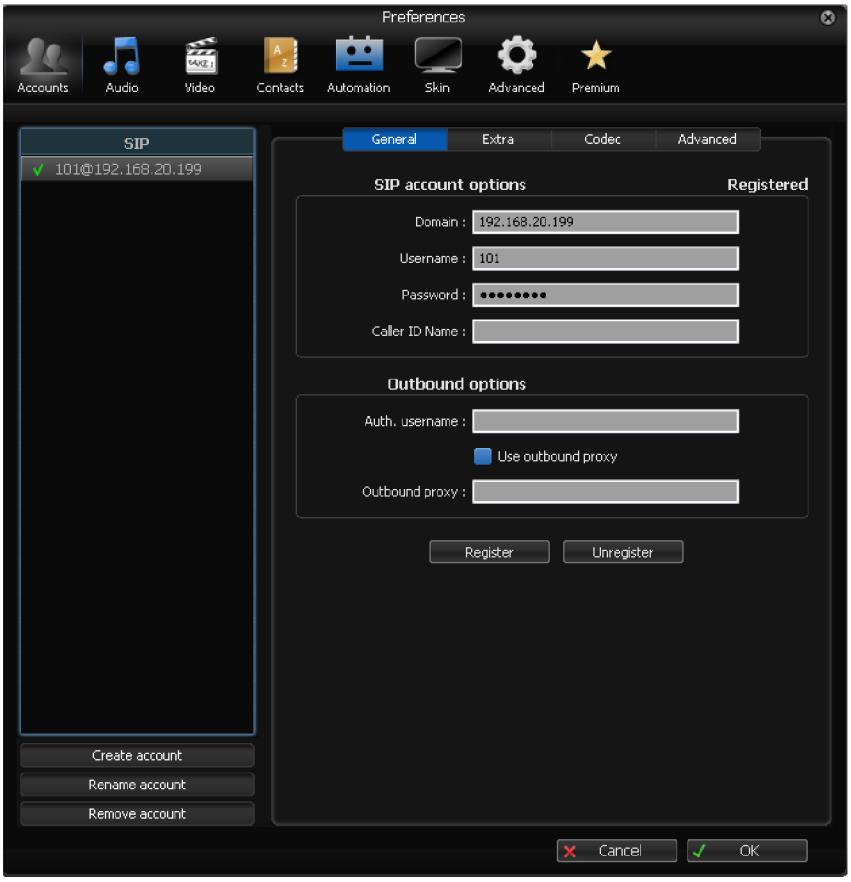
```
[general]
bindport=5060
srvlookup=yes
qualify=1000
disallow=all
allow=alaw
dtmfmode=rfc2833
language=cz
allowguest=yes


[101] ;První účet
type=friend
username=101
secret=heslo101
host=dynamic
context=sip
callerid=101

[102] ;Druhý účet
type=friend
username=102
secret=heslo102
host=dynamic
context=sip
callerid=102
```

Byly definovány dva účty SIP s dynamickou adresou hosta patřící do kontextu SIP. Tyto účty ovšem po uložení konfiguračního souboru ještě registrovat nelze. Po každé úpravě v konfiguraci je nutno obnovit Asterisk a zanést tak novou konfiguraci do jádra. To je provedeno příkazem *config reload /etc/asterisk/sip.conf*, který je potřeba zadat do příkazové řádky Asterisku.

Po úspěšném znovunačtení je možno provést registraci. K tomu je využit softwarový telefon Zoiper. Po konfiguraci v klientu a následné registraci účtu na serveru by měl vytvořený účet indikovat úspěšnou registraci viz Obr. 8. Obdobným způsobem je možné vytvořit účty pro protokoly IAX a DAHDI. Pro ty jsou vyhrazeny konfigurační soubory *iax.conf* a *chan_dahdi.conf*.



Obr. 8: Nastavení softwarového telefonu Zoiper

Následuje ověření na straně serveru. O procesu registrace informuje terminál (CLI – Command Line Window).

```
debianMel*CLI> sip show peers
Name/username             Host                                    Dyn Forcerport Comedia    ACL Port     Status      Des
cription
101/101                   (Unspecified)                            D  Auto (No)  No             0        UNKNOWN

102/102                   (Unspecified)                            D  Auto (No)  No             0        UNKNOWN

2 sip peers [Monitored: 0 online, 2 offline Unmonitored: 0 online, 0 offline]
[Nov 12 17:20:42] NOTICE[888]: chan_sip.c:24648 handle_response_peerpoke: Peer '101' is now Reachable. (2ms / 1000ms)
[Nov 12 17:20:42] NOTICE[888]: chan_sip.c:28455 handle_request_subscribe: Received SIP subscribe for peer without mailbo
x: 101
debianMel*CLI> sip show peers
Name/username             Host                                    Dyn Forcerport Comedia    ACL Port     Status      Des
cription
101/101                   192.168.20.121                           D  Auto (No)  No         54376    OK (2 ms)

102/102                   (Unspecified)                            D  Auto (No)  No             0        UNKNOWN

2 sip peers [Monitored: 1 online, 1 offline Unmonitored: 0 online, 0 offline]
```

Obr. 9: Oznámení o dostupnosti klienta

Na Obr. 9 lze pozorovat, jak ústředna informuje o novém klientovi. Pomocí příkazu *sip show peers* dostaneme výpis účtů SIP obsažených v konfiguračním souboru *sip.conf*. Na počátku ústředna klienty eviduje ale vede je jako nedostupné. Po registraci pomocí programu Zoiper a následné opětovné kontroly vidíme, že klient 101 je nyní dosažitelný na dané IP adrese.

5.1.3 Konfigurace kontextu

V předchozí kapitole proběhla registrace klapky 101. Obdobně bychom registrovali i klienta druhého (např. 102). V okamžiku úspěšné registrace obou klientů ovšem hovor stále nebude realizován, a to z důvodu nenastaveného kontextu. Obě klapky patří do kontextu „sip“, ten je ovšem nutno prvně vytvořit. Kontexty se definují v konfiguračním souboru *extensions.conf* v adresáři */etc/Asterisk/*. Jeho struktura je následující [21]

```
[general]               ;obecné globální nastaevní
static=yes
writeprotect=yes        ;umožňuje ukládat dialplan z příkazové řádky
autofallthrough=yes
priorityjumping=no
clearglobalvars=no      ;určuje, jestli při příkazu reloadu číselného
plánu budou globální proměnné odstraněny

[globals]               ;globální proměnné, volají se pomocí $(Nazev)

[libovolny_kontext]      ;příkazy pro libovolny_kontext
exten = klapka,priorita,příkaz(parametry) ;povinná syntaxe
```

V rámci testování je obsahu souboru *extensions.conf* následující:

```
[general]
static=yes
writeprotect=yes
autofallthrough=yes
clearglobalvars=no
prioritytyjumping=no


[sip]
exten=>_10X,1,Dial(SIP/${EXTEN})


;Alternativa:
;exten=>102,1,Dial(SIP/${EXTEN})
;exten=>102,1,Dial(SIP/${EXTEN})
```

Kromě klapek, které si může vytvořit sám uživatel, existují klapky vyhrazené pro specifické účely [21]:

- i : Invalid – automaticky použitá klapka v případě klapky neplatné či nenalezené
- s : Start – klapka použitá při neznámém nebo neuvedeném volaném čísle
- h : Hangup – použita v případě ukončení hovoru volaným účastníkem
- t : Timeout – použita při vypršení časovače v určeném kontextu
- T : AbsoluteTimeout – použita při vypršení časovače AbsoluteTimeout
- a : Asterisk extension – při použití * v hlasové poště
- o: Operator – pro operátora při použití 0 v hlasové poště

Ve vytvořené konfiguraci je uvedena i alternativní zápis. V rámci testování je zvolena zjednodušená varianta zápisu, která má jeden řádek kódu. Lze ovšem zapsat i jinak. Zjednodušení je umožněno pomocí zástupných znaků [21]:

- X: jakékoliv číslo 0-9.
- Z: čísla 1-9.
- N: čísla 2-9.
- [125-9]: čísla uvedená v závorce, v tomto případě 1,2,5,6,7,8,9.
- . : jeden nebo více znaků.
- ! : žádný nebo více znaků.

Poté, co je konfigurace kontextu hotova, je potřeba znovu načíst dialplan. To je provedeno zadáním příkazu *dialplan reload* do konzole Asterisku. Nyní je možno realizovat telefonní hovor.

5.1.4 Realizace hovoru

Pokud jsou oba softwarové telefony registrovány u ústředny, je možno započít hovor v souladu s vytvořeným kontextem. Na Obr. 10 je ilustrován průběh inicializace spojení ze strany ústředny, Obr. 11 zprávu INVITE zachycenou programem Wireshark.

```
== Using SIP RTP CoS mark 5
  -- Executing [102@sip:1] Dial("SIP/101-0000003f", "SIP/102") in new stack
== Using SIP RTP CoS mark 5
  -- Called SIP/102
  -- SIP/102-00000040 is ringing
     > 0x55d484444dc0 -- Probation passed - setting RTP source address to 192.168.20.200:8000
  -- SIP/102-00000040 answered SIP/101-0000003f
  -- Channel SIP/102-00000040 joined 'simple_bridge' basic-bridge <7f69c388-0170-4c89-a108-221e30dcffb9>
  -- Channel SIP/101-0000003f joined 'simple_bridge' basic-bridge <7f69c388-0170-4c89-a108-221e30dcffb9>
     > Bridge 7f69c388-0170-4c89-a108-221e30dcffb9: switching from simple_bridge technology to native_rtp
     > Remotely bridged 'SIP/101-0000003f' and 'SIP/102-00000040' - media will flow directly between them
     > 0x7f2428019c20 -- Probation passed - setting RTP source address to 192.168.20.121:8000
  -- Channel SIP/102-00000040 left 'native_rtp' basic-bridge <7f69c388-0170-4c89-a108-221e30dcffb9>
  -- Channel SIP/101-0000003f left 'native_rtp' basic-bridge <7f69c388-0170-4c89-a108-221e30dcffb9>
== Spawn extension (sip, 102, 1) exited non-zero on 'SIP/101-0000003f'
```

Obr. 10: Reakce serveru na žádost o hovor

```
Session Initiation Protocol (INVITE)
  Request-Line: INVITE sip:101@192.168.20.121:40593;transport=UDP SIP/2.0
    Method: INVITE
    Request-URI: sip:101@192.168.20.121:40593;transport=UDP
    [Resent Packet: False]
  Message Header
    Via: SIP/2.0/UDP 192.168.20.199:5060;branch=z9hG4bk50aa17e3;rport
    Max-Forwards: 70
    From: <sip:102@192.168.20.199;transport=UDP>;tag=as7355579f
    To: <sip:101@192.168.20.199;transport=UDP>;tag=25065738
    Contact: <sip:102@192.168.20.199:5060>
    Call-ID: OK3wCCmPdQJLGvlr0ZyAQA..
    CSeq: 102 INVITE
    User-Agent: Asterisk PBX 14.6.0
    Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO, PUBLISH, MESSAGE
    Supported: replaces, timer
    Content-Type: application/sdp
    Content-Length: 252
  Message Body
    Session Description Protocol
      Session Description Protocol Version (v): 0
      Owner/Creator, Session Id (o): root 348508874 348508875 IN IP4 192.168.20.200
      Session Name (s): Asterisk PBX 14.6.0
      Connection Information (c): IN IP4 192.168.20.200
      Time Description, active time (t): 0 0
      Media Description, name and address (m): audio 8000 RTP/AVP 8 101
      Media Attribute (a): rtpmap:8 PCMA/8000
      Media Attribute (a): rtpmap:101 telephone-event/8000
      Media Attribute (a): fmtp:101 0-16
      Media Attribute (a): ptime:20
      Media Attribute (a): maxptime:150
```

Obr. 11: Obsah zprávy INVITE

Na Obr. 11 lze přehledně vidět obsah posílané zprávy INVITE, a to ze strany klapky 102 na 101. Jde vidět, že při přístupu do sítě není problém se zachycením takovýchto

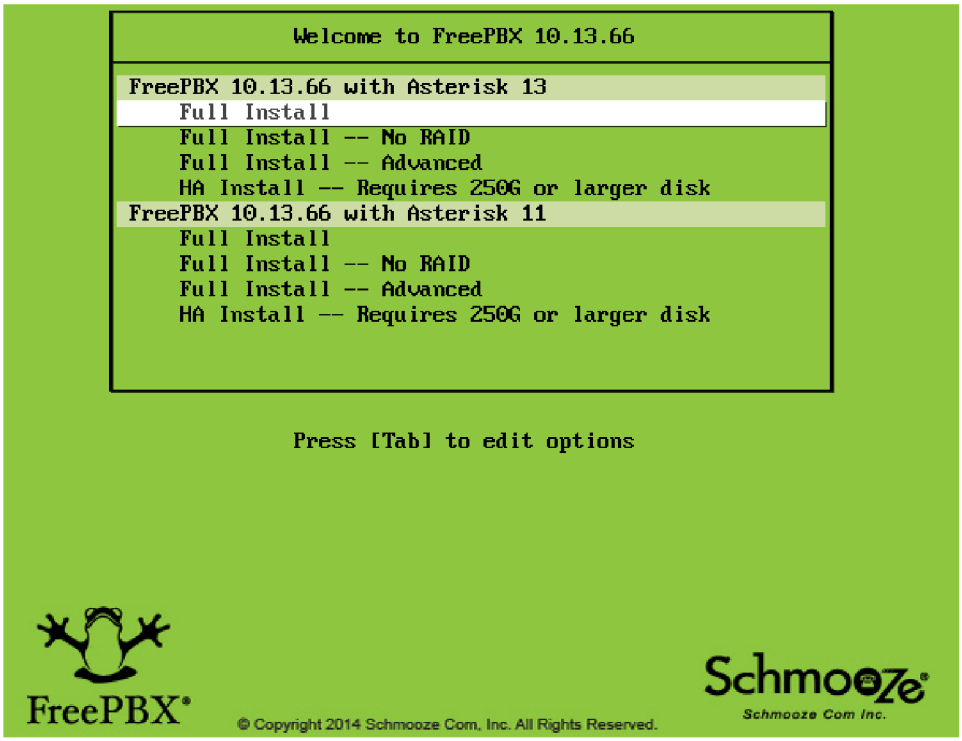
zpráv a jejich následném analyzování, neboť se zprávy posílají jako prostý text.

5.2 AsteriskNOW

První testovanou nástavbou je projekt společnosti Digium. Ústředna běží na systému SCHMZ 6.6 od společnosti Schmooze, který je založen na operačním systému CentOS. Tato ústředna je vybavena grafickým rozhraním FreePBX. O projekt se donedávna starala, byť neoficiálně, společnost Schmooze, nicméně aktuálně je tato společnost pod vlastnictvím společnosti Sangoma. Právě proto oba projekty (AsteriskNOW a FreePBX) už téměř splývají. Hlavním rozdílem jejich nejnovějších verzích je právě operační systém, resp. jeho úprava, na kterém běží. Dalším je podpora. K nástavbě AsteriskNOW vychází aktualizace jen velmi zřídka, neboť společnost Sangoma v první řadě všechna vylepšení aplikuje do svého hlavního projektu FreePBX.

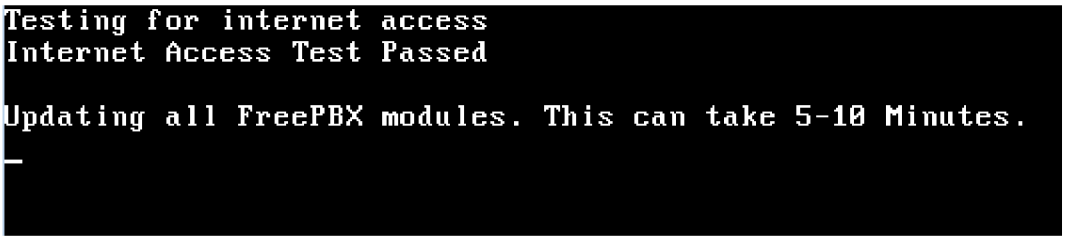
5.2.1 Instalace

Prvním krokem je stažení obrazu instalačního disku. Ten je volně k dispozici na oficiálních stránkách projektu (https://www.asterisk.org/downloads/asterisknow). Aktuální verze je 10.13.66-17, která nabízí verzi Asterisku 11 či 13. V rámci testování je zde zvolena novější verze 13.



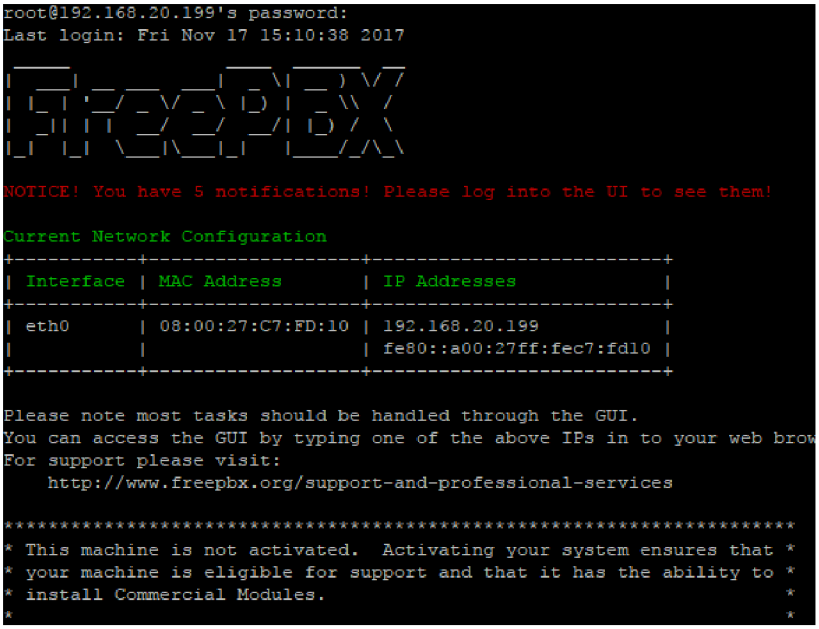
Obr. 12: Úvodní obrazovka instalačního programu AsteriskNOW

Z Obr. 12 je patrné, že AsteriskNOW je ve skutečnosti FreePBX, který pouze běží na jiném systému. Instalační průvodce na úvod vyžaduje nastavení síťové karty, resp. statickou IP adresu, nebo dynamicky přidělenou DHCP serverem.



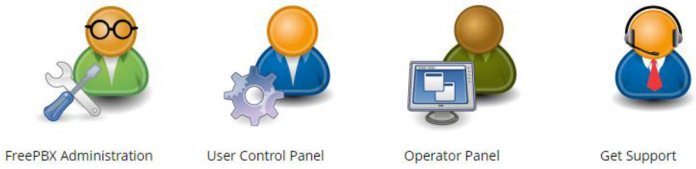
Obr. 13: Aktualizace před prvním spuštěním AsteriskNOW.

Instalace pak již probíhá automaticky a nevyžaduje další zásahy. Po jejím dokončení je systém restartován a následně je zkontrolována dostupnost internetového připojení a jsou aktualizovány všechny dostupné moduly. Co se týče instalace, je nutno dodat, že je značně nespolehlivá. Instalace se několikrát přerušila z neznámého důvodu, a bylo třeba ji opakovat. Následný proces aktualizace modulů také zabral místo avizovaného časového rozpětí 5-10 minut zhruba o 60 minut více, což mohlo být způsobeno také testovací stanicí, nicméně podobné problémy zmiňují i jiní uživatelé na rozličných internetových diskusích týkající se projektu.



Obr. 14: Obrazovka po spuštění.

Po aktualizaci všech modulů je možné se do ústředny přihlásit. Pro první spuštění viz Obr. 14. Na rozdíl od čisté instalace, je zde barevně přehled síťových rozhraní s MAC adresou a příslušnou IP adresou. Také je zde doporučení, aby uživatel veškeré operace dělal skrze webové rozhraní (GUI). Je zde možnost aktivace systému, která zpřístupní instalaci komerčních modulů. Grafické rozhraní je k dispozici přes webový prohlížeč na adrese 192.168.20.199.



Obr. 15: Grafické rozhraní FreePBX u AsteriskNOW

Po zadání příslušné IP adresy do webového prohlížeče je uživatel vyzván k zadání přístupového jména a hesla. Posléze jsou k dispozici čtyři možnosti, viz Obr. 15. Pro administraci ústředny AsteriskNOW je potřeba zvolit možnost první – FreePBX Administration. Zde proběhne úvodní konfigurace. Lze zde vybrat jazyk prostředí, hlášek, a interní časové nastavení. AsteriskNOW nabízí českou lokalizaci zvukových hlášek.

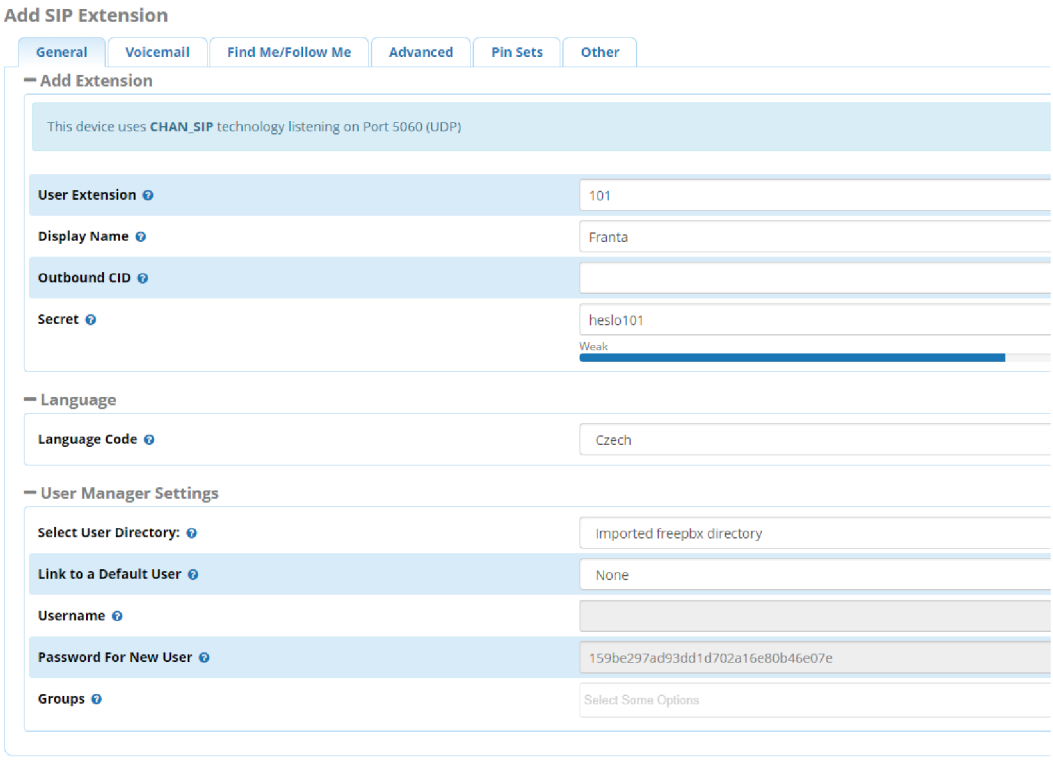
Místo klasického systémového firewallu zde společnost Sangoma implementovala svůj proprietární *Sangoma Smart Firewall*, který lze v několika krocích nastavit.

5.2.2 Konfigurace SIP účtů

Před vytvořením klapek SIP je nutno pozměnit konfiguraci portů. AsteriskNOW, resp. FreePBX podporuje jak protokol SIP, tak i PJSIP. Ve výchozím nastavení je protokolu SIP přiřazeno port 5160 místo portu 5060 tak, jak je zvykem. Port 5060 je naopak určen protokolu PJSIP. V záložce *Settings/Asterisk SIP Settings* je tedy potřeba tyto porty prohodit.

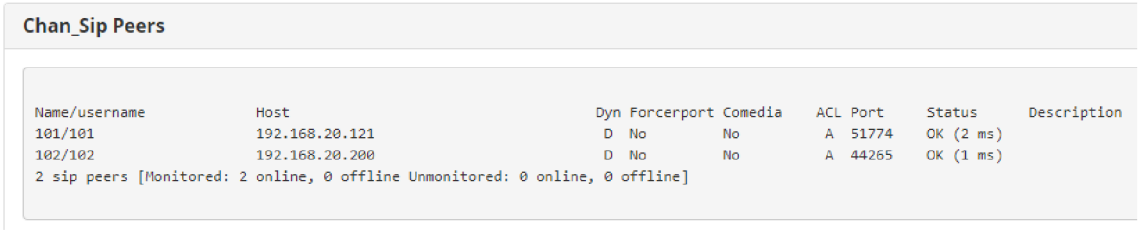
Přidání nového uživatele probíhá *Chan_SIP Extensions* dostupné v přes záložku *Applications/Extensions*. Po kliknutí na tlačítko *Add new CHAN_SIP Extension* se objeví formulář, viz Obr. 16. Je zde rovněž potvrzena správná konfigurace portu. V dodatečných záložkách jdou nastavit podrobnější nastavení, jako je třeba kontext.

AsteriskNOW má ve výchozím nastavení předkonfigurován kontext *from-internal*, do kterého jsou klapky automaticky přidávány.



Obr. 16: Vytvoření klapky v grafickém rozhraní

Poté, co je vyplněna veškerá nutná konfigurace se přidání potvrdí tlačítkem *Submit*. Nutno dodat, že jakákoliv změna se do Asterisku promítne až po znovu načtení veškerých konfiguračních nastavení. To je v grafickém rozhraní FreePBX provedeno tlačítkem *Apply Config* v pravém horním rohu. Obdobně se přidá i druhý účastník. Stav registrace lze ověřit v záložce *Reports/Asterisk Info* ().



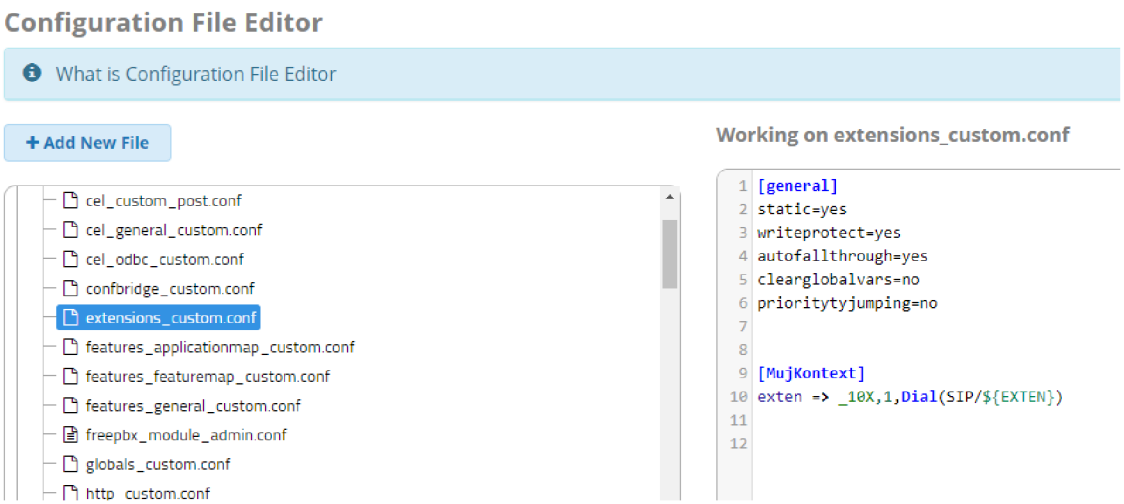
Obr. 17: Výpis registrovaných peerů

5.2.3 Konfigurace kontextu

Jak již bylo řečeno v předchozí kapitole, AsteriskNOW má v sobě předemvytvořený kontext *from-internal*. FreePBX rozhraní je postaveno tak, aby uživatel nemusel zasahovat do vnitřních souborů přes příkazovou řádku, ale měnil konfiguraci pouze přes grafické rozhraní. FreePBX ovšem v novějších verzích neumožňuje tvorbu vlastních kontextů jednoduchým způsobem. Nabízí naopak kontexty své – předem definované.

Prvním krokem je povolení vlastních kontextů. Tím dosáhneme toho, že ústředna při dalším znovunačtení konfiguračních souborů bude brát v potaz i kontexty jiné, než předdefinované. Toto je provedeno v záložce *Settings/Advanced Settings*. Položka *Disable -custom Context Includes* je defaultně nastavena na hodnotu *Yes*. Změníme ji tedy na *No*. Nejen kontexty, ale i ostatní konfigurační soubory jsou k dispozici k nahlédnutí přes webové rozhraní v záložce *Admin/Config Edit*. V každém takovém konfiguračním souboru je ale způsobem komentáře umístěno varování, že by do něj nemělo být zasahováno tímto způsobem, nýbrž přes nástroje grafického rozhraní. V paletě je třeba najít konfigurační soubor *extensions_custom.conf*, které je možné upravit v pravé části, viz Obr. 18. Tento soubor je automaticky načítán do hlavního souboru *extensions.conf*. Pokud bychom vytvořili úplně nový soubor (např. *moje_kontexty*), bylo by potřeba do tohoto hlavního konfiguračního souboru ho přidat pomocí následující řádky kódu:

```
#include moje_kontexty.conf
```



Obr. 18: Konfigurace vlastního kontextu

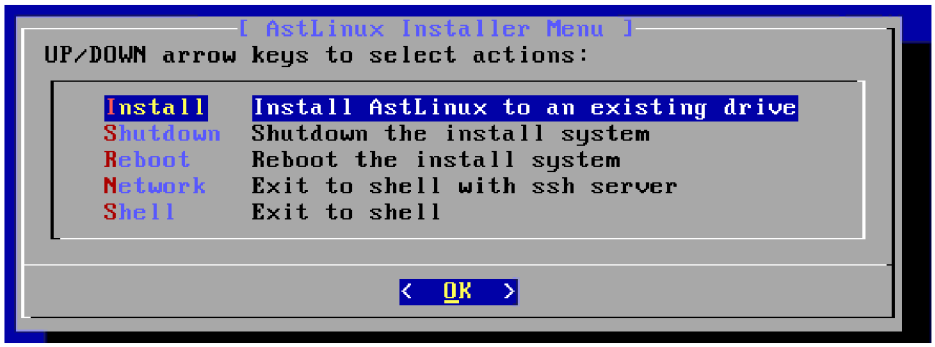
Jakmile je kontext hotov, je nutno spustit znovunačtení konfiguračních nastavení. Poté je možné provést hovor. Výpis Wiresharku nebude přiložen, neboť SIP zprávy se posílají ve stejném sledu, jako u čisté instalace Asterisku, jsou totiž definovány doporučením RFC.

5.3 AstLinux

AstLinux je mezi testovanými konfigurátory svým způsobem jedinečný z mnoha důvodů. Jedním z nich je jeho jednoduchost. Samotný instalační disk má oproti ostatním testovaným objektům cca desetkrát menší velikost. Jedná se o vhodné řešení pro různorodé komerční vestavěná (embedded) řešení, neboť je vyvíjen za tím účelem, aby byl kompaktní a mohl být spouštěn například z paměti flash. Může být ovšem instalován i na regulérní pevné disky.

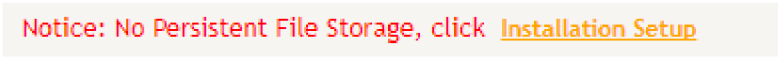
5.3.1 Instalace

Instalace proběhla ve verzi 1.3.1 a obsahuje Asterisk 13. Instalační program nenabízí téměř žádné možnosti. Po spuštění obrazu disku a následné instalace (Obr. 19) je uživatel dotázán na disk, který chce použít k instalaci. Poté mu již žádné další předvolby nabídnuty nejsou.



Obr. 19: Instalační průvodce AstLinux

Nutno podotknout, že instalační proces trval necelých dvacet vteřin, načež je ústředna po restartu systému připravena k použití. Ke konfiguraci ústředny – ať už přes webové rozhraní nebo příkazovou řádku – je potřeba se přihlásit. Uživatel si bohužel během instalace nemohl definovat své vlastní přihlašovací údaje a je tedy nutno použít defaultní; *root* s heslem *astlinux*. V tuto chvíli je ovšem vše nainstalováno pouze dočasně. AstLinux potřebuje trvalé úložiště, do kterého by mohl zapisovat. O tomto faktu je uživatel zpraven na úvodní obrazovce webového rozhraní, viz Obr. 20.

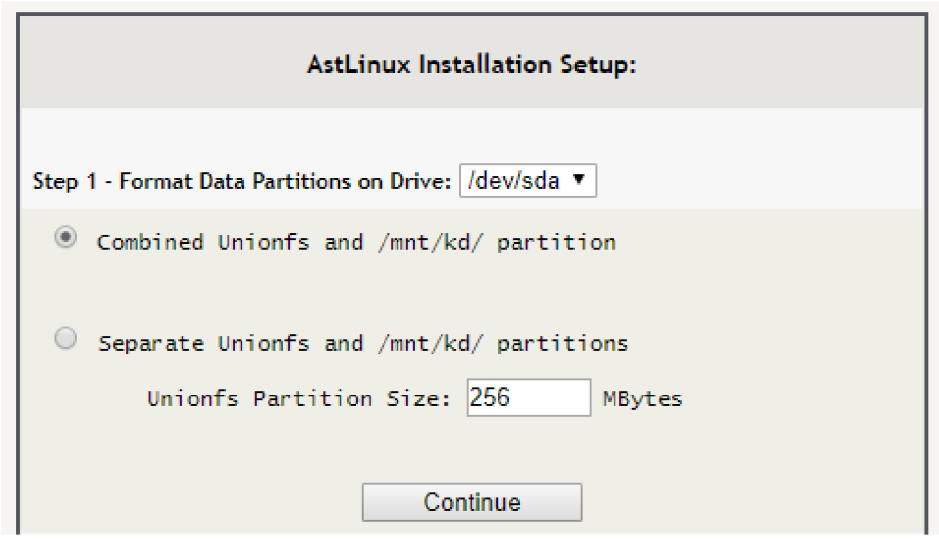


Obr. 20: Oznámení o nutnosti trvalého úložiště

Obsah úložiště se u AstLinuxu dělí na dvě kategorie [3]:

- Data System Overlay (Uniofs) – hesla, moduly spouštěné při startu aj.,
- Konfigurační data (/mnt/kd) – konfigurační soubory Asterisku, systémová nastavení a hlasové zprávy

Pro nové instalace je výchozím doporučovaným nastavením kombinace těchto dvou kategorií na jednu diskovou jednotku.



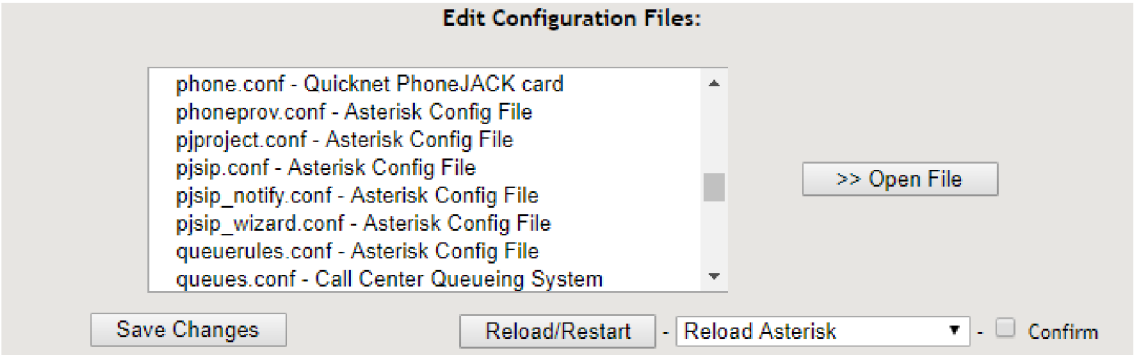
Obr. 21: Nastavení diskových oddílů

Jak je vyobrazeno na Obr. 21, jednou z možností je také rozdělení obou zmíněných souborových kategorií na dva oddíly. V rámci testování je ale použit kombinovaný oddíl. Po rozdělení je celý systém restartován a uživatel je informován o úspěchu.

5.3.2 Konfigurace SIP účtů

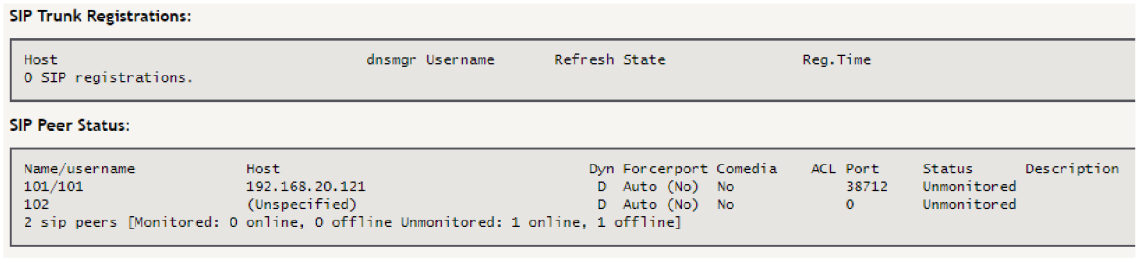
Konfigurace uživatelských účtů SIP, resp. klapek doposud u obou nástaveb probíhala vyplněním formuláře ve webovém rozhraní. AstLinux zde vybočuje. V grafickém rozhraní lze nastavit především systémová nastavení, tj. např. SMTP relay, NTP servery, ACME, DNScrypt a další. Jedná se o protokoly vhodné pro implementaci v embedded systémech.

Klapky lze přidat v záložce *Edit*. Zde ovšem není pouhá registrace účtů, nýbrž tato sekce funguje jako pomyslný textový editor v grafickém rozhraní. Je zde seznam všech konfiguračních souborů Asterisku, do kterých lze v této sekci přistupovat a upravovat je. Jedná se o manuální konfiguraci, a je zde tedy potřeba znalost syntaxe a příkazů. Přidání SIP účtu tedy proběhne standardním způsobem; je třeba si otevřít konfigurační soubor *sip.conf* a přidat účty tam, viz Obr. 22. Poté je změny možné pomocí tlačítka *Save changes* uložit a následně provést *Reload* Asterisku (také pomocí tlačítka).



Obr. 22: Grafické prostředí pro modifikaci konfigurace AstLinux

Stav účastníků a jejich registrace lze ověřit pomocí webového rozhraní v tabulce SIP Peer Status. AstLinux nabízí i přehled registrovaných trunků (Obr. 23).



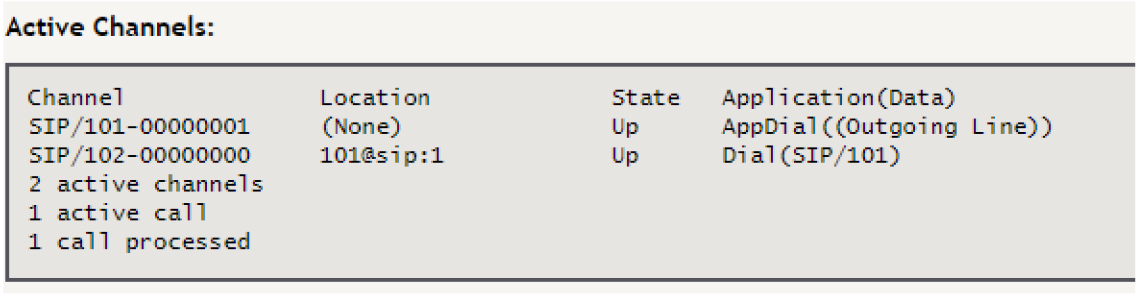
Obr. 23: Přehled registrací

Pro realizování hovoru je nyní nutné nakonfigurovat kontext.

5.3.3 Konfigurace kontextu

Proces konfigurace kontextu je velmi podobný vytváření SIP účtů. Webové prostředí AstLinuxu obecně nabízí velmi malé množství grafické konfigurace samotného Asterisku. Prostředí slouží spíše pro obecná nastavení sítě a operačního systému jako takového, a také jako monitoring provozu a stavu ústředny.

Konfigurace kontextu je proveden v záložce *Edit*, kde je potřeba vybrat soubor *extensions.conf*. Po dokončení zápisu je nutné tlačítkem *Reload* provést jeho znovunačtení.



Obr. 24: Monitoring aktivních komunikačních kanálů

Zmíněný monitoring ústředny, konkrétně monitoring aktivních komunikačních kanálů, resp. probíhajících hovorů nabízí tabulka *Active Channels*.

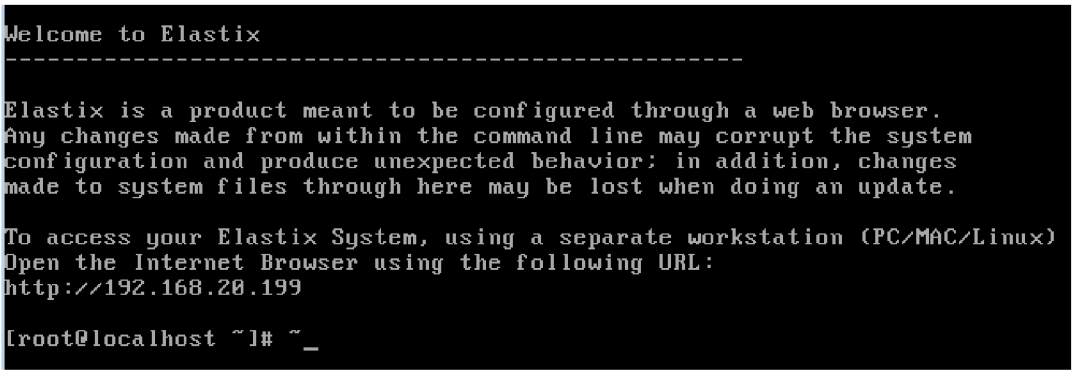
5.4 Elastix

Dalším testovaným objektem je nástavba Elastix. Na rozdíl od AstLinuxu či AsteriskNOW se jedná o unifikovanou komunikační platformu, tedy ne pouze pobočkovou ústřednu. Sjednocuje funkce IP PBX, E-mailu, Fax a IM serveru. Jeho webové rozhraní oplývá možnostmi, kterými disponují například softwary pro call centra.

5.4.1 Instalace

V rámci testování byla zvolena verze Elastixu 2.5.0 (stable). Jedná se o poslední verzi, která je založena na ústředně Asterisk. Pozdější verze obsahují ústřednu 3CX. Tato nástavba využívá modifikovaného systému CentOS.

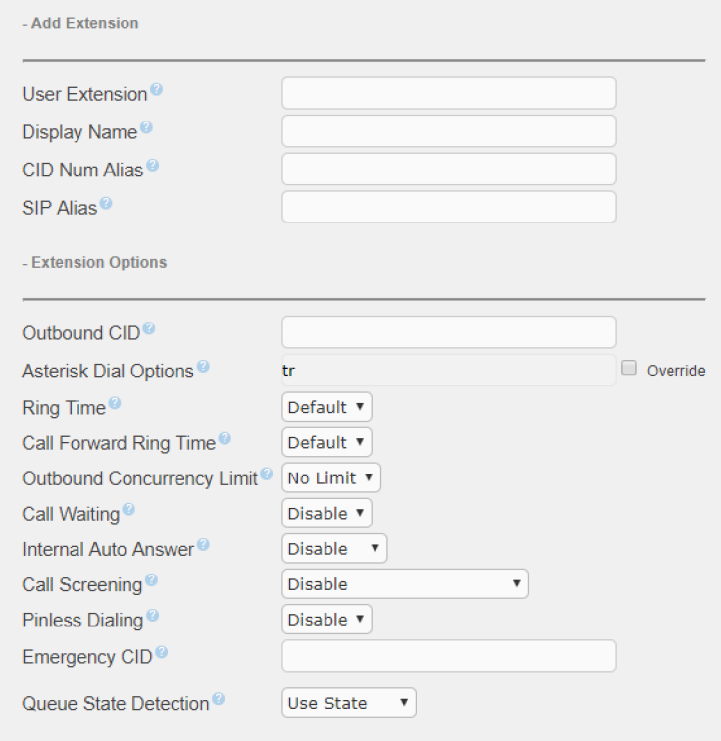
Instalační průvodce nabízí dva režimy instalace – v textovém režimu a v režimu grafickém. Je možné zvolit českou znakovou sadu, a i samotná instalace je v českém jazyce. Instalace dále nabízí možnost automatické konfigurace síťového rozhraní pomocí DHCP, nebo konfiguraci manuální. Následuje rozdělení diskového úložiště a poté instalace. Instalační proces zabral zhruba 5 minut. Po dokončení procesu a následném přihlášení je uživatel vyzván k tomu, aby veškerou úpravu konfigurace prováděl přes webové rozhraní, jinak by mohlo dojít k nestabilitě a poškození systému.



Obr. 25: Úvodní obrazovka Elastix

5.4.2 Konfigurace SIP účtů

Na rozdíl od nástavby AstLinux, zde se přidávání účtů provádí opět pomocí webového formuláře. Sekci se SIP účty lze najít v záložce *PBX/PBX Configuration/Extensions*. Po kliknutí na tlačítko *Add extension* je možné vybrat typ účtu, neboť kromě SIP lze zvolit i IAX nebo DAHDI účet.



Obr. 26: Přidání SIP účtu

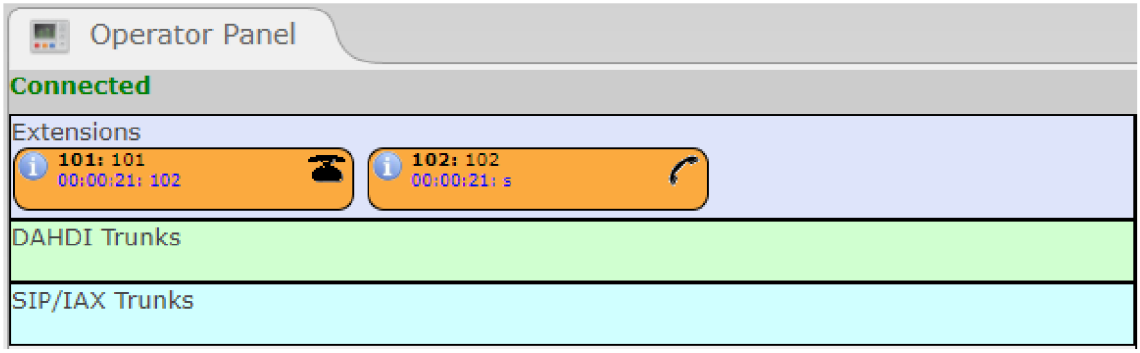
Jak lze vidět na Obr. 26, rozhraní nabízí několik různých možností nastavení SIP účtu. Ačkoliv Elastix nabízí, co se týče počtu těchto položek v nastavení o něco méně, než-li třeba FreePBX (AsteriskNOW), je nutno zmínit, že umožňuje mimo jiné i třeba tzv. Pinless Dialing. Jedná se o funkci, kdy není od konkrétního uživatele vyžadováno zadání PINu (série čísel) při volání na určitou klapku a může tedy volat napřímo.

5.4.3 Konfigurace kontextu

Stejně jako u FreePBX, Elastix obsahuje defaultně předem nakonfigurovaný kontext *from-internal*. Ten umožňuje volání v rámci ústředny, tedy vnitřní. Vlastní pravidla pro vytáčení lze nastavit v záložce *PBX Configuration/Inbound Call Control/Inbound Routes*. Stejným způsobem je možno nastavit i pravidla pro hovory, které jsou směrovány mimo ústředny, tedy externí, a to v sekci *Outbound Routes*.

Alternativně lze samozřejmě veškerou konfiguraci provést metodou manuální úpravy konfiguračního souboru *extensions.conf* v záložce *PBX/Tools/Asterisk File Editor*. Doporučuje se ale metoda konfigurace pomocí grafického rozhraní.

Po dokončení konfigurace SIP účtů a kontextu lze zrealizovat hovor. Elastix v tomto ohledu přináší unikátní monitorovací nástroj – panel operátora, viz Obr. 27.



Obr. 27: Panel operátora

Zde lze v reálném čase pozorovat aktuálně registrované klapky a zároveň u nich i zobrazit aktivní komunikační kanály. Pomocí tohoto panelu lze hovor také zavěsit.

5.5 FreePBX

Poslední testovanou nástavbou je projekt FreePBX. Jedná se o nejpoužívanější nástavbu klasického Asterisku, jenž je pod správou telekomunikační společnosti Sangoma.

FreePBX své webové rozhraní poskytuje i dalším projektům jako je právě třeba AsteriskNOW. Souvislost mezi těmito dvěma projekty je nemalá. Jak již bylo zmíněno v kapitole 2.6.2, původně se o projekt AsteriskNOW starala společnost Schmooze, nicméně nyní jsou oba projekty pod správou společnosti Sangoma. V popředí je ovšem projekt FreePBX, neboť obsahuje novější verzi operačního systému (založeného na CentOS) s názvem SangomaOS. Obsahuje také novější verzi Asterisku (15). Kromě operačního systému a verze Asterisku jsou ale oba projekty, co se týče možností webového rozhraní, totožné. V závislosti na jinou verzi Asterisku a také jiného operačního systému je ale možné, že se ústředna bude za určitých okolností chovat jinak. Proto proběhne testování také i zde. U nástavby FreePBX jsou zejména kvůli stejnému postupu jako u AsteriskNOW vynechány kroky demonstrující instalaci, konfiguraci SIP účtů a kontextu.

6 ÚTOKY NA PROTOKOL SIP

V rámci diplomové práce, budou veškeré testované konfigurátory (zmíněné v kapitole 4.1) podrobeny útokům na protokol SIP. Princip těchto útoků a předpokládané chování ústředny dle doporučení RFC 4475 [16] jsou vysvětleny v kapitole 3.2.

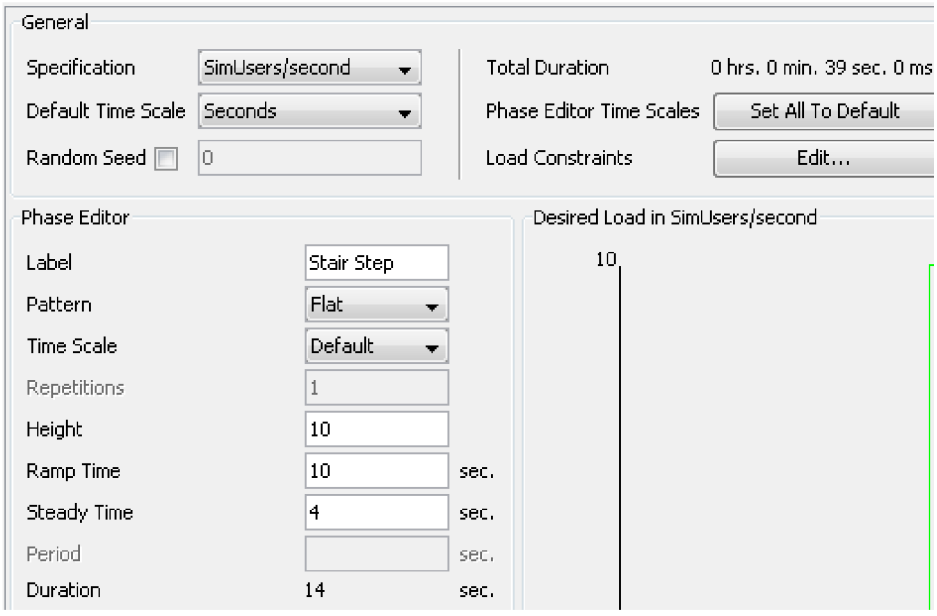
Útoky jsou provedeny pomocí testovacího systému Spirent Avalanche. Je třeba dané scénáře vytvořit v obslužném konfiguračním programu, kde jsou také navrženy jednotlivé útoky pomocí nástroje *Attack Designer*, jenž je v tomto obslužném programu obsažen.

Na úvod je popsáno obecné nastavení testovacího systému, které je následováno konfiguracemi jednotlivých útoků. Výstup útoků je prezentován v tabulce Tab. 8 v kapitole 8.5. V této tabulce jsou uvedeny odezvy ústředny pro každou z testovaných ústředen.

6.1 Všeobecné nastavení testovacích scénářů

Kromě nastavení, které je specifické pro každý z konkrétních útoků, je nutné nastavit i obecné parametry, které platné pro všechny scénáře na testeru Spirent Avalanche.

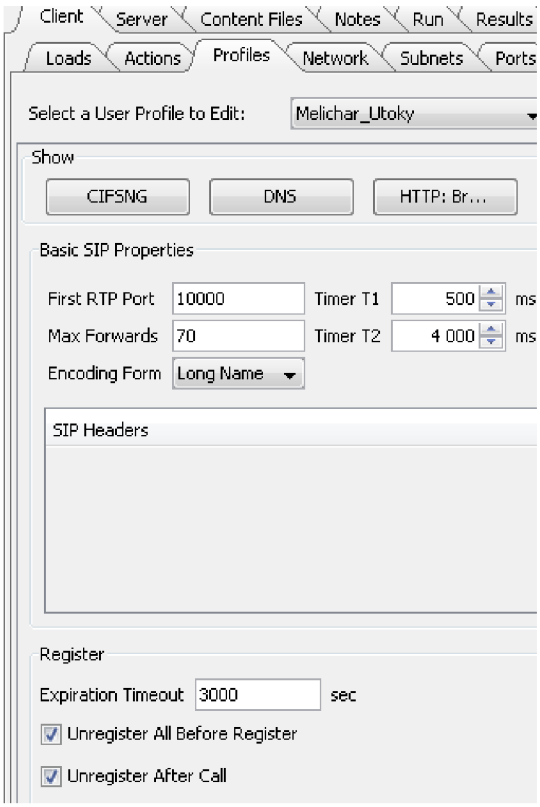
Nastavení testovacího systému se dělí na dvě části. Nastavuje se část klientská v záložce *Client*, a část serverová v záložce *Server*. Konfigurace začíná částí klientskou. Jako první je třeba nastavit zátěž, jakou tester má generovat. Je zvoleno maximum deseti uživatelů za vteřinu. Každé nastavení zátěže je rozděleno na tři části. Část náběhu, část špičky, a část klesající. Zátěž v každé této části, resp. počet hovorů v těchto částech se nastavuje pomocí jednotek *SimUsers,* které symbolizují simulované účastníky. Pro detail nastavení zátěže viz Obr. 28.



Obr. 28: Detail nastavení zátěže testeru

Testu je třeba také definovat časové rozpětí. Na obrázku výše jsou viditelné jednotlivé části testu, a to *Ramp Time* a *Steady Time*. Celková délka testu je nastavena na 39 vteřin.

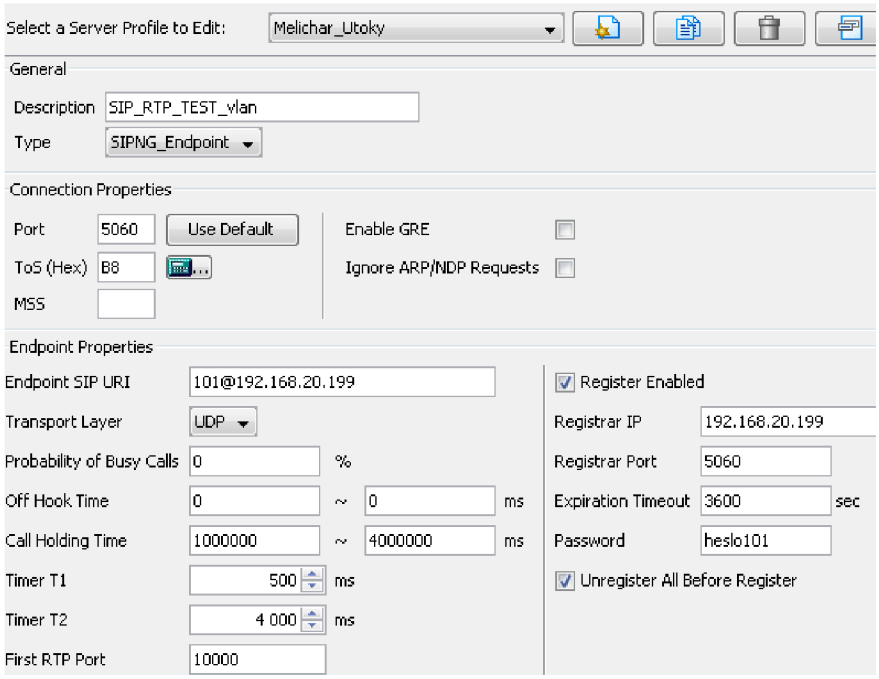
Další položkou, kterou je nutné nastavit, je sekce *Profiles*.



Obr. 29: Nastavení sekce *Profiles* - Klient

V této záložce (viz Obr. 29) se mimo portů SIP nastavuje také rozsah portů RTP. Hlavní pozornost je ale třeba věnovat části *Register*. Tato část udává dobu, po které vyprší registrace SIP účtu u ústředny. Dále jsou zde dvě položky, které zajišťují, že v době spuštění testu nebudou používané účty registrovány. Obě tato okénka je potřeba zaškrtnout. Posledním nastavením v klientské části konfigurace je nastavení portů testeru v závislosti na vnější síti, které ovšem nebude zveřejňováno z bezpečnostních důvodů. Nutno podotknou, že ve veškerých záložkách je nutno zvolit ten správný uživatelský profil, v mém případě *Melichar_Utoky*.

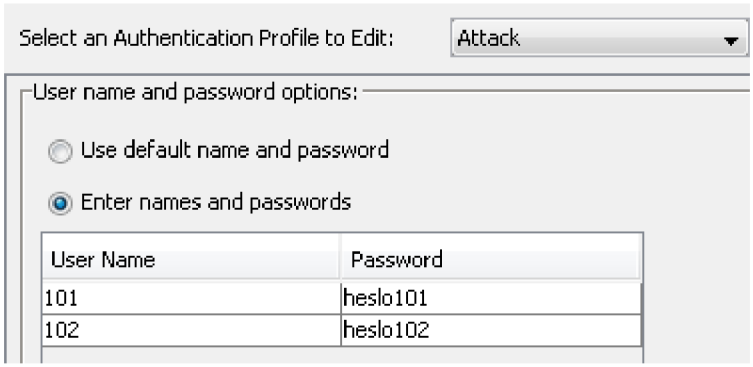
Pokračováním je nastavení serverové části testeru. Jako první je nutno nastavit hlavní profil, tedy *Melichar_Utoky*, což se provádí v sekci *Profiles*, viz Obr. 30: Nastavení sekce .



Obr. 30: Nastavení sekce *Profiles* – Server

V této sekci je nutné nastavit správný port, na kterém funguje SIP (5060), SIP URI účastníka, jehož účet je nastaven v konfiguračním soubor *sip.conf* v ústředně, v pravé části pak adresu Registrar serveru a heslo onoho účastníka, které je vyžadováno při registraci. Mimo Obr. 30 je ještě pole *Media Properties*, kde se definuje, jaký kodek a obsah se bude při hovoru přenášet. Pro účel testován je zvolen sample soubor s kodekem G711.a, který ústředna podporuje.

Při simulování hovorů je nezbytné, aby se tester generoval platné SIP zprávy, tedy měl platné přihlašovací údaje. Tyto údaje jsou opět v konfiguračním souboru ústředny *sip.conf*. Na Obr. 31 lze vidět definované klapky pod vytvořeným autentizačním profilem *Attack*.



Obr. 31: Nastavení sekce *Authentications*

Tímto je provedeno obecné nastavení testeru. Následuje popis konfigurace testovacího systému pro jednotlivé útoky.

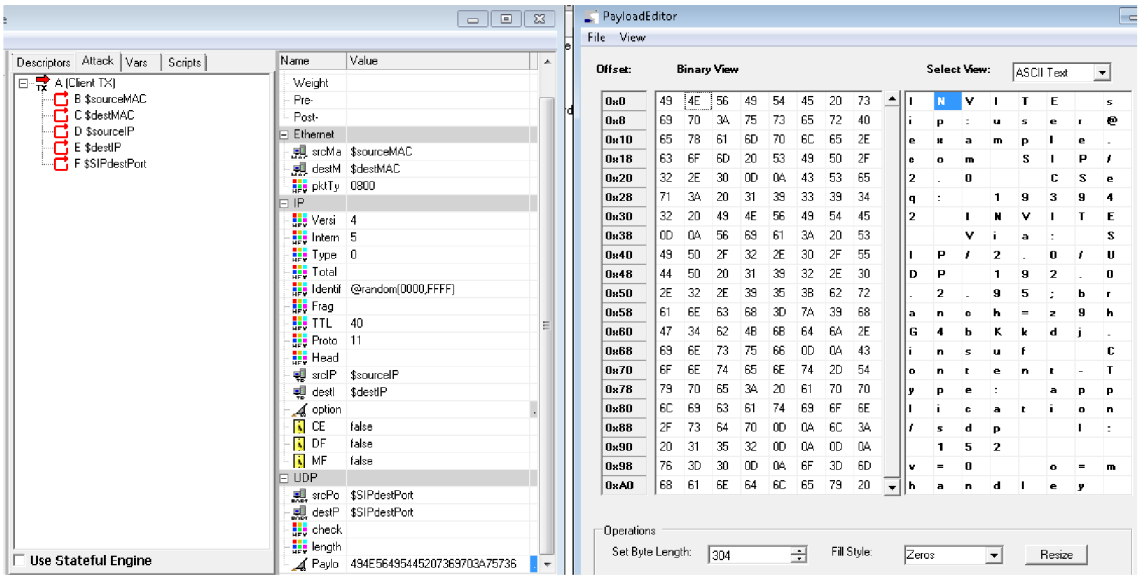
6.2 Vybrané modifikované útoky

6.2.1 Missing Required Headers

Tento typ útoku posílá zprávu SIP INVITE, která je neúplná. Jak bylo zmíněno v kapitole 3.2.1, je zde absence polí *Call-ID*, *From* a *To*.

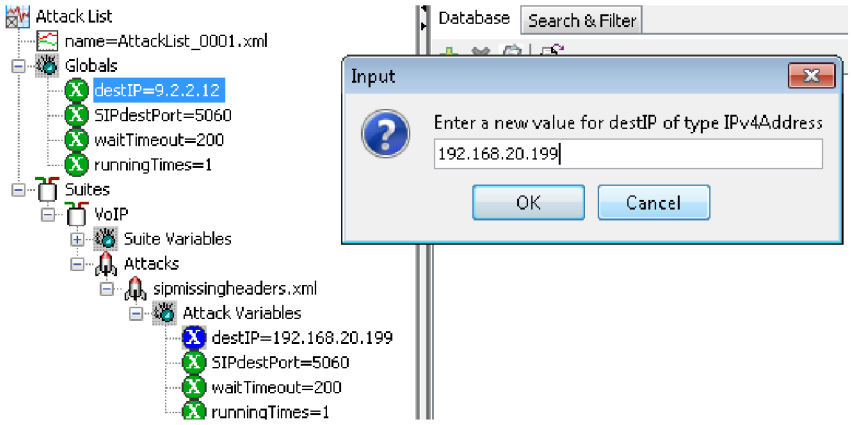
Nastavení konkrétního útoku probíhá v klientské části konfigurace. V záložce *Actions* je potřeba zvolit vytvořený profil *Melichar_Utoky*. Dalším krokem je vytvoření tzv. *Attack Listu*, který lze pojmenovat libovolně. Vytvořený Attack List je nutno otevřít v editoru *Attack List Editor*, který se spouští tlačítkem *Edit*.

Z databáze se poté vybere šablon s názvem *sipmissingheaders.xml*, kterou je nutné editovat pomocí nástroje *Attack Designer*, který se spouští tlačítkem *Attack Designer* v okně dané šablony. Prostředí tohoto nástroje je vyobrazeno na Obr. 32.



Obr. 32: Editační prostředí nástroje Attack Designer

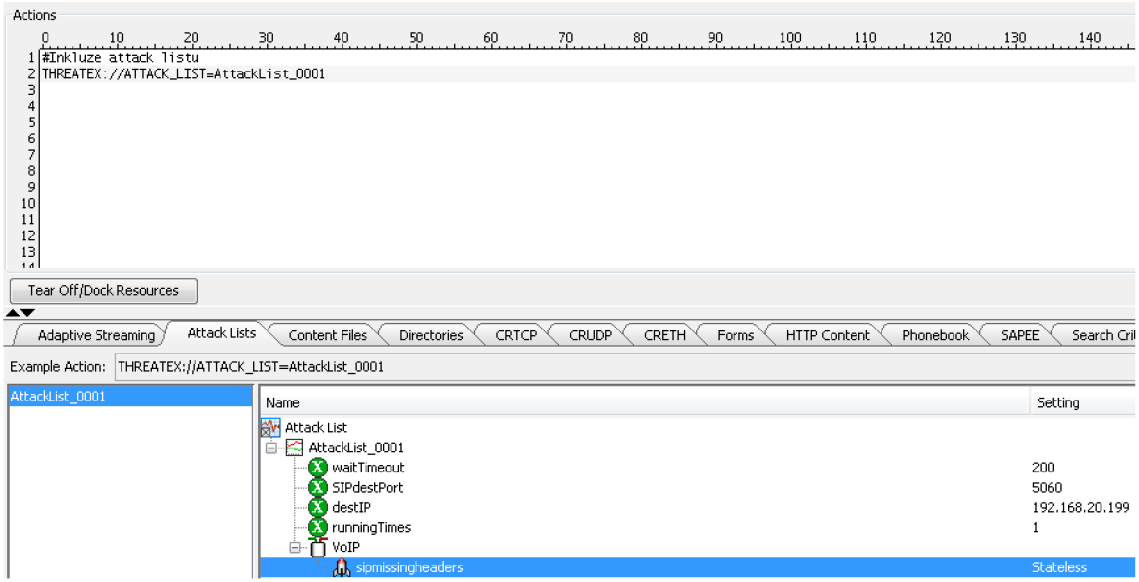
Na pravé části obrázku jde vidět posílaná zpráva znak po znaku. Právě zde je možné šablonu editovat, znak po znaku. Obrázek demonstruje fakt, že zmíněná pole ve zprávě chybí. Dále je třeba nastavit cíl útoku, čímž je ústředna (viz Obr. 33).



Obr. 33: Nastavení cílové adresy útoku

Po dokončení modifikace útoku je potřeba nastavit cíl útoku pomocí IP adresy v poli *destIP* na korektní adresu ústředny spolu s portem, na kterém naslouchá SIP.

Zbývá zahrnout vytvořený útok, resp. útokovou sadu (Attack List) do aktuálně prováděného testu. V záložce *Action* se nachází stejnojmenné pole, do které je nutné vložit příkaz, který toto provede. Finální nastavení tohoto testu je vyobrazeno na Obr. 34.



Obr. 34: Zahrnutí Attack listu do testu

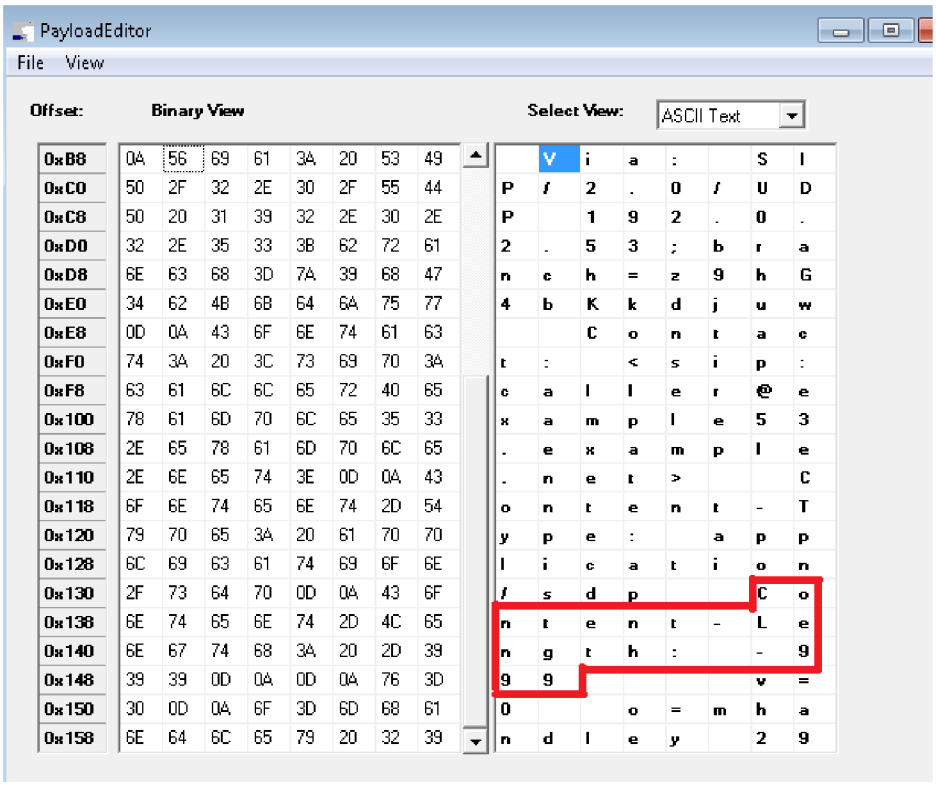
Nyní je možné provést útok.

6.2.2 Negative Content Length

Tento typ útoku využívá opět SIP zprávy INVITE. V tomto případě má zpráva všechny náležitosti, ale jeden parametr má neplatný. Tímto parametrem je Content-Length. V tomto poli je záporná hodnota, což není v souladu s doporučením RFC 4475 a jedná se tedy o hodnotu neplatnou. Dle tohoto doporučení by se ústředna měla chovat totožně,

pokud do stejného pole vložíme místo hodnoty negativní hodnotu nečíselnou.

Konfigurace se provádí stejně, jako tomu bylo v předešlém případě. V záložce *Actions* je potřeba kliknout na *Edit* a vybrat šablonu útoku. Vyžadovanou hodnotu *Content-Length* je možné nastavit pomocí nástroje *Attack Designer*. Zde je možné editovat *Payload* neboli obsah zprávy.



Obr. 35: Nastavení záporné hodnoty Content-Length

Na Obr. 35 lze vidět, že v poli *Content-Length* je negativní hodnota, konkrétně „−999“. Právě tato hodnota by se měla zobrazit i v logu ústředny při příjmu zprávy.

Zbývá pouze vložit vytvořený útok do *Attack Listu* a ten zahrnout do konkrétního testu, obdobně, jako tomu bylo v kapitole 6.2.

6.2.3 RFC 2543 INVITE

Dalším podnětem testování je metoda, kdy se opět posílá zpráva INVITE, nicméně dle syntaxe starého doporučení RFC 2543. Jedná se o předchozí verzi doporučení RFC 3261. Taková zpráva je zcela plnohodnotná, a tudíž SIP elementy, které si chtějí zachovat zpětnou kompatibilitu by měly takovou zprávu akceptovat.

Jak bylo zmíněno v kapitole 3.2.3, v této zprávě zcela chybí dvě pole, a jedno není explicitně definováno. V tomto testu bude zkoumáno chování ústředny při přijmutí takové zprávy.

Nastavení probíhá opět v záložce *Actions*, kde pomocí tlačítka *Edit* lze nástrojem *Attack Designer* zobrazit a modifikovat obsah zprávy.

7 ZÁSUVNÝ MODUL PRO PBX ASTERISK

Jedním z mnoha odvětví, kterým je potřeba porozumět za účelem pochopení podstaty a principu Asterisku jsou zásuvné moduly. Jak již bylo několikrát v tomto textu zmíněno, Asterisk je ústředna založená na principu modularity. Dle Tab. 6 je ovšem zřejmé, že jednotlivé konfigurátory obsahují různé verze Asterisku [6]. Moduly, které obsahují různé verze ústředny mohou mít na různých verzích odlišný zdrojový kód a mohou například implementovat makra, která jiná verze nepodporuje. Právě tato kompatibilita bude také zkoumána v rámci této kapitoly.

7.1 Úvodní informace

Jednou z mnoha věcí, které si práce klade za cíl, je sestrojení vlastního funkčního zásuvného modulu. V kapitole 6 jsou popsány útoky, kterým bude ústředna podrobena. Co se týče vlastního zásuvného modulu, konkrétním cílem je zde naprogramování takového modulu, který by dokázal přijímat pakety, resp. SIP zprávy, od koncových stanic (uživatelů) a dokázal s nimi pracovat. Snahou je tento datový tok analyzovat, tedy vyčíst z něj hodnoty a tato konkrétní data poté vyhodnotit.

Doporučení RFC 3261 velmi jasně specifikuje parametry, které musí SIP zpráva obsahovat, aby se dala považovat za validní. Zásuvný modul tedy musí být navržen tak, aby jednotlivá pole zprávy SIP dokázal nejen detekovat, ale v některých případech i analyzovat a na výstupu vypsat informace o každém zachyceném a analyzovaném paketu do příkazové řádky ústředny Asterisk, případně do libovolného logu. Jedná se ovšem o programování na úrovní jádra Asterisku a proto je zde nutné postupovat obezřetně, neboť by například při neoprávněném resp. nesprávném zásahu do paměti nebo při změně některé proměnné mohlo dojít k nechtěnému zasažení do funkčnosti jiných modulů nebo částí Asterisku.

Z důvodů, které jsou popsány v následující kapitole, bylo nutné vývoj modulu přizpůsobit i na jiný nežli výchozí stack Asterisku (chan_sip). Výstupem tedy budou dva různé soubory.

7.2 Modul pracující se stackem chan_sip

V tomto textu jsou popsány oba SIP stacky, které jsou podporovány ústřednou Asterisk. Jak bylo zmíněno, chan_sip je stackem výchozím a od kanálového ovladače PJSIP se liší tím, že veškerá jeho funkčnost je obsažena do jediného zdrojového souboru. Předtím, než bude zmíněna část zabývající se hovory, je nutné uvést obecné náležitosti externího modulu, které má zdrojový soubor obsahovat. Zdrojový soubor je typu *.c* z čehož vyplývá, že jazykem, v kterém se moduly programují je strukturovaný, vyšší programovací jazyk C.

Obsah zdrojového souboru musí začínat definicí externího modulu, aby kompilátor při kompilaci poznal, že se jedná o modul a má jej zakomponovat do ústředny [15].

```
#ifndef AST_MODULE
#define AST_MODULE
#endif
```

V další části se vkládají potřebné knihovny, jejichž funkce daný modul využívá. Následuje samotný zdrojový kód modulu. Na konec modulu je nutné vložit funkce, které se volají při načítání a vypínání modulu uvnitř spuštěné ústředny [15].

```
static int load_module(void)
{
ast_log(LOG_NOTICE, "Zásuvný modul byl úspěšně nahrán do ústředny
\n");
return AST_MODULE_LOAD_SUCCESS;
}


static int unload_module(void)
{
ast_log(LOG_NOTICE, "Zásuvný modul se uspesne vypnul \n");
return 0;
}
```

Pojmenování těchto funkcí musí být dodrženo. Zároveň musí tyto funkce vždy vracet hodnotu typu *int*. U funkce *load_module* má Asterisk v knihovně *utils.h* definovaná následující makra [15].

```
enum ast_module_load_result
{
AST_MODULE_LOAD_SUCCESS = 0,
AST_MODULE_LOAD_DECLINE = 1,
AST_MODULE_LOAD_SKIP = 2,
AST_MODULE_LOAD_PRIORITY = 3,
AST_MODULE_LOAD_FAILURE = -1
}
```

Každé z těchto maker má přiřazenou celočíselnou hodnotu typu *int*, která umožňuje tuto funkci rozvětvit a na základě nějaké podmínky pak vrátit dané makro. Jako poslední je nutno vložit makro, které je součástí procesu registrace modulu do ústředny.

```
AST_MODULE_INFO_STANDARD(ASTERISK_GPL_KEY, "Zasuvny modul realizovany
v ramci DP - Ondrej Melichar");
```

Uvedený příklad obsahuje minimální potřebné informace [15], které musí makro obsahovat. Řetězec *ASTERISK_GPL_KEY* značí, že autor modulu souhlasí s licenčními

podmínkami Asterisku [15]. Tento řetězec je následován krátkým popisem modulu.

Část obsahující vyžadovanou funkcionalitu je o něco složitější. Již dříve bylo zmíněno, že veškeré funkce stacku chan_sip jsou obsaženy v jednom jediném souboru. Důsledkem této skutečnosti je fakt, že chan_sip není rozšiřitelný mezi jiné moduly [15]. Ačkoliv je chan_sip implementován do Asterisku, což je modulární systém, sám o sobě na nižší úrovni modulární není, neboť neobsahuje potřebná API pro komunikaci s dalšími moduly. Nezvládne tedy využít funkce jiného modulu nebo z něj získat hodnotu nějaké proměnné a inverzně nedokáže sdílet své funkce a hodnoty svých proměnných s jinými moduly.

Z uvedeného faktu vyplývá, že pokud by byl uvažován kanálový ovladač chan_sip, není možné vytvořit externí zásuvný modul, který by pracoval se SIP zprávami. Tento datový tok je totiž směrován na port 5060, kde naslouchá stack chan_sip, což značí, že takovýto datový tok je analyzován a spravován výlučně jen tímto stackem.

Toto tvrzení nejen vyplývá ze zdroje [6], ale také bylo v průběhu celé práce empiricky ověřeno. V rámci testování bylo tedy navrženo jediné možné řešení – editace samotného kanálového ovladače chan_sip a následné nahrazení výchozího modulu tímto modifikovaným.

Zdrojový soubor kanálového ovladače chan_sip čítá přes 35 000 řádků a každý z nich plní jistý účel. Proto je nutné najít vhodné místo, kam přidat vyžadované funkce tak, aby nebyla ohrožena stávající funkčnost. Po důkladné analýze lze konstatovat, že toto místo se nachází uvnitř funkce *handle_request_do*. Tato pasáž dostává na svůj vstup přímo danou zprávu, kterou je nutno zpracovat, a také socket odesílatele.

Informace, které je nutné extrahovat z paketu, jsou obsaženy v SIP zprávě, která přichází na vstup této funkce. Než je možné, jakkoliv začít analyzovat danou zprávu, je potřeba ověřit, že paket není poškozen a podmínkou *if* ověřit, jestli je možné rozebrat jej pomocí funkce *parse_request*. Pokud ano, lze pokračovat.

```
struct sip_request *req;

      if (parse_request(req) == -1)  /* Ověření paketu */
      {
            ast_str_reset(req->data); /* Předběžné ukončení procesu */
            return 1;
      }
```

Samotná funkce *parse_request* sloužící k ověření paketu je velmi obsáhlá, a proto nebude vložena do tohoto textu. Její obsah je však k nahlédnutí v příloze A. Pakliže je podmínka splněna, je možné začít analýzu dané zprávy.

Kontrola parametru *Content-Length*, který dle doporučení RFC 3261 musí být vždy přítomen a nesmí být záporný, je provedena následovně.

```
const char *CL;
CL = sip_get_header(req, "Content-Length");

int CL_INT = atoi(CL);
if(CL_INT<0)
      {
      ast_log(LOG_WARNING,"Hodnota Content-Length je: %d \n",CL_INT);
      ast_log(LOG_WARNING,"Hodnota je zaporna! Neplatna zprava dle RFC
      3261 \n");
      }

else
      {
      ast_log(LOG_NOTICE,"Hodnota Content-Length je: %d \n",CL_INT);
      ast_log(LOG_NOTICE,"Hodnota Content-Length neni zaporna. Platna
      zprava dle RFC 3261 \n");
      }
```

Na úvod je definován ukazatel na proměnnou typu *const char*. Do této proměnné se následně pomocí funkce *sip_get_header* uloží položka z hlavičky zprávy SIP podle toho, jaký textový řetězec je obsažen v parametru. Tato funkce je definována v předchozí části souboru, nicméně pro její obsáhlost zde nebude vypsána. K nahlédnutí bude samozřejmě v příloze A. Následně je vytvořena proměnná typu *int*, do které je vložena hodnota obsažena v paměťovém místě, na které ukazuje ukazatel, a to po převodu *char* na *int* pomocí funkce *atoi*, neboť hodnota v paměti je brána jako typ *char*. Jelikož dle doporučení RFC 3261 musí mít položka *Content-Length* hodnotu vyšší nebo rovnou 0, následuje podmínka, která toto implementuje. Zde se vyhodnotí hodnota parametru Content-Length a uživatel je pomocí funkce *ast_log* zpraven o výsledku v příkazovém řádku Asterisku, popř. záznamem v logu. Pokud je vyhověno doporučení RFC 3261, v příkazové řádce se objeví oznámení typu *NOTICE*. V opačném případě jde o oznámení *WARNING*.

Detekce zprávy INVITE dle starého doporučení je již o něco obsáhlejší. Pracuje se zde s implicitními funkcemi Asterisku a přistupuje se k položkám struktur definovaných v jeho knihovnách.

```
struct sip_via *via_test;
via_test = parse_via(sip_get_header(req, "Via"));

char fromtag[128];
gettag(req, "From", fromtag, sizeof(fromtag));
```

Jako první je vytvořena nová instance ukazatele na implicitní struktury *sip_via*. Tomuto ukazateli jsou pak přiřazeny souřadnice v paměti, kde se nachází parametr *Via* v hlavičce SIP zprávy, kde je obsažen dodatečný parametr *Branch*. Toto je provedeno pomocí funkce *parse_via* která slouží jako tzv. *parser* neboli analyzátor zprávy. Dále je vytvořeno pole *fromtag* typu *char* o velikost 128 Bytů, do kterého je uložen parametr

Tag z pole *From* dané SIP zprávy *req*, která byla zmíněná výše. Velikost řetězce vloženého do pole *fromtag* omezuje parametr *sizeof(fromtag)*. Dalším krokem je analýza získaných dat a vyhodnocení.

```
if(via_test->branch)
{
ast_log(LOG_NOTICE,"Parametr Branch ma hodnotu %s \n",via_test-
>branch);

   if(!(ast_strlen_zero(fromtag)))
   {
       ast_log(LOG_NOTICE, "Parametr Tag ma hodnotu %s \n",  fromtag);
       ast_log(LOG_NOTICE, "Nejedna se o zpravu dle RFC 2543 \n");
      }

    else
      {
      ast_log(LOG_WARNING, "Absence pole Tag.\n");
      }
    }
else
{
ast_log(LOG_WARNING, "Absence pole Branch. \n");
}
```

K vyhodnocení slouží sada podmínek. V první podmínce je pokus o přistoupení do prvku struktury *via_test*. Pokud se daný prvek – *Branch* – ve struktuře nenachází, lze vyhodnotit, že se jedná o zprávu dle doporučení RFC 2543. Pokud zde absence není, lze přejít k druhé podmínce, která zkoumá, jestli pole *fromtag* bylo naplněno, a to pomocí implicitní funkce *ast_strlen_zero*. Podle toho, jestli je pole prázdné nebo ne je vyhodnoceno, jestli se jedná o zprávu dle RFC 2543 či nikoliv. Podmínka týkající se pole *Content-Length* byla vyhodnocena již dříve.

Posledním z útoků, který se modul snaží detekovat, je *Missing Required Headers*.

```
const char *CID = sip_get_header(req, "Call-ID");
const char *FROM = sip_get_header(req, "From");
const char *TO = sip_get_header(req, "To");


int CID_INT = strlen(CID);
int FROM_INT = strlen(FROM);
int TO_INT = strlen(TO);
```

Pro každou položku, kterou je potřeba v rámci vyhodnocení analyzovat, je vytvořena proměnná typu ukazatele na *const char*, které je přiřazen ukazatel na místo paměti získaný již dříve použitou metodou *sip_get_header*. Hodnotu v paměti, na kterou ukazatel ukazuje je potřeba vyhodnotit. Postup je takový, že jsou vytvořené tři proměnné typu *int*, kterým je přiřazena délka textového řetězce, na který ukazují vytvořené pointery (ukazatele). Následuje vyhodnocení získaných dat.

```
if(CID_INT!= 0)
      {
      if(FROM_INT!=0)
            {
            if(TO_INT!=0)
                  {
                    ast_log(LOG_NOTICE,"Zprava SIP obsahuje veskera
                    povinna pole dle RFC 3261 \n");
                  }
            }
      }
else
      {
      ast_log(LOG_WARNING,"Ve zprave SIP se vyskytla absence pole
      From, To, nebo Call-ID! \n");
      }
```

Výsledek vyhodnocení záleží na několika podmínkách. Stručně se dá říci, že algoritmus prochází jednotlivé proměnné typu *int* a zkoumá, jestli jsou nulové. Pokud je některá z nich nulová, indikuje to absenci dané položky v SIP zprávě a uživatel je zpraven o tomto faktu v příkazové řádce zprávou typu *WARNING*. Funkčnost a implementace modulu jsou rozebrány v kapitole 8.

7.3 Modul pracující se stackem res_pjsip

U stacku res_pjsip, který využívá knihoven PJSIP, je možnost tvorby externích kooperujících modulů umožněna díky API, která zajišťují mezi-modulární komunikaci a lze tedy k datům, která jsou směřována na port 5060 přistupovat i z více než jednoho modulu. V rámci práce bylo nutné do testování zahrnout stack res_pjsip, neboť bez něj by nebylo samostatný zásuvný modul možné vytvořit.

Knihovna PJSIP obsahuje nespočet různých funkcí a knihoven [10]. Nejen ty, ale i celé moduly na sebe mnohdy vzájemně odkazují a vzájemně spolupracují. Pro plné porozumění je tedy nejprve nutné tyto vazby řádně prozkoumat.

Zdrojový soubor modulu má opět koncovku *.c* stejně jako tomu bylo u stacku chan_sip. Hlavička souboru musí znovu implementovat makro registrace modulu k ústředně.

```
#ifndef AST_MODULE
#define AST_MODULE
#endif
/*
inkluze knihoven
*/

/*** MODULEINFO
        <depend>pjproject</depend>
        <depend>res_pjsip</depend>
        <support_level>core</support_level>
 ***/
```

Moduly spolupracující s PJSIPem nově musí nově [15] obsahovat i pasáž

MODULEINFO [10], která definuje, na kterém modulu daný zásuvný modul přímo závisí, a jaké úrovni pracuje. V tomto případě pracuje na úrovni jádra.

```
static pjsip_module res_modul_xmelic20 = {
    .name = { "Zasuvny modul pro ustrednu Asterisk - stack PJSIP -
Ondrej Melichar - v4", 72 },
    .priority = PJSIP_MOD_PRIORITY_TRANSPORT_LAYER - 2,
    .on_rx_request = analyza_paketu,
    .on_rx_response = analyza_paketu,
};

static int load_module(void)
{

    ast_log(AST_LOG_WARNING, "Zasuvny modul byl uspesne spusten! \n");
    ast_sip_register_service(&res_modul_xmelic20);
    return AST_MODULE_LOAD_SUCCESS;
}
static int unload_module(void)
{
    ast_sip_unregister_service(&res_modul_xmelic20);
    return 0;
}

AST_MODULE_INFO(ASTERISK_GPL_KEY,   AST_MODFLAG_LOAD_ORDER,   "Zasuvny
modul pro ustrednu Asterisk - stack PJSIP - Ondrej Melichar - v4",
      .requires = "res_pjsip",
      .load = load_module,
      .unload = unload_module,
      .load_pri = PJSIP_MOD_PRIORITY_TRANSPORT_LAYER  - 2,
);
```

Výše zobrazený kód musí zdrojový soubor nutně obsahovat. Lze si povšimnout několika sekcí, které jsou modifikované nebo dokonce nově přidané, oproti chan_sip stacku. Na úvod je potřeba vytvořit novou instanci struktury *pjsip_module* s názvem modulu – v tomto případě *res_modul_xmelic20*. Uvnitř této funkce se definuje popis modulu, jak bude uvedeno při zobrazení informací o modulu v ústředně. Dále se definuje položka *.priority*, což nastavuje, jakou důležitost dostane modul mezi ostatními v rámci práce s příchozími daty. Priorita je u stacku *res_pjsip* velmi důležitá. Priorita je u PJSIPu vyjádřena číselně, a to od 0 do 64, kde 0 má nejvyšší prioritu. Různé části stacku res_pjsip si příchozí data předávají podle toho, jakou mají prioritu a pokud by u zásuvného modulu byla priorita příliš nízká, mohlo by se stát, že se k modulu data vůbec nedostanou, neboť některý z předchůdců mohl data zahodit a přerušit tak předání dále. I proto je nutné správně porozumět ostatním modulům. V rámci práce je nutno nastavit prioritu PJSIP_MOD_PRIORITY_TRANSPORT_LAYER - 2, což odpovídá numerické hodnotě 6 [10], jinak není zaručena správná funkčnost. Dále jsou zde položky *.on_rx_request* a *.on_rx_response*, které udávají, kam mají být příchozí data směrována. Konkrétně jsou směrována na hlavní vytvořenou funkci s názvem *analyza_paketu*

U funkcí na načtení a vypnutí modulu přibylo využití funkce *ast_sip_register_service*, která značí, že daný modul pracuje s protokolem SIP a registruje jej jako službu [10].

Hlavní roli u tohoto modulu hraje již zmíněná funkce *analyza_paketu*, která přímo pracuje s daty, která dostane na vstup, a provádí vyhodnocení tak, jako tomu bylo u stacku chan_sip. Tato funkce je velmi obsáhlá, a proto zde samozřejmě nebude prezentován celý její kód, ale budou zde v rámci demonstrování rozdílu o oproti stacku chan_sip uvedeny některé úryvky. Prototyp funkce je následující.

```
static pj_bool_t analyza_paketu(pjsip_rx_data *rdata)
```

Ukazatel na proměnnou *rdata* typu *pjsip_rx_data* je hlavním zdrojem dat. Knihovna PJSIP obsahuje značný počet struktur, do kterých se musí hluboce zanořovat pro získání hodnoty jednotlivých položek hlavičky SIP zprávy.

Pro demonstrování přístupu k hodnotám sloužícím k vyhodnocení validity zprávy je uvedena následující pasáž.

```
char from_buffer[1024];
pjsip_uri *uri_from = NULL;
uri_from = pjsip_uri_get_uri(rdata->msg_info.from->uri);
pjsip_uri_print(PJSIP_URI_IN_FROMTO_HDR, uri_from, from_buffer,
sizeof(from_buffer)-1);

if(from_buffer!=NULL)
{
ast_log(LOG_NOTICE,"Parametr From: ma hodnotu %s \n",from_buffer);
      /*
      další kód
      */
}
else
{
        ast_log(LOG_WARNING,"Zprava neobsahuje parametr From!");
}
```

Uvedený úryvek se zabývá zjištěním přítomnosti a obsahu parametru *From*. Vytvoří se instance struktury *uri_from,* resp. ukazatel, který z počátku neukazuje na nic. Následně je do něj uložena adresa na místo v paměti pomocí funkce *pjsip_uri_get_uri*, která vstupuje do struktury *msg_info*, čímž je získáno URI pole *From*. Získané informace jsou vyjmuty a vloženy do pole typu *char* s názvem *from_buffer*.

Podobně jsou získány i ostatní pole týkající se útoku Missing Required Headers. Co se týče detekce zprávy dle syntaxe RFC 2543, následuje úryvek z této pasáže.

```
if(((int)rdata->msg_info.from->tag.slen))
 {
  ast_log(LOG_NOTICE,"Parametr Tag ma hodnotu %.*s \n",(int)rdata-
 >msg_info.from->tag.slen ,rdata->msg_info.from->tag.ptr);
 }
else
 {
  ast_log(LOG_WARNING,"Chybi parametr Tag! Zprava dle RFC 2543. \n");
 }
```

V této sekci se do podmínky vkládá délka pole *Tag*, respektive ukazatel na něj. Pokud je místo, na které ukazatel ukazuje prázdné, je zřejmé, že toto pole chybí. Pokud prázdné není, je možno jej přečíst pomocí vstoupení do struktury *msg_info* a ukazatele na tento parametr, a výpis do konzole ohraničit hodnotou délky tohoto parametru, aby bylo zajištěno, že se do příkazové řádky Asterisku vypíše opravdu pouze daný parametr.

Obdobně je řešeno i ověření pole *Content-Length* a tedy i jeho konkrétní hodnota.

```
if(!(rdata->msg_info.clen))
   {
    ast_log(LOG_WARNING,"Chybi parametr Content-Length! Zprava dle RFC
2543. \n");
   }

else
   {
     ast_log(LOG_NOTICE,"Parametr Content-Length ma hodnotu %d
\n",rdata->msg_info.clen->len);
    }
```

V podmínce je ověřeno, že ukazatel ukazuje na místo, které není prázdné a v případě, že prázdné není, vstoupí opět do struktury *msg_info* a získá hodnotu proměnné. Tato hodnota je také vypsána do konzole. Funkčnost a implementace modulu jsou testovány v následující kapitole.

8 REALIZACE ÚTOKŮ A IMPLEMENTACE MODULU

V této kapitole bude ověřen průběh zvolených útoků metodou monitorování logu ústředny a taktéž sledováním síťového provozu pomocí nástroje Wireshark. Pro demonstraci bude každý útok zobrazen na čisté instalaci Asterisku, neboť se zde očekává správná odezva ústředny, nicméně na závěr kapitoly budou odpovědi jednotlivých ústředen na všech nástavbách zobrazeny a porovnány v tabulce. V této kapitole bude taktéž na čisté instalaci ústředny zkoumána přenositelnost modulu mezi jednotlivými konfigurátory Asterisku.

8.1 Ověření funkčnosti a přenositelnosti modulu

Ověření přenositelnosti proběhne u obou dvou modulů, které popisuje kapitola 7, respektive se jedná o modifikovanou verzi modulu chan_sip jako celku, a plnohodnotný zásuvný modul, pokud jde o stack res_pjsip. K demonstraci ověření funkčnosti poslouží čistá instalace Asterisku. Kompatibilita či nekompatibilita ostatních testovaných konfigurátorů bude zkoumána v podkapitolách a bude shrnuta v závěrečných tabulkách.

8.1.1 Modul pracující se stackem chan_sip

Aby bylo možné modifikovaný modul chan_sip, který byl vytvořen v kapitole 7.2, úspěšně nahrát do ústředny, je nutné jej zkompilovat [2]. Vytvořený soubor se zdrojovým kódem je potřeba umístit do podsložky Asterisku *channels*, kde se nachází původní soubor a přepsat jej. Následně je provedena kompilace Asterisku příkazem *make*. Soubor je ve výchozím režimu zahrnut do kompilačního procesu, a proto jej není potřeba dodatečně přidávat příkazem *make menuselect*. Po dokončení kompilace je možné ověřit příkazem *module show like chan_sip*.

```
*CLI> module show like chan_sip
Module                         Description                       Use Count  Status      Support Level
chan_sip.so                    Session Initiation Protocol (SIP)  0          Running              core
1 modules loaded
```

Obr. 36: Ověření stavu modulu

Modul je ve stavu *running* a proto lze konstatovat, že nedošlo k žádné kompilační chybě. Nyní je možné ověřit funkci analýzy paketů v rámci běžného provozu, tedy bez síťových útoků či porušených paketů. Do ústředny bude vyslána zpráva *REGISTER*.

```
NOTICE[8985]: chan_sip.c:29157 handle_request_do: Hodnota Content-Length je: 0
NOTICE[8985]: chan_sip.c:29158 handle_request_do: Hodnota Content-Length neni zaporna. Platna zprava dle RFC 3261
NOTICE[8985]: chan_sip.c:29168 handle_request_do: Parametr Branch ma hodnotu z9hG4bK-524287-1---a514c6b7e6f2ad8f
NOTICE[8985]: chan_sip.c:29171 handle_request_do: Parametr Tag ma hodnotu a6798654
NOTICE[8985]: chan_sip.c:29172 handle_request_do: Nejedna se o zpravu dle RFC 2543
NOTICE[8985]: chan_sip.c:29217 handle_request_do: Zprava SIP obsahuje veskera povinna pole dle RFC 3261
```

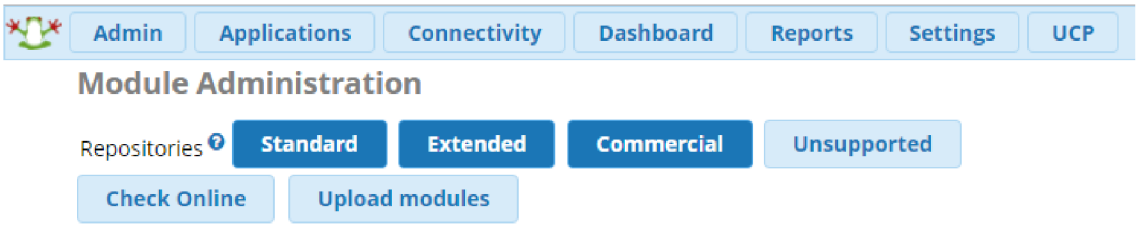
Obr. 37: Funkční zásuvný modul – stack chan_sip

Na Obr. 37 jde vidět správná funkce editovaného modulu chan_sip. Modul správně analyzuje danou SIP zprávu a dokáže v ní lokalizovat daná pole, jejichž hodnoty poté extrahuje a informuje o nich uživatele pomocí zprávy *NOTICE*.

Tímto je tedy potvrzeno, že modul za běžného provozu pracuje správně. Práce si ovšem klade za cíl průzkumu problematiky přenositelnosti. Možnosti přidávání modulů do jednotlivých konfigurátorů jsou popsány níže.

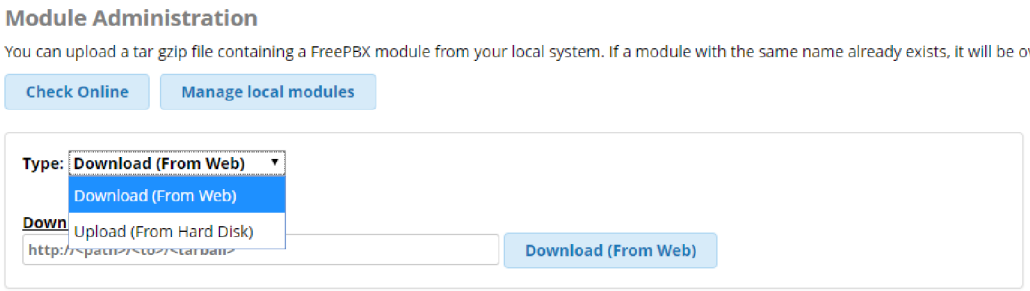
AsteriskNOW

Nástavba na bázi FreePBX má ve svém webovém rozhraní vlastní sekci pro přidávání modulů a nachází se v záložce *Admin/Module Admin*, viz Obr. 38.



Obr. 38: Sekce s moduly u AsteriskNOW

Po zvolení možnosti *Upload Modules* je uživatel přesměrován do části pro nahrávání modulů. Nástavba typu FreePBX má dvě možnosti přidání modulu. Lze jej nahrát z lokálně přidaného pevného disku, nebo skrze URL adresu.



Obr. 39: Způsoby nahrání modulu u AsteriskNOW

Tento postup ovšem nelze na mnou vytvořený modul aplikovat. AsteriskNOW, potažmo FreePBX, má svou vlastní architekturu takového modulu. Je nutno si uvědomit, že FreePBX očekává, že při nahrávání modulu dodržíme právě tuto architekturu. To zahrnuje grafické rozhraní modulu, instalační a odinstalační proces a další soubory napsané v PHP. Samotná funkce modulu je pak obsažena v .xml souboru. Pokud si uživatel přeje přidat funkční modul bez jakéhokoliv grafického rozhraní, který by pouze plnil určitou funkci, je nutno zvolit trochu obtížnější přístup, byť dle [7] to ústředna FreePBX, resp. AsteriskNOW nabízí taktéž.

Z kapitoly 7 je zřejmé, že mnou vytvořený modul, který je reprezentován souborem se zdrojovým kódem s koncovkou *.c* je nutné kompilovat spolu s Asteriskem. Ústředna AsteriskNOW resp. FreePBX nenabízí kompilaci samotného Asterisku.

Zdrojové soubory Asterisku jsou zabaleny do instalačních souborů celého AsteriskNOW a proto pokud bychom chtěli zkompilovat a znovu nainstalovat Asterisk, muselo by to být provedeno způsobem opětovné instalace celého prostředí. V tomto případě by ale nastal problém s přidáním modulu, neboť do instalačních souborů nelze zasahovat a není tedy možné modul zkompilovat spolu s jádrem, neboť se jedná o soubory ve formátu *img*, což je soubor dat čistého obrazu disku.

AstLinux

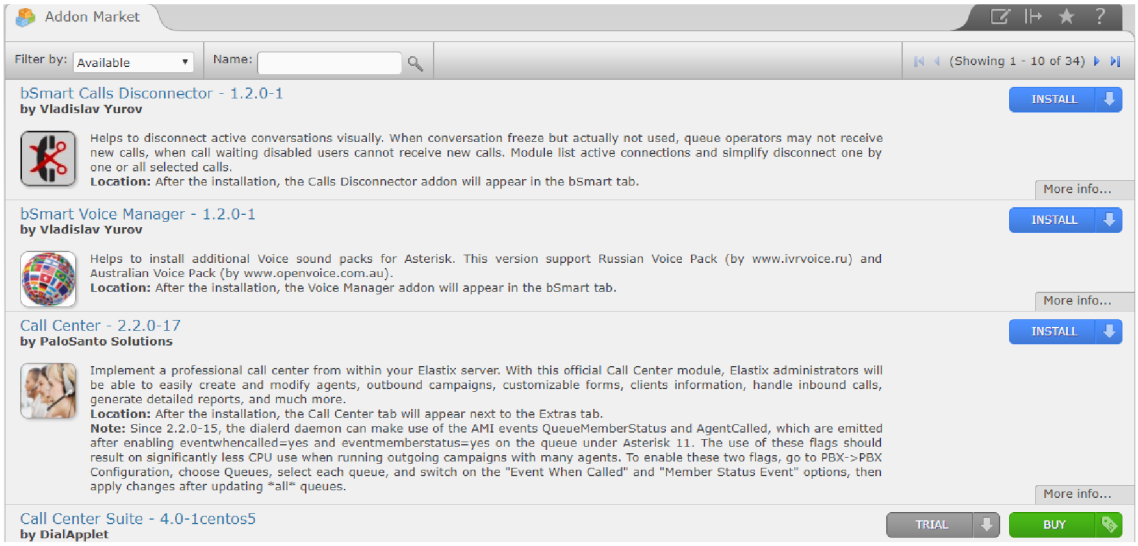
AstLinux nabízí monitoring příkazové řádky i ve svém webovém rozhraní. Zde jsou vypsány všechny události, o kterých Asterisk informuje uživatele. Stejně jako u ostatních konfigurátorů Asterisku, i zde je možnosti v samotném příkazovém řádku systému AstLinux přepnutí do CLI Asterisku. Zde je možno provést výpis všech modulů [3].

Bylo zjištěno, že AstLinux bohužel i přes svou open-source podstatu neumožňuje přidání jakýchkoliv modulů. Jediným způsobem, jak rozšířit předdefinované nástroje o další je několik tzv. add-onů, které byly vytvořeny vývojáři jako komplexní balíček určitých funkcí. Těchto add-onů je opravdu pouze několik, a nejvýraznějším z nich je *Weather AGI*, jehož funkcí je, po zavolání na klapku 1199, sdělení aktuální předpovědi počasí [3].

Elastix

Nástavba Elastix má vlastní grafické rozhraní, které je kvalitní, propracované a srovnatelné s grafickým rozhraním FreePBX. Elastix obsahuje spoustu užitečných nástrojů, které jsou již předem naprogramované a použitelné, jako je například logování, automatické nahrávání hovorů dle klapky nebo rozhraní (SIP/Dahdi/IAX). Dále taky obsazenost kanálů a počet aktuálních hovorů. Nabízí také možnost funkce emailového serveru, nebo IM (Instant Messaging) serveru *Openfire* [5].

Co se týče modulů, ty mají podobný formát, jako u FreePBX. Instalují se zde ne přímo moduly, ale tzv. add-ony. Nelze ovšem instalovat add-ony vlastní, ale pouze ty, které jsou umístěny v záložce *Add-on Market*. Zde se dají instalovat moduly přidávané oficiálními vývojáři. Některé jsou zdarma, jiné placené s možností trial verze [5].



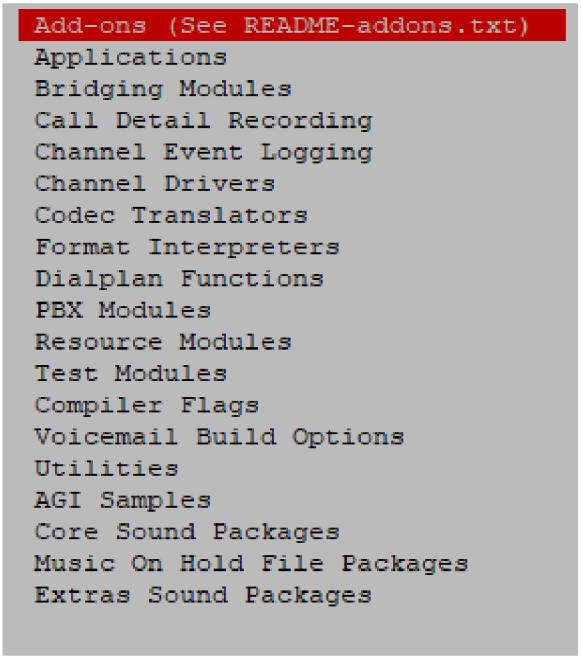
Obr. 40: Add-on Market Elastix

FreePBX

Rozložení jednotlivých sekcí v grafickém rozhraní je totožné s rozvržením u nástavby AsteriskNOW. Do FreePBX lze pomocí nástroje v záložce *Admin/Module Admin* přidat další moduly, které ovšem musí dodržovat jistou modulovou architekturu specifickou pro FreePBX. Samotný modul implementující funkce do Asterisku touto cestou přidat nelze [9].

8.1.2 Modul pracující se stackem res_pjsip

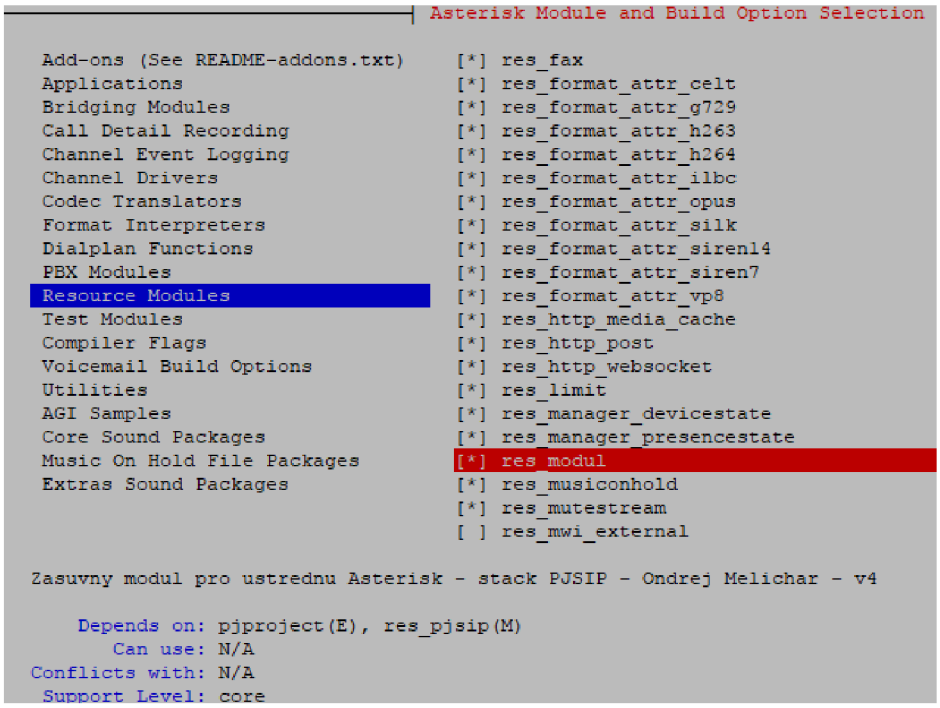
V této kapitole je demonstrován způsob implementace nezávislého modulu do ústředny Asterisk, resp. do její čisté instalace. Zdrojový soubor *res_modul.c* je nutno vložit do složky s moduly *res* mezi zdrojové soubory Asterisku. Dalším krokem implementace je výběr modulů, které chceme zkompilovat a nainstalovat. Toto nastavení je ovládáno pomocí nástroje *Menuselect* [25].



Obr. 41: Nástroj Menuselect pro výběr modulů

Zde je několik kategorií, kde je možné vybrat jednotlivé části Asterisku, které chceme zahrnout do kompilačního procesu.

Moduly v Asterisku mají souborovou koncovku *.so* (shared object), vytvořený modul v kapitole 7.3 má zatím koncovku *.c*, což je soubor se zdrojovým kódem jazyka C. Kompilace takového modulu probíhá při zadání příkazu *make*. Je samozřejmě možno ověřit, že modul je zahrnut do kompilace, a to pomocí příkazu *make menuselect*. Nutno dodat, že je potřeba nastavit okno terminálu na minimální velikost 80x27, jinak nástroj nebude fungovat [25]. Zde by měl být nově vytvořený modul, resp. jeho zdrojový soubor vidět tak, jak je vyobrazeno na Obr. 42.



Obr. 42: Výběr vytvořeného modulu v kompilační fázi Menuselect

Jsou zde zobrazeny i informace o modulu, které jsou obsaženy v sekci *MODULEINFO* ve zdrojovém kódu. Kompilaci jádra započneme tedy ve stejné složce pomocí příkazu *make* na což navazuje postup z kapitoly 5.1.1.

Následuje ověření funkčnosti modulu. Obecné nastavení modulů se provádí v konfiguračním souboru *modules.conf*. Zde je jako výchozí možnost uvedeno automatické nahrávání modulů při zapnutí ústředny. Této možnosti pro účely testování využito nebude.

```
[modules]
autoload=no
```

Právě spuštěné moduly lze zobrazit příkazem *modules show*. Do konzole se vypíší všechny aktuálně používané moduly. Testovací modul by v tomto výpisu být neměl. Spustit ho lze příkazem *module load*.

```
module load res_modul
```

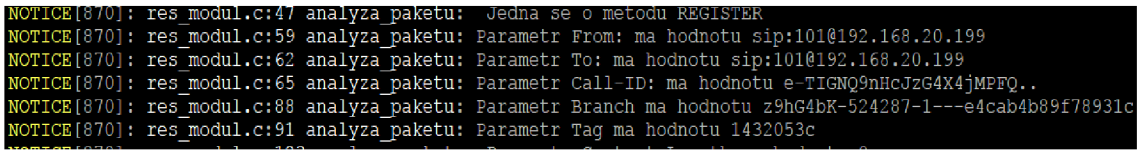
```
debianMel*CLI> module load res_modul
Loaded res_modul
[Apr  2 11:18:42] WARNING[1120]: res_modul.c:164 load_module: Zasuvny modul byl uspesne spusten!
debianMel*CLI> module show like res_modul
Module                         Description                              Use Count  Status      Support Level
res_modul.so                   Zasuvny modul pro ustrednu Asterisk - st 0          Running           unknown
1 modules loaded
```

Obr. 43: Úspěšné načtení modulu

Jak je dokázáno na Obr. 43, modul je nyní úspěšně načten a funkční, neboť do konzole se vypsalo definované oznámení. Lze také vypsat jeho stav pomocí příkazu *module show like res_modul*. Takto načtený modul lze opět deaktivovat pomocí příkazu *module unload*.

```
module unload res_modul
```

Nyní jen zbývá ověřit funkčnost modulu za běžného provozu. Do ústředny je tedy vyslána zpráva *REGISTER*.



Obr. 44: Funkční zásuvný modul – stack res_pjsip

Na Obr. 44 lze pozorovat, že zásuvný modul správně reaguje na přijaté pakety. Příkazová řádka Asterisku informuje o tom, že zprávu podává modul *res_modul*, konkrétně funkce *analyza_paketu*. Paket je správně analyzován a podává informaci o tom, že se jedná o metodu *REGISTER* a sděluje další údaje, které jsou klíčové pro následné sledování odezvy ústředny na síťové útoky. Tímto tedy byla ověřena funkčnost vytvořeného nezávislého zásuvného modulu pro čistou instalaci Asterisku.

Byla zkoumána také možnost přenositelnosti modulu na konfigurátory Asterisku, nicméně zde nastává stejný problému jako u modulu chan_sip. Prvním problémem je fakt, že externí moduly se musí kompilovat společně s jádrem při instalaci Asterisku. Toto bohužel není možné, neboť instalační soubory konfigurátorů Asterisku se skládají z obrazů disků *.img*, do kterých nelze zasahovat.

Druhým problémem jsou knihovny. Knihovny, které modul využívá jsou s různými verzemi Asterisku resp. PJSIPu obměňovány a mohou tedy nastat problémy s některými funkcemi uvnitř kódu. Tento problém by se dal řešit úpravou kódu pro každý z testovaných konfigurátorů, nicméně stále je zde problém nemožnou kompilací spolu s jádrem.

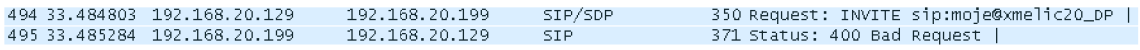
8.2 Missing Required Headers

Před spuštěním útoku je nutné zapnout funkci *debug*, která vyšetřuje SIP provoz v ústředně pomocí příkazu *sip set debug on*. Tímto bude dosaženo toho, že ústředna vyobrazí příchozí SIP zprávy do konzole. Následně je možné spustit test.

Po spuštění testu v obslužném programu Spirent Avalanche lze pozorovat příchozí SIP zprávy v logu ústředny.

Obr. 45: Přijmutí neúplné zprávy ústřednou Asterisk

Na Obr. 45 lze pozorovat mnou modifikovaný obsah UDP paketu. Jasně je zde vidět absence polí *From, To* a *Call-ID*. Lze zde také pozorovat pozměněné SIP URI a informace o vlastníkovi v poli *o*. Ve spodní části obrázku informuje ústředna, že ve zprávě nebyly tyto pole nalezeny, nezmiňuje se ovšem o poli *To*, ačkoliv zde toto pole jasně chybí. Odpověď ústředny je zobrazena na Obr. 46.

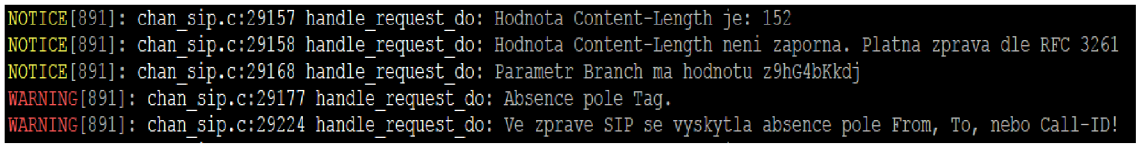


Obr. 46: Odpověď ústředny Asterisk na neúplnou zprávu

Vidíme, že ústředna informuje účastníka o špatném požadavku zprávou *400 Bad Request*, což odpovídá doporučení RFC 3261, potažmo RFC 4475. Reakce ostatních konfigurátorů shrnuje závěrečná tabulka v kapitole 8.5.

8.2.1 Reakce modulu pracující se stackem chan_sip

Funkčnost modulu byla již za běžných podmínek ověřena. Zbývá tedy podrobit jej útokům. V ideální případě by modul měl vypsat do konzole informace o paketu a detekovat útok.



Obr. 47: Reakce na útok MRH – chan_sip

Na Obr. 47 jde vidět správná funkčnost modulu. Modul informuje o velikosti položky Content-Length a o obsahu parametru Branch. Zároveň ale po analýze zprávy zjistil, že paketu chybí pole Tag, které se váže na parametr From. V následovně tedy vyhodnotil, že se jedná o útok Missing Required Headers, neboť došlo k nedodržení doporučení RFC 3261.

8.2.2 Reakce modulu pracující se stackem res_pjsip

V této podkapitole bude testován vytvořený modul spolupracující se stackem res_pjsip, konkrétně bude sledována odezva na útok Missing Required Headers.

```
[Apr  2 18:49:51] ERROR[995]: pjproject:0 <?>:          sip_endpoint.c
Error processing packet from 192.168.20.129:1097: Missing required head
er(s) (PJSIP_EMISSINGHDR)  To [code 171050]:
INVITE sip:moje@xmelic20_DP SIP/2.0
CSeq: 193942 INVITE
Via: SIP/2.0/UDP 192.0.2.95;branch=z9hG4bKkdj.insuf
Content-Type: application/sdp
l: 152

v=0
o=xmelic20 29739 7272939 IN IP4 192.0.2.95
s=-
c=IN IP4 192.0.2.95
t=0 0
m=audio 49217 RTP/AVP 0 12
m=video 3227 RTP/AVP 31
a=rtpmap:31 LPC

-- end of packet.
```

Obr. 48: Reakce na útok MRH – res_pjsip

Z obrázku vyplývá, že ústředna, ve které je jako výchozí stack zvolen res_pjsip má již v sobě zabudovanou efektivní ochranu proti tomuto útoku. Z výstupu v příkazové řádce Asterisku se objevují zprávy typu Error, které informují uživatele o tomto typu útoku. Lze si povšimnou, že zde nejsou žádné informační zprávy od zásuvného modulu. To je způsobeno tím, že stack res_pjsip má tuto analýzu proti útokům již zabudovanou v sobě. Původní premisa byl taková, že pokud se zvýší priorita zásuvného modulu tak, aby její hodnota byla vyšší než hodnota toho modulu, který má v sobě res_pjsip implicitně zahrnut, zprávy ze zásuvného modulu budou v konzoli vidět. Nicméně nestalo se tak. I při prioritě 0, tedy nejvyšší, nebyly zprávy zobrazeny. Je to způsobeno tím, že tento implicitní analyzátor má pomyslnou prioritu „-1“ a má přednost před ostatními prvky v modulárním řetězci. Vadné zprávy jsou ihned zahazovány. Tuto skutečnost je možné ověřit v síťovém analyzátoru Wireshark.

231	32.084560	192.168.20.129	192.168.20.199	SIP/SDP	350 Request: INVITE sip:moje@xmelic20_DP |
232	32.084566	192.168.20.129	192.168.20.199	SIP/SDP	350 Request: INVITE sip:moje@xmelic20_DP |
233	32.084571	192.168.20.129	192.168.20.199	SIP/SDP	350 Request: INVITE sip:moje@xmelic20_DP |
234	32.084577	192.168.20.129	192.168.20.199	SIP/SDP	350 Request: INVITE sip:moje@xmelic20_DP |
235	32.084585	192.168.20.129	192.168.20.199	SIP/SDP	350 Request: INVITE sip:moje@xmelic20_DP |
236	32.084591	192.168.20.129	192.168.20.199	SIP/SDP	350 Request: INVITE sip:moje@xmelic20_DP |
237	32.084597	192.168.20.129	192.168.20.199	SIP/SDP	350 Request: INVITE sip:moje@xmelic20_DP |
238	32.084603	192.168.20.129	192.168.20.199	SIP/SDP	350 Request: INVITE sip:moje@xmelic20_DP |
239	32.084608	192.168.20.129	192.168.20.199	SIP/SDP	350 Request: INVITE sip:moje@xmelic20_DP |
240	32.084614	192.168.20.129	192.168.20.199	SIP/SDP	350 Request: INVITE sip:moje@xmelic20_DP |

Obr. 49: Analýza útoku MRH Wiresharkem – res_pjsip

Na Obr. 49 je tento závěr potvrzen. Ústředna na přijaté zprávy neodpovídá a rovnou je zahazuje. Potvrzuje se tedy fakt, že stack res_pjsip má lepší zabezpečení nežli chan_sip.

8.3 Negative Content Length

Po spuštění testu, který zahrnuje útok pomocí zprávy SIP INVITE se záporným parametrem v poli *Content-Length* lze tuto zprávu opět vidět v CLI prostředí Asterisku.

```
<--- SIP read from UDP:192.168.20.129:1157 --->
INVITE sip:moje@xmelic20_DP SIP/2.0
Max-Forwards: 254
To: sip:test01@xmelic20_DP
From: sip:test02@xmelic20_DP;tag=32394234
Call-ID: ncl.0ha0isndaksdj2193423r542w35
CSeq: 0 INVITE
Via: SIP/2.0/UDP 192.0.2.53;branch=z9hG4bKkdjuw
Contact: <sip:testDP@modifikovany_payloadd>
Content-Type: application/sdp
Content-Length: -123

v=0
o=xmelic20 29739 7272939 IN IP4 192.0.2.53
s=-
c=IN IP4 192.0.2.53
t=0 0
m=audio 49217 RTP/AVP 0 12
m=video 3227 RTP/AVP 31
a=rtpmap:31 LPC
<-------------->
```

Obr. 50: Přijatá zpráva INVITE

Z obrázku lze vypozorovat nejen zápornou hodnotu parametru *Content-Length*, která je zde nastavena na -123, ale také mnou modifikované SIP URI, pole *To* a *From* či pole s informací o vlastníku (*o*) a pole kontaktní – *Contact*.

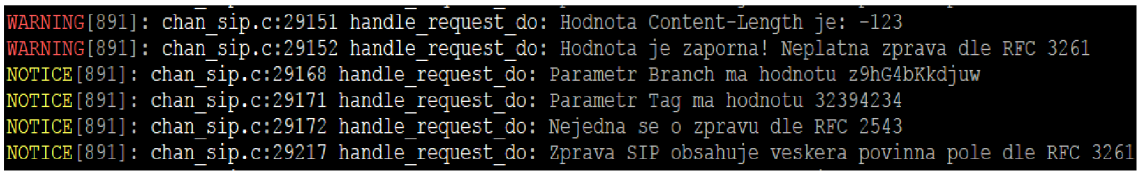
```
491 31.887104  192.168.20.129     192.168.20.199     SIP              532 Request: INVITE sip:moje@xmelic20_DP |
492 31.887488  192.168.20.199     192.168.20.129     SIP              546 Status: 401 Unauthorized |
```

Obr. 51: Analýza zprávy programem Wireshark

Na Obr. 51 je tato zpráva zobrazena pomocí analyzátoru Wireshark spolu s odpovědí ústředny.

8.3.1 Reakce modulu pracujícího se stackem chan_sip

Výchozí modul chan_sip nijak uživatele neinformuje o záporné hodnotě pole Content-Length pokud si tuto hodnotu uživatel sám nevyčte z paketu po zapnutí možnosti *sip set debug on*. Modifikovaný a nově kompilovaný modul by o tomto informovat měl.

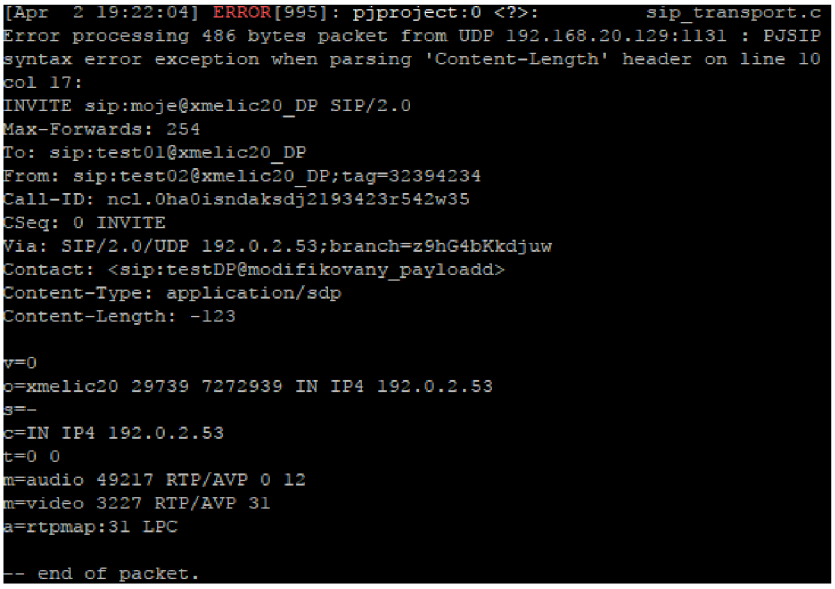


Obr. 52: Reakce na útok NCL – chan_sip

Z Obr. 52 vyplývá že modul chan_sip nyní informuje uživatele o skutečnosti, že přijatý paket má zápornou hodnotu pole Content-Length a je zpráva je tedy neplatná. Zbytek kontrolovaných parametrů taktéž vypíše, nicméně ty odpovídají standardu.

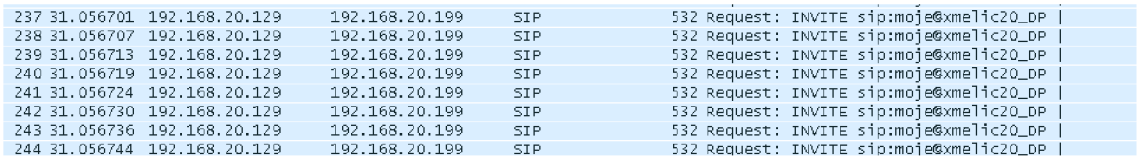
8.3.2 Reakce modulu pracující se stackem res_pjsip

Při útoku Missing Required Headers tento stack analyzoval paket ještě dříve, než jej předal k dispozici ostatním modulům. Předpoklad je tedy takový, že paket bude opět ihned zahozen.



Obr. 53: Reakce na útok NCL – res_pjsip

Z uvedeného obrázku lze usoudit, že došlo ke stejnému opatření jako u předchozího útoku. V tomto případě ovšem ústředna oznámila, že došlo k problému při pracování paketů, konkrétně při zpracování pole Content-Length. Nedošlo zde však k formulaci, že hodnota pole je negativní, respektive nebyl zde jasně specifikován problém, kvůli kterému ústředna paket zahodila tak, jako tomu bylo u předchozího útoku. Ověření okamžitého zahození lze spatřit na Obr. 54. Útočník zasílal opakované neplatné zprávy *INVITE*, nicméně Asterisk tyto zprávy ihned po zjištění nevalidity zahodil.



Obr. 54: Analýza útoku NCL Wiresharkem – res_pjsip

8.4 RFC 2543 INVITE

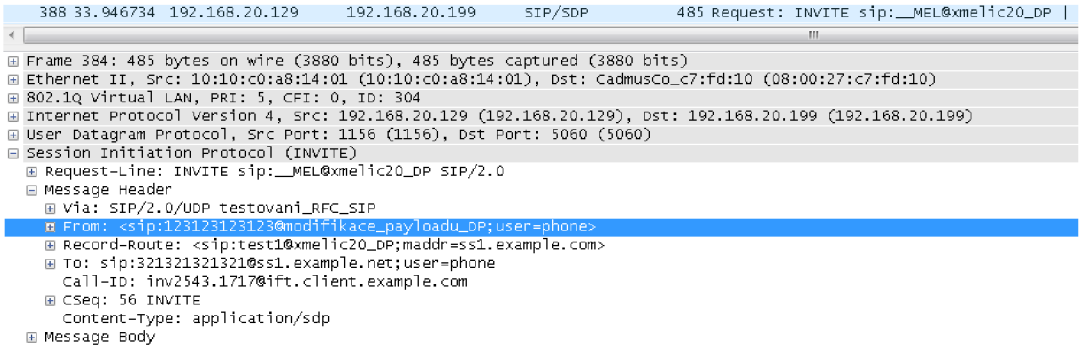
Dle doporučení 4475, respektive RFC 3261 by ústředna na tuto zprávu neměla odpovídat chybou. Po spuštění útoku lze modifikovanou zprávu se syntaxí dle starého doporučení pozorovat v CLI prostředí Asterisku.

```
<--- SIP read from UDP:192.168.20.129:1159 --->
INVITE sip:__MEL@xmelic20_DP SIP/2.0
Via: SIP/2.0/UDP testovani_RFC_SIP
From: <sip:123123123123@modifikace_payloadu_DP;user=phone>
Record-Route: <sip:test1@xmelic20_DP;maddr=ss1.example.com>
To: sip:321321321321@ss1.example.net;user=phone
Call-ID: inv2543.1717@ift.client.example.com
CSeq: 56 INVITE
Content-Type: application/sdp

v=0
o=xmeli3##=s19 IN IP4 192.0.2.5
s=-
c=IN IP4 192.0.2.5
t=0 0
m=audio 49217 RTP/AVP 0
<-------------->
```

Obr. 55: Příjem zprávy ve formátu dle RFC 2543

Na Obr. 55 lze vidět modifikovanou zprávu. Reakci ústředny je nutné hledat v logu síťového provozu, tedy za pomoci analyzátoru Wireshark.



Obr. 56: Modifikovaná zpráva zachycená Wiresharkem

Obrázek výše demonstruje zachycení modifikované zprávy. Jde vidět, že zprávě chybí sounáležitosti zmíněné v kapitole 3.2.3. Je faktem, že ústředna Asterisk na tuto zprávu INVITE nereagovala chybou. Nereagovala ale ovšem ani pozitivně. Na uvedenou zprávu neodpověděla vůbec. Ačkoliv byla v serveru interpretována jako zpráva typu SIP a tedy validní, reakce nenastala. Zpětná kompatibilita tu tedy zaznamenána nebyla.

8.4.1 Reakce modulu pracující se stackem chan_sip

V předchozí kapitole bylo zjištěno, že Asterisk na zprávy dle starého doporučení nereaguje. Bylo dokázáno, že zprávy do ústředna přijme, ale nijak je neregistruje ani na ně neodpovídá. Tuto skutečnost by měla změnit implementace zásuvných modulů.

```
NOTICE[891]: chan_sip.c:29157 handle_request_do: Hodnota Content-Length je: 0
NOTICE[891]: chan_sip.c:29158 handle_request_do: Hodnota Content-Length neni zaporna. Platna zprava dle RFC 3261
WARNING[891]: chan_sip.c:29184 handle_request_do: Absence pole Branch. Detekovana zprava dle RFC 2543!
NOTICE[891]: chan_sip.c:29217 handle_request_do: Zprava SIP obsahuje veskera povinna pole dle RFC 3261
```

Obr. 57: Reakce na útok RFC 2543 - chan_sip

Obr. 57 demonstruje odezvu modulu chan_sip na zprávu dle starého doporučení. Správně analyzoval zprávu a došlo k informování o této skutečnosti skrze příkazovou řádku. Ve zprávě chybí pole Branch a je tedy klasifikována jako zpráva dle RFC 2543 a do příkazové řádky je vytištěno varování. Na následujícím řádku je pouze konstatováno, že zpráva odpovídá doporučení 3261, což se nijak nerozporuje s tím, že se jedná o zprávu dle RFC 2543. Jelikož se jedná o zprávu typu *INVITE*, je patrné, že hodnota Content-Length je nulová.

8.4.2 Reakce modulu pracující se stackem res_pjsip

U zprávy dle starého doporučení bylo zjištěno, že Asterisk na zprávy nijak nereaguje a o ničem uživatele neinformuje. Po implementaci modifikovaného modulu byla u stacku chan_sip tato skutečnost změněna a modul nyní informaci uživateli podává. Následuje test stacku res_pjsip ve spolupráci s vytvořeným modulem.

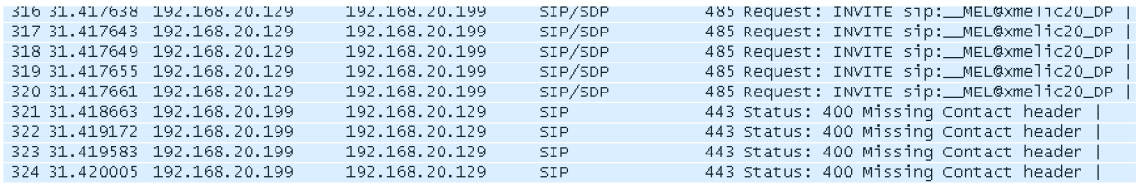
```
NOTICE[995]: res_modul.c:47 analyza_paketu:  Jedna se o metodu INVITE
NOTICE[995]: res_modul.c:59 analyza_paketu: Parametr From: ma hodnotu sip:123123123123@modifikace_pay
e
NOTICE[995]: res_modul.c:62 analyza_paketu: Parametr To: ma hodnotu sip:321321321321@ssl.example.net
NOTICE[995]: res_modul.c:65 analyza_paketu: Parametr Call-ID: ma hodnotu inv2543.1717@ift.client.exam

NOTICE[995]: res_modul.c:102 analyza_paketu: Chybi parametr Branch! Zprava dle RFC 2543.
NOTICE[995]: res_modul.c:110 analyza_paketu: Chybi parametr Tag! Zprava dle RFC 2543.
NOTICE[995]: res_modul.c:118 analyza_paketu: Chybna hodnota parametru Content-Length!
NOTICE[995]: res_modul.c:47 analyza_paketu:  Jedna se o metodu INVITE
```

Obr. 58: Reakce na útok RFC2543 – res_pjsip

Obr. 58 dokazuje, že v případě tohoto útoku ústředna zprávu hned zpočátku nezahodí ale pošle ji dalším modulům, což je v souladu s doporučením RFC 3261. Tím se data dostala i k zásuvnému modulu *res_modul* a mohlo dojít k analýze. Z obrázku vyplývá, že zpráva neobsahuje parametry Branch a Tag a klasifikuje ji tedy jako zprávu dle RFC 2543. Stack chan_sip na podobné zprávy nereagoval.

Pomocí síťového analyzátoru Wireshark je nyní potřeba zjistit, jestli tomu tak je i u stacku res_pjsip.



Obr. 59: Analýza útoku RFC2543 Wiresharkem – res_pjsip

Při testování reakce na tento útok došlu k odlišnostem oproti chan_sip. Z Obr. 59 jde vidět, že Asterisk zprávu nezahodil, neignoroval ji (jako tomu bylo v předešlých případech) ale naopak odpověděl. Odpovědí je ale chybová zpráva 400 Missing Contact Header, která poukazuje na chybějící informace v hlavičce paketu. Lze tedy konstatovat, že ani zde není zpětná kompatibilita se starým doporučením RFC 2543.

8.5 Výsledky testování

Cílem testování bylo zjistit odezvu dané ústředny na různé varianty realizovaných útoků. Každá z testovaných nástaveb, včetně referenční čisté instalace Asterisku, byla podrobena třem navrženým útokům. Reakce ústředen na modifikované pakety a informaci o přenositelnosti modulu chan_sip vyobrazuje Tab. 8.

chan_sip	**Přenositelnost modulu**	**Missing Required Headers**	**Negative Content Length**	**RFC 2543 INVITE**
Asterisk 14.16	✓	400 Bad Request	401 Unauthorized	bez odezvy
AsteriskNOW 10.13	✗	400 Bad Request	200 OK	bez odezvy
AstLinux 1.3	✗	400 Bad Request	404 Not Found	bez odezvy
Elastix 2.5	✗	400 Bad Request	401 Unauthorized	bez odezvy
FreePBX 14.0	✗	400 Bad Request	200 OK	bez odezvy

Tab. 8: Výsledky přenositelnosti modulu a reakcí na útoky – stack chan_sip

V druhém sloupci je výsledek implementace vytvořeného modulu na danou distribuci. Testovací modul dosáhl stavu funkčnosti pouze na čisté verzi Asterisku 14.16. Podmínkou funkčnosti modulu je, aby jádro Asterisku a daný modul byly zkompilovány ve stejnou chvíli. U nástaveb pobočkové ústředny Asterisk ovšem vytvořený modul nelze předem přidat do instalačního balíčku a tuto podmínku tedy nelze splnit.

V třetím sloupci tabulky se nachází první z typů útoku na ústřednu. Jedná se o útok, kdy zpráva SIP INVITE neobsahuje potřebná pole. Dle RFC 4475 by ústředna

měla odpovědět zprávou 400 Bad Request, což bylo splněno u všech z testovaných subjektů jednotně. Takovou zprávu nelze přijmout.

Lze konstatovat, že čtvrtý sloupec je ze všech nejzajímavější, neboť zde je největší variace odpovědí v závislosti na dané distribuci. Doporučení RFC 4475 nedefinuje přesnou předpokládanou odpověď ústředny, nicméně informuje, že by ústředna měla odpovědět nějakou formou chyby. Chybou reagovali tři testované subjekty, z toho dva zprávou *401* Unauthorized a jeden 404 Not Found. Zajímavá je reakce AsteriskNOW a FreePBX. Tyto dvě nástavby, ačkoliv odlišné verzí Asterisku, reagovali na zprávu se zápornou hodnotou pole Content-Length pozitivně, tedy zprávou 200 OK.

Poslední sloupec Tab. 8 ukazuje reakci ústředny na jednotlivých konfigurátorech po zaslání zprávy SIP INVITE dle starého doporučení RFC 2543. Ve všech případech ústředna zprávu přijala, interpretovala ji jako INVITE, ale zpět neodeslala zprávu žádnou.

V další části testování byl zkoumán stack res_pjsip, který je založen na knihovnách PJSIP. Důvodem byl fakt, že nelze vytvořit modul spolupracující se stackem chan_sip, který by dokázal pracovat s příchozími pakety. U vytvořeného zásuvného modulu, jenž byl pro tento stack navržen, ovšem také zůstával problém s nemožností kompilace spolu s jádrem. Vytvořený modul bylo tedy možné testovat pouze na čisté instalaci Asterisku. Výsledky tohoto testování jsou vyobrazeny v Tab. 9.

Asterisk 14.16 – stack PJSIP	
Funkčnost zásuvného modulu *res_modul*	✓
Reakce na útok Missing Required Headers	bez odpovědi – paket zahozen
Reakce na útok Negative Content Length	bez odpovědi – paket zahozen
Reakce na útok RFC 2543 INVITE	400 Missing Contact Header

Tab. 9: Výsledky funkčnosti modulu a reakcí na útoky – stack res_pjsip

Z tabulky vyplývá, že chování kanálového ovladače res_pjsip v podstatě neodpovídá doporučení RFC 4475, nicméně z praktického hlediska zde nedochází k ničemu nežádoucímu. Ústředna na nevalidní zprávy vůbec neodpovídá, což v praxi znamená, že nedochází k jejímu zbytečnému zatěžování. Rozdíl oproti stacku chan_sip lze pozorovat u všech třech útoků. Zatím co u prvních dvou ústředna paket rovnou zahodila, u útoku RFC 2543 odpověděla zprávou 400 Missing Contact Header. Tato odpověď není v souladu s RFC 3261 a lze tedy konstatovat, že ústředna není v tomto případě zpětně kompatibilní.

ZÁVĚR

Předmětem diplomové práce bylo studium open-source pobočkové ústředny Asterisk a nástaveb, které se od ní odvíjí. V rámci práce byly zkoumány funkce a možnosti, kterými ústředna oplývá a také modifikace takovýchto funkcí. Celkově bylo testováno pět subjektů s různou verzí ústředny. Zřetel byl kladen zejména na konfigurátor Asterisku, AsteriskNOW. Bylo zjištěno, že tato nástavba je, co se funkcí a grafického rozhraní týče, totožná s další nástavbou – FreePBX. Právě tato nástavba je spolu s AsteriskNOW pod správou stejné společnosti, kterou je Sangoma. První z dvojice byl původně pod správou společnosti Schmooze, jejichž modifikace operačního systému SCHMZ se v AsteriskNOW nachází. Kromě tohoto faktu se obě nástavby liší pouze vzhledem instalačního programu, ale hlavně verzí Asterisku. FreePBX, jako známější a rozšířenější z dvojice, dostává od vývojářů mnohem častější opravy a aktualizace, o čemž mimo jiné svědčí právě zastaralá verze Asterisku.

Další testovanou nástavbou byl projekt Elastix. Do verze 2.5 byl tento konfigurátor postaven na Asterisku, nicméně novější verze již obsahuje ústřednu 3CX. Tato poměrně známá nástavba byla zahrnuta do testování v poslední možné verzi, která ještě obsahuje Asterisk. Elastix oplývá bohatou sadou funkcí, neboť kromě pobočkové ústředny může sloužit také jako FAX, e-mail, nebo IM server, čímž se liší od ostatních testovaných subjektů. Nabízí také originální nástroj *Panel operátora*, který v reálném čase imituje spojovací pole, kde lze spatřit aktivní klapky a počet aktivních kanálu. Tyto kanály lze z tohoto grafického prostředí ukončit.

Posledním testovacím subjektem byla kompaktní a minimalistická nástavba zvaná AstLinux. Projekt není na tak profesionální úrovni jako jeho konkurenti, nicméně i přes to nabízí plnohodnotnou funkčnost ústředny Asterisk. Tato nástavba se vyznačuje tím, že je napsána s minimálním množstvím kódu, tedy bez nadbytečných funkcí, a běží na systému Gentoo, resp. jeho nejnutnější části. Výsledkem je ústředna s malými nároky na kapacitu, která nabízí funkčnost i z paměti flash, a jako jediná tak nepotřebuje pevný disk na svůj chod. Jediným záporem je velmi jednoduché grafické rozhraní. Webové prostředí nabízí, co se týče grafických nástrojů pouze monitorování. Veškeré konfigurace Asterisku jsou prováděny webovým editorem textových souborů, v kterých se upravují konfigurační soubory.

Jednou z věcí, které si práce kladla za cíl, bylo vytvoření vlastního zásuvného modulu, který by uživatele informoval o přicházejících SIP zprávách, dokázal je analyzovat a reagovat na síťový provoz. Problematika modulů je v práci podrobně popsána. Při tvorbě prvotního modulu bylo zjištěno, že pokud ústředna funguje na kanálovém ovladači chan_sip, nelze takový modul vytvořit, neboť stack chan_sip je monolitický, a nedovoluje po logické stránce z vnějšku přistupovat k jeho funkcím či k proměnným, s kterými pracuje. Jediným způsobem, jak tedy přidat funkce do již stávajícího modulu chan_sip, bylo jeho kód modifikovat a zkompilovat jej znovu spolu s jádrem. V tomto ohledu bylo dosaženo vytyčeného cíle. Modifikovaný modul byl funkční a správně analyzoval pakety a reagoval na síťový provoz. Tento způsob ovšem nelze reprodukovat na konfigurátorech Asterisku. Asterisk je totiž v těchto systémech zabalen přímo v instalačních souborech (což jsou obrazy disku), do kterých nelze zasahovat. Aby byla splněn cíl práce, tedy vytvoření plnohodnotného zásuvného modulu, byl nativní stack chan_sip nahrazen modernějším stackem res_pjsip založeným

na knihovně PJSIP. Tento stack je vysoce modulární, není monolitický a začíná se v posledních verzích Asterisku více uplatňovat. Sám stack se skládá z několika modulů, které mezi sebou vzájemně pracují. Byl navržen tak, aby bylo možné k němu libovolně připojovat další aplikace a moduly, neboť obsahuje několik API, přes které k jeho funkcím lze přistupovat. V tomto duchu byl tedy naprogramován vlastní zásuvný modul založený na knihovnách PJSIP. Tento modul byl v rámci práce implementován do ústředny a byla prokázána jeho funkčnost. V ohledu přenositelnosti je nutno opět konstatovat, že aby modul dosáhl požadované funkčnosti, je nutné jej kompilovat spolu s jádrem, což nebylo u testovaných konfigurátorů možné, a proto nebylo možné jej k nim připojit.

Poslední částí práce bylo podrobení nástaveb Asterisku třem typům útoků. Jelikož se práce zabývala dvěma stacky, probíhalo také testování u obou těchto kanálových ovladačů. Nejprve byla nastudována problematika a v souladu s RFC 3261 a RFC 4475 byly tyto útoky provedeny. Tomu předcházelo vytvoření patřičných scénářů pomocí testovacího systému Spirent Avalanche, kde byly také tyto útoky vytvořeny s mnou definovaným užitečným obsahem (payload). Reakce ústředny na všech zmíněných konfigurátorech byly u stacku chan_sip přehledně zaneseny do srovnávací tabulky Tab. 8. V rozporu s doporučením RFC byly u útoku *Negative Content Length* ústředny dvě, konkrétně FreePBX a AsteriskNOW. Z třetího typu testování (RFC 2543 INVITE) vyplynulo, že ústředna Asterisk zprávy INVITE dle staré syntaxe z doporučení RFC 2543 nepodporuje, neboť na tyto zprávy ani na jedné z testovaných nástaveb neodpověděla. Co se týče stacku res_pjsip, zde proběhlo testování na čisté instalaci Asterisku a odezvy ústředny dokumentuje tabulka Tab. 9. Odezva ústředny se u tohoto stacku liší od chan_sip. V případě útoků Negative Content Length a Missing Required Headers došlo k okamžitému zahození takového paketu a vyskytla se zde absence jakékoliv odpovědi odesílateli, tudíž se daný paket v důsledku prioritizační politiky k vytvořenému modulu res_modul ani nedostal a nemohl tak být analyzován, resp. vyhodnocen. U útoku RFC 2543 INVITE došlo k odpovědi zprávou 400 Missing Contact Header, z čehož lze vyvodit závěr, ani zde není ústředna zpětně kompatibilní. Jelikož nedošlo k okamžitému zahození paketu, dostala se zpráva v modulárním řetězci až k vytvořenému modulu a mohla tak být analyzována a vyhodnocena. Útok zde tedy byl modulem úspěšně detekován.

Z celého testování lze tedy vyvodit několik závěrů. Asterisk jako open-source pobočková ústředna je obecně ve své podstatě vysoce modulární. Co se ovšem týče hlubší integrace modulů do jádra, nebo do jeho zasahování v oblasti protokolu SIP, nativní stack chan_sip je již nevyhovující a sám o sobě modulární není. Proto je v praxi čím dál tím více nasazován do VoIP řešení na bázi Asterisku kanálový ovladač res_pjsip, který nabízí široké spektrum API, která dovolují přistupovat ke všem funkcím a proměnným tohoto stacku. Konfigurátory Asterisku bohužel neumožňují implementaci čistých Asterisk modulů, neboť mají vlastní modulovou politiku, která se musí dodržovat. Na druhou stranu nabízí ovšem mnoho výhod, které ocení například konkrétní společnosti, které potřebují jednoduchá a kompaktní řešení.

LITERATURA

[1] *Asterisk 14.1 Release Summary* [online]. Digium, 2017 [cit. 2017-12-03]. Dostupné z: https://downloads.asterisk.org/pub/telephony/asterisk/releases/asterisk-14.1.0-summary.txt

[2] *Asterisk module dev* [online]. Pchero21 [cit. 2017-12-03]. Dostupné z: http://wiki.pchero21.com/wiki/Asterisk_module_dev

[3] *AstLinux documentation* [online]. AstLinux Project, 2017 [cit. 2017-11-25]. Dostupné z: https://doc.astlinux-project.org/userdoc:new-install

[4] Avalanche - Testing the security of app aware networks. *Spirent* [online]. Spirent Communications, 2017 [cit. 2017-11-10]. Dostupné z: https://www.spirent.com/Products/Avalanche

[5] BARAJAS PUENTE, Gerardo. *Elastix Unified Communications Server Cookbook*. Birmingham: Packt Publishing, 2015. ISBN 978-1-84951-934-2.

[6] BRYANT, Russell. *Asterisk: the definitive guide*. Fourth edition. Sebastopol: O'Reilly, 2013. ISBN 978-144-9332-426.

[7] Building a Single Asterisk Module. *FreePBX OpenSource Project - Documentation* [online]. Sangoma Technologies, 2014 [cit. 2017-11-18]. Dostupné z: https://wiki.freepbx.org/display/FOP/Build+Single+Asterisk+Module

[8] ČÍKA, Petr. *Multimediální služby*. Brno: Vysoké učení technické v Brně, 2012. ISBN 978-80-214-4443-0.

[9] Differences betweent FreePBX and AsteriskNOW. *FreePBX Community* [online]. Sangoma Technologies, 2014 [cit. 2017-10-26]. Dostupné z: https://community.freepbx.org/t/differences-between-freepbx-and-asterisknow-no-other/23598/8

[10] Documentation. *PJSIP - Open Source SIP, Media, and NAT Traversal Library* [online]. [cit. 2018-04-08]. Dostupné z: https://trac.pjsip.org/repos/wiki/WikiStart

[11] DWIVEDI, Himanshu. Hacking VoIP: Protocols, Attacks, and Countermeasures. San Francisco: No Starch Press, 2009. ISBN 978-1-59327-163-3.

[12] MELICHAR, O. Výkonnostní testování open source PBX FreeSWITCH na různých platformách. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2016. 77 s. Vedoucí bakalářské práce Ing. Pavel Šilhavý, Ph.D..

[13] Open Source Communications Software | Asterisk Official Site. *Asterisk* [online]. Digium, 2017 [cit. 2017-12-07]. Dostupné z: http://www.asterisk.org/

[14] ORSÁK, D. Zabezpečení Open source PBX proti útokům. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. 111 s. Vedoucí diplomové práce Ing. Pavel Šilhavý, Ph.D..

[15] PRIJONO, Benny. PJSIP Developer's Guide. *PJSIP - Open Source SIP, Media, and NAT Traversal Library* [online]. 2006 [cit. 2018-03-30]. Dostupné z: http://www.pjsip.org/release/0.5.4/PJSIP-Dev-Guide.pdf

[16] *RFC 2543* [online]. Internet Engineering Task Force, 1999 [cit. 2017-12-08]. Dostupné z: https://www.ietf.org/rfc/rfc2543.txt

[17] RFC 3261. *Internet Engineering Task Force* [online]. 2002 [cit. 2017-12-08]. Dostupné z: https://www.ietf.org/rfc/rfc3261.txt

[18] *RFC 4475* [online]. Internet Engineering Task Force, 2006 [cit. 2017-12-02]. Dostupné z: https://tools.ietf.org/html/rfc4475

[19] SIP stack future. *Inside Asterisk* [online]. Digium, 2013 [cit. 2018-03-26]. Dostupné z: http://www.digium.com/blog/2013/11/20/asterisk-12-part-iv-sip-stack-future/

[20] ŠILHAVÝ, Pavel. *Telekomunikační a informační systémy*. Brno: Vysoké učení technické v Brně, 2014. ISBN 978-80-214-5027-1.

[21] ŠILHAVÝ, Pavel. *Telekomunikační a informační systémy - Laboratorní cvičení*. Brno: Vysoké učení technické v Brně, 2014. ISBN 978-80-214-4895-7

[22] Telefonní Ústředna Asterisk. *Teorie a praxe telefonie* [online]. CESNET, 2008 [cit. 2017-10-06]. Dostupné z: http://www.ip-telefon.cz/data/downld/44.pdf

[23] The History of VoIP. *BeBusinessed* [online]. Illinois, 2016 [cit. 2017-10-01]. Dostupné z: https://bebusinessed.com/history/voip-history/

[24] Transport Layer Security Protocol. *MSDN* [online]. 2017. l. [cit. 2017-10-24]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/aa380516(v=vs.85).aspx

[25] Using Menuselect to Select Asterisk Options. *Asterisk Project Wiki* [online]. Digium [cit. 2018-04-08]. Dostupné z: https://wiki.asterisk.org/wiki/display/AST/Using+Menuselect+to+Select+Asterisk+Options

[26] Využití IP sítí pro přenos hlasu. *Elektrorevue* [online]. Vysoké učení technické v Brně, Fakulta elektrotechniky a informatiky: Dan Komosný, 2001 [cit. 2017-09-12]. Dostupné z: http://www.elektrorevue.cz/clanky/01015/index.html

[27] WALLACE, Kevin. *VoIP bez předchozích znalostí*. Brno: Computer Press, 2007. Cisco systems. ISBN 978-80-251-1458-2.

[28] Základy spojovací techniky. *Střední průmyslová škola na Proseku* [online]. Praha: Střední průmyslová škola na Proseku, 2013 [cit. 2017-09-21]. Dostupné z: https://ucitel.sps-prosek.cz/~prochap/telekomunikace/

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ACME	Automatic Certificate Management Enviroment
AEL	Asterisk Extension Logic
AN	Access Network
API	Application Programming Interface
ARP	Address Resolution Protocol
CB	Central Battery
CSRC	Contributing Source Count
ČR	Česká Republika
DNS	Domain Name Systém
DoS	Denial of Service
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IAX	Inter-Asterisk eXchange
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISDN	Integrated Serviced Digital Network
IVR	Interactive Voice Response
LTS	Long Term Support
MB	Místní Baterie
MRQ	Missing Required Headers
MTO	Místní Telefonní Obvod
MTÚ	Místní Telefonní Ústředna
MÚ	Místní Ústředna
NCL	Negative Content Length
NTP	Network Time Protocol
PBX	Private Branch Exchange
PCM	Pulse-Code Modulation
PHP	Hypertext Preprocessor
PIN	Personal Identification Number

POTS	Plain Old Telephone Service
PSTN	Public Switched Telecommunication Network
QoE	Quality of Experience
QoS	Quality of Service
RFC	Request For Comments
RSU	Remote Subscriber Unit
RTCP	Realtime Transport Control Protocol
RTP	Real-time Transfer Protocol
SCP	Secure Copy
SDP	Session Description Protocol
SFTP	SSH File Transfer Protocol
SIP	Session Initiation Protocol
SIPS	Session Initiation Protocol Secure
SMTP	Simple Mail Transfer Protocol
SSL	Secure Socket Layer
SSRC	Synchronization Source Count
TDM	Time Division Multiplex
TLS	Transport Layer Security
TN	Transport Network
TTO	Tranzitní Telefonní Obvod
TTÚ	Tranzitní Telefonní Ústředna
UAC	User Agent Client
UAS	User Agent Server
UB	Ústřední Baterie
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
ÚTÚ	Ústřední Telefonní Ústředna
VoBB	Voice over BroadBand
VoIP	Voice over Internet Protocol
WWW	World Wide Web

A OBSAH PŘILOŽENÉHO CD

- xmelic20_DP.pdf – elektronická verze této práce
- chan_sip.c – soubor s modifikovaným zdrojovým kódem modulu chan_sip
- res_modul.c – soubor s vlastním zdrojovým kódem modulu res_modul
- Melichar_Asterisk.spf – exportované nastavení projektu v testovacím prostředí Spirent Avalanche
- sipnegativecontentlength.xml – modifikovaný paket použitý ve scénáři útoku Negative Content Length
- sipmissingheaders.xml – modifikovaný paket použitý ve scénáři útoku Missing Required Headers
- sipinvite2543.xml – modifikovaný paket použitý ve scénáři útoku RFC 2543 INVITE