VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

REKONSTRUKCE ZACHYCENÉHO PROVOZU KOMUNIKAČNÍCH NÁSTROJŮ

RECONSTRUCTION OF CAPTURED TRAFFIC FROM COMMUNICATION TOOLS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE TOMÁŠ AMBROŽ
AUTHOR

VEDOUCÍ PRÁCE Ing. MARTIN KMEŤ
SUPERVISOR

BRNO 2016

Zadání bakalářské práce/18833/2015/xambro11

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Ambrož Tomáš**
Obor: Informační technologie
Téma: **Rekonstrukce zachyceného provozu komunikačních nástrojů**
Reconstruction of Captured Traffic from Communication Tools
Kategorie: Počítačové sítě

Pokyny:

1. Zoznámte sa s komunikačnými nástrojmi WhatsApp a Viber, identifikujte zpôsoby komunikácie z hľadiska počítačové siete.
2. Vykonajte hĺbkovú analýzu štruktúry komunikačných protokolov vybraných nástrojov. Identifikujte udalosti, ktoré sú zaujímavé z pohľadu forenznej analýzy.
3. Diskutujte možnosti rekonštrukcie komunikácie.
4. Implementujte detekciu komunikácie daných nástrojov spolu s extrakciou zistiteľných informácií z toku do nástroja Netfox Detective.
5. Otestujte implementáciu a zhodnoťte jej prínos.

Literatura:

- KARPÍŠEK, Filip, Ibrahim BAGGILI a Frank BREITINGER. *WhatsApp network forensics: Decrypting and understanding the WhatsApp call signaling messages*. 2015
- HANCKE, P. *Whatsapp exposed: Investigative report*. 2015
  URL: https://webrtchacks.com/wp-content/uploads/2015/04/WhatsappReport.pdf
- WHAnonymous. *Authentication overview (WAUTH-2)*.
  URL: https://github.com/WHAnonymous/Chat-API/wiki/Authentication-Overview-(WAUTH-2)
- APPELMAN, Michiel, Jeffrey BOSMA a Gerrie VEERMAN. *Viber Communication Security: unscramble the scrambled*. 2011.
  URL: https://www.dropbox.com/s/0jbp62nu0f46nf1/Viber%20Communication%20Security%20unscramble%20the%20scrambled.pdf?dl=0

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kmeť Martin, Ing.**, UIFS FIT VUT
Konzultant: Karpíšek Filip, Ing., UIFS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

V dnešní době čím dál více přibývá textové a hlasové komunikace a zasílání multimediálních dat pomocí komunikačních aplikací. Z tohoto důvodu vzniká potřeba bezpečnostních služeb státu tuto komunikaci monitorovat a analyzovat za účelem odhalování trestné činnosti a následnému použití získaných informací z komunikace v soudním řízení. Proto jsem se v mé bakalářské práci pokusil analyzovat a rekonstruovat zachycenou komunikaci dvou velmi populárních komunikačních aplikací, a to Vibru a Whatsappu. Následně ze svých poznatků o komunikačních protokolech jsem implementoval rozšíření pro aplikaci Netfox Framework, která je určena právě pro rekonstrukci zachycené komunikace různých komunikačních protokolů.

Abstract

Text and voice communications increase more and more now and sending of multimedia datas with the help of communication applications. There is a need of security services of our state this communication to monitor and analyse for purpose detecting of offence and following use of gained informations from communication in the action in court. Therefore, in my Bachelor's thesis I attempted to analyse and reconstruct of intercepted communication of two very popular communication applications, Viber and WhatsApp. Following extension for application Netfox Framework, which is determinated just for reconstruction of intercepted communication of various communication reports, was implemented on based of my informations about communication reports.

Klíčová slova

Viber, WhatsApp, HTTP, SSL, RC4

Keywords

Viber, WhatsApp, HTTP, SSL, RC4

Citace

AMBROŽ, Tomáš. *Rekonstrukce zachyceného provozu komunikačních nástrojů.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Kmeť Martin.

Rekonstrukce zachyceného provozu komunikačních nástrojů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Martina Kmetě. Další informace mi poskytl při konzultaci Ing. Filip Karpíšek. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Tomáš Ambrož
16. května 2016

Poděkování

Chtěl bych poděkovat Ing. Martinu Kmeťovi za vedení mojí bakalařské práce a Ing. Filipu Karpíškovi za odbornou pomoc na této práci.

© Tomáš Ambrož, 2016.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **2**

2 Viber **3**
2.1 Získání dat 3
2.2 Analýza zachycených dat 4

3 WhatsApp **9**
3.1 Autentifikace 9
3.2 Signalizace hovoru 10
3.3 PBKDF2 13
3.4 Symetricté algoritmy 13
3.5 AES 15
3.6 RC4 15

4 Implementace **18**
4.1 Netfox Detective 18
4.2 Extrahování hesla 20
4.3 Generování klíčů pro dešifrování 21
4.4 Zpracování zpráv 22
4.5 Zpracování exportovaných objektů WaCallMsg 24
4.6 GUI pro vkládaní hesla 26

5 Testování **29**
5.1 Testování extrahování hesla 29
5.2 Testování rekonstrukce komunikace 30

6 Závěr **32**

Literatura **34**

Kapitola 1

Úvod

Mobilní zařízení se dnes nepoužívají jen k telefonování, jako tomu bylo dříve, ale slouží i k posílání zpráv, videí, obrázků. Komunikace čím dál více probíhá přes mobilní aplikace, které jsou nainstalovány v mobilních telefonech a komunikují přes Internet. Z tohoto důvodu je komunikace velmi levná, a proto dochází ke stálému zvětšování objemu komunikace přes tyto mobilní aplikace. Takováto komunikace je samozřejmou součástí současného života. Tento fakt zapříčiňuje potřeby bezpečnostních složek zachytávat a rekonstruovat komunikaci, která probíhá mezi komunikačními aplikacemi.

Cílem mé bakalářské práce je analýza a rekonstrukce zachycené komunikace dvou velmi používaných komunikačních aplikací, a to Vibru a WhatsAppu. V kapitole 2 bude ověřováno, zda Viber posílá a odesílá multimediální data a aktuální polohu v nezašifrované podobě i v novější verzi aplikace. Důvod, proč nás zajímá, zda je komunikace šifrovaná, je, že u zašifrované komunikace není možné určit její obsah a tím pádem ani její význam. Proto, aby mohl být zjištěn obsah, který zašifrovaná zpráva nese, je nutné nejprve tuto komunikaci dešifrovat. A tady vzniká celkem závažný problém. Pro dešifrování je nutné znát šifru a také klíč, pomocí kterých je komunikace zašifrována. A tyto informace se velmi těžce zjišťují. Dále bude vysvětlen způsob přijímání a odesílání multimediálních dat a aktuální polohy.

V kapitole 3 bude popsáno, jak aplikace WhatsApp navazuje spojení se serverem a jak probíhá hovor. Také zde bude popsáno, jak funguje šifrování zpráv pomocí šifry RC4. Kapitola 4 se zabývá implementací rozšíření pro Netfox Detective, který má za úkol extrahovat důležité informace pro forenzní analýzu z komunikace aplikace WhatsApp. Dále se kapitola 5 zaměřuje na testování jednotlivých částí implementace a sběrem testovacích dat pro účely testování. Na závěr v kapitole 6 bude provedeno shrnutí, co je možné ze zachycené komunikace zjistit.

Kapitola 2

Viber

Viber je komunikační program určený především pro mobilní telefony, který umožňuje volání, posílání zpráv, obrázků, videí a aktuální polohy všem uživatelům, kteří mají nainstalovanou tuto aplikaci.

Tato kapitola se zabývá ověřením, zda je možné multimediální obsah (obrázky, videa) a aktuální polohu zachytit v nezašifrované podobě ve Viberu verze 5, tak jak je uvedeno v článcích, *Viber vulnerable to MITM attack, million users at risk* [15] a *Network and device forensic analysis of Android social-messaging applications* [21], které se zabývají touto problematikou ve Viberu ve verzi 4.

2.1 Získání dat

V počítačových sítích MITM (Man in the middle) znamená, že útočník je schopen odposlouchávat případně i upravovat data, která si posílají dva uzly mezi sebou. Tyto uzly samozřejmě neví, že je útočník odposlouchává nebo jim zasahuje do komunikace. Abych realizoval ten útok za účelem zachytávat komunikaci Viberu, musím si sestavit zapojení dle následujícího schématu v obrázku 2.1. Sestavení mobilních zařízení použitých v tomto zapojení, která jsou odposlouchávaná, je popsáno v tabulce 2.1. Počítač na schématu je připojen jak k internetu, tak vytváří přístupový bod k WiFi síti, na niž se připojí mobilní zařízení. Následně na počítači zachytávám jejich komunikaci, která probíhá na WiFi síti. Zachycenou komunikaci analyzuji. Toto je popsáno v následující podkapitole.

Při vytvoření přístupového bodu k WiFi síti na počítači postupuji následovně[20]:

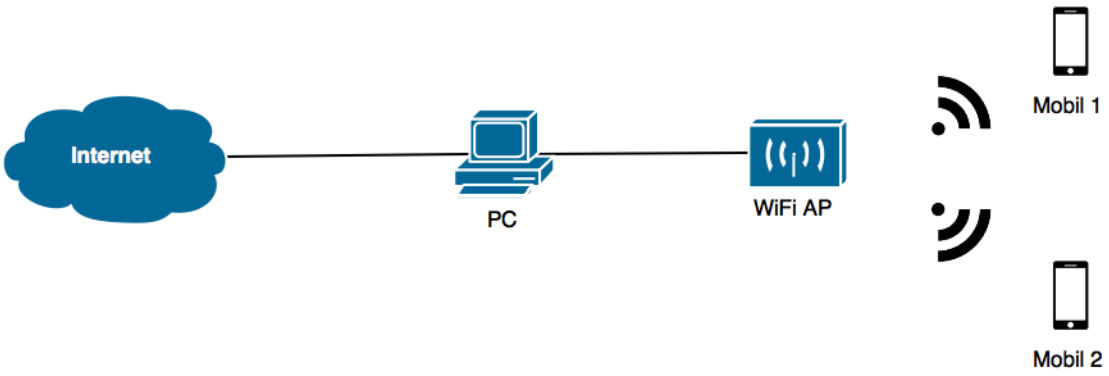
1. spustím příkazovou řádku s administrátorskými právy.
2. zadám příkaz `netsh wlan set hostednetwork mode=allow ssid=testAP key=mojeheslo`, který vytvoří virtuální WiFi adaptér.
3. nastavím sdílení přijpojení k internetu pro vytvořený virtuální WiFi adaptér. Toto nastavení se provede v Síťových připojeních ve vlastnostech síťového adaptéru připojeného k internetu.
4. spustím virtuální adaptér a připojím do WiFi sítě mobilní zařízení. Spuštění virtuálního adaptéru se provede příkazem `netsh wlan start hostednetwork`.

Konfigurace PC, která byla použita:

- Microsoft Windows 10 Pro 64-bit
- Broadcom 802.11ac, WiFi síťová karta
- Wireshark 1.12.[1]

Název zařízení	Zařízení	Operační systém	Verze viberu
Mobil 1	Samsung SIII (GT-9300)	Android 4.3	Viber 4.3.0 [2]
Mobil 2	Samsung SIII (GT-9300)	Android 4.3	Viber 5.7.0 [3]

Tabulka 2.1: Tabulka s použitými mobilními telefony



Obrázek 2.1: Schéma zapojení

2.2 Analýza zachycených dat

Nejdříve se budu zabývat analýzou komunikace ve Viberu verze 4 a popíšu obsah této komunikace a jaká data jsou důležitá z pohledu forenzní analýzy. Poté provedu srovnání, zda je možné získat stejný obsah komunikace i ve verzi 5.

Viber pro komunikaci využívá portu 80, 443, 4244, 5242, 5243, 9785 [4]. Zaměřím se na komunikaci probíhající na portu 80, který se používá pro odesílání a příjem multimediálních dat a aktuální polohy za pomocí protokolu HTTP.

[1] Dostupné z: https://1.na.dl.wireshark.org/win32/Wireshark-win32-1.12.9.exe

[2] Dostupné z: http://www.apkmirror.com/apk/viber-media-s-a-r-l/viber/viber-4-3-0-712-android-apk-download/

[3] Dostupné z Google Store

HTTP (HyperText Transfer Protocol) je aplikační protokol, pomocí kterého klient a server mezi sebou mohou komunikovat. Protokol HTTP funguje nad TCP. První verzí tohoto protokolu byla verze 0.9, která je popsána v RFC 1945 [6]. V této verzi protokolu klient mohl pouze zaslat požadavek na server, jenž požadavek zpracoval a poslal požadovaný dokument nebo zprávu o chybě. Od verze 1.0 (RFC 1945 [6]) lze serveru zasílat atributy souboru a to podle standardu MINE (Multipurpose Internet Mail Extensions) popsaného v RFC 1521, který již sloužil pro přenos dokumentů elektronickou poštou.

Protokol HTTP verze 1.1 (RFC 2616 [8]) definuje spoustu požadavků (POST, GET, PUT, DELETE, ...). Pro analýzu jsou důležité hlavně tyto dva GET a POST. Požadavek GET slouží k získání požadovaného dokumentu. Tento požadavek je důležitý při získávání aktuální polohy a při příjmu multimediálního obsahu. Druhý požadavek, který bude důležitý pro zkoumání, je POST. Pomocí požadavku POST je klient schopný předat serveru data. U tohoto požadavku klient musí správně určit délku přenášených dat hlavičkou Content-Length, aby server byl schopen zjistit, kde končí přenášený soubor. Tento požadavek je využíván k odesílání multimediálních dat. [8]

Na obrázku 2.2 jsou zobrazena data HTTP protokolu, která Viber odesílá, při oznamování *aktuální polohy*. Viber odešle dotaz serveru Googlu požadavkem GET pro získání obrázku s aktuální polohou, ve kterém určuje zeměpisné souřadnice polohy, rozměr obrázku, měřítko a přiblížení. V odpovědi na dotaz je zasílán obrázek ve formátu PNG s požadovanými parametry. Z hlavičky také může být zjištěna informace o zařízení, které uživatel použil pro komunikaci. Zde se dá najít například verze operačního systému zařízení či výrobce a model zařízení. *Odesílání dat* je zajištěno pomocí požadavku POST na adresu

```
Hypertext Transfer Protocol
⊞ GET /maps/api/staticmap?zoom=16&size=400x259&scale=%3Cscale%3E&maptype=roadmap&sensor=true&center=49.374215,16.017653 HTTP/1.1\r\n
  User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.3; GT-I9300 Build/JSS15J)\r\n
  Host: maps.google.com\r\n
  Connection: Keep-Alive\r\n
  Accept-Encoding: gzip\r\n
  \r\n
```

Obrázek 2.2: Požadavek na získání obrázku mapy

share.viber.com/upload.php. To můžeme vidět na obrázku 2.3. Dále z tohoto obrázku můžeme odvodit, že zpráva se dělí na čtyři části a to na tři textové a jednu binální. Textové části mají kódování UTF-8. První textová část obsahuje unikátní identifikátor souboru, druhá textová část obsahuje sekvenční číslo a třetí textová část obsahuje formát odesílaného souboru. V binární části se nacházejí data odesílaného souboru. Zde také opět můžeme nalézt v hlavičce popis zařízení.

```
Hypertext Transfer Protocol
⊞ POST /upload.php HTTP/1.1\r\n
  Content-Type: multipart/form-data; boundary=151f35f035c8ae1cef6672d8c4\r\n
  X-Filesize: 27628\r\n
  X-Checksum: b9e0c184de93e0c62dbb63dad3b3a860\r\n
  X-Offset: \r\n
  User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.3; GT-I9300 Build/JSS15J)\r\n
  Host: share.viber.com\r\n
  Connection: Keep-Alive\r\n
  Accept-Encoding: gzip\r\n
⊞ Content-Length: 28247\r\n
  \r\n
  [Full request URI: http://share.viber.com/upload.php]
  [HTTP request 3/4]
  [Response in frame: 18052]
MIME Multipart Media Encapsulation, Type: multipart/form-data, Boundary: "151f35f035c8ae1cef6672d8c4"
  [Type: multipart/form-data]
  First boundary: --151f35f035c8ae1cef6672d8c4\r\n
⊟ Encapsulated multipart part:  (text/plain)
    Content-Disposition: form-data; name="UDID"\r\n
    Content-Type: text/plain; charset=utf-8\r\n\r\n
  ⊟ Line-based text data: text/plain
      c44eb502bae850eaac1b9c547bc038ae6288cb87
  Boundary: \r\n--151f35f035c8ae1cef6672d8c4\r\n
⊟ Encapsulated multipart part:  (text/plain)
    Content-Disposition: form-data; name="Seq"\r\n
    Content-Type: text/plain; charset=utf-8\r\n\r\n
  ⊟ Line-based text data: text/plain
      1056296009
  Boundary: \r\n--151f35f035c8ae1cef6672d8c4\r\n
⊟ Encapsulated multipart part:  (text/plain)
    Content-Disposition: form-data; name="Filetype"\r\n
    Content-Type: text/plain; charset=utf-8\r\n\r\n
  ⊟ Line-based text data: text/plain
      jpg
  Boundary: \r\n--151f35f035c8ae1cef6672d8c4\r\n
⊟ Encapsulated multipart part:  (application/octet-stream)
    Content-Disposition: form-data; name="File"; filename="file"\r\n
    Content-Type: application/octet-stream\r\n
    Content-Transfer-Encoding: binary\r\n\r\n
  ⊞ Media Type
  Last boundary: \r\n--151f35f035c8ae1cef6672d8c4--\r\n
```

Obrázek 2.3: Odesílání obrázku

Proces *příjmu dat* se sestává z dotazu klienta na multimediální data, která jsou uložená na serveru (viz. obrázek 2.4) a odpovědi serveru na dotaz (viz. obrázek 2.5). Dotaz se zasílá pomocí požadavku POST na adresu *share.viber.com/download.php*. V dotazu se vyskytují tři položky, a to položka bucked, která se skládá z share plus data nahrání souboru. Další položkou je id. Tato položka obsahuje identifikátor souboru. Poslední položkou je formát požadovaného souboru. Po obdržení požadavku server zašle zprávu s adresou na požadovaný soubor, která je veřejně přístupná.

```
Hypertext Transfer Protocol
⊞ POST /download.php HTTP/1.1\r\n
  Content-Type: application/x-www-form-urlencoded;charset=utf-8\r\n
⊞ Content-Length: 103\r\n
  User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.3; GT-I9300 Build/JSS15J)\r\n
  Host: share.viber.com\r\n
  Connection: Keep-Alive\r\n
  Accept-Encoding: gzip\r\n
  \r\n
  [Full request URI: http://share.viber.com/download.php]
  [HTTP request 1/4]
  [Response in frame: 17494]
HTML Form URL Encoded: application/x-www-form-urlencoded
⊞ Form item: "Bucket" = "Share2015-12-30"
⊞ Form item: "ID" = "0224a0e092810ca7db972da7e3a28a69aa6ba20a5927f212bb6a93adf5b37f05"
⊞ Form item: "Filetype" = "jpg"
```

Obrázek 2.4: Dotaz na získání požadovaného obrázku

```
Hypertext Transfer Protocol
⊞ HTTP/1.1 302 Moved Temporarily\r\n
  Content-Type: text/html\r\n
  Date: Wed, 30 Dec 2015 14:49:38 GMT\r\n
  Location: https://share-vb.cdn.viber.com/0224-share2015-12-30/0224a0e092810ca7db972da7e3a28a69aa6ba20a5927f212bb6a93adf5b37f05.jpg\r\n
  Server: nginx\r\n
  X-Stat-Server: share55\r\n
⊞ Content-Length: 0\r\n
  Connection: keep-alive\r\n
  \r\n
  [HTTP response 1/4]
  [Time since request: 0.153973000 seconds]
  [Request in frame: 17486]
  [Next response in frame: 17495]
```

Obrázek 2.5: Odpověď od serveru s odkazem na požadovaný obrázek

SSL (Secure Socket Layer) je protokol, který poskytuje zabezpečení a spolehlivost komunikace mezi dvěma aplikacemi. Tento protokol funguje nad TCP. SSL se skládá ze dvou vrstev:

- SSL Handshake Protokol
- SSL Record Protokol

Vrstva SSL Handshake Protokol zajišťuje autentifikaci mezi dvěma uzly. Při autentifikaci se určí algoritmus, pomocí kterého bude komunikace šifrována a klíče, pomocí kterých budou data šifrována a dešifrovaná. Autentifikace probíhá před prvním odesíláním či příjmem dat. [9]

Vrstva SSL Record Protokol se stará o zapouzdření dat aplikačních protokolů (např. HTTP, Telnet, . . .).

Na závěr této kapitoly bylo provedeno *porovnání verzí 4 a 5* z pohledu forenzní analýzy. Z pohledu forenzní analýzy lze ve verzi 4 získat spoustu zajímavých informací. Z příjmu a odesílaní multimediálních dat je možné zjistit například obrázky nebo videa, která byla přenášená, dále je možno určit, kdy byla tato data přenášena a informace o zařízení, které uživatel pro komunikaci použil. V těchto informacích o zařízeních je možno nalézt operační systém, který zařízení používá, výrobce a model zařízení. Z aktuální polohy je možné opět určit, kdy informace o poloze byly odeslány či přijaty, jaké zařízení bylo použito pro komunikaci a zeměpisné souřadnice, kde se odesílatelovo zařízení zrovna nachází.

Naproti tomu ve Viberu verze 5 jsem zjistil, že odesílání a příjem multimediálních dat je již chráněno šifrováním a tato komunikace se přesunula z portu 80 z největší pravděpodobností na port 443, protože se zvětšil objem posílaných dat přes tento port. Na základě článku *Viber Communication Security: unscramble the scrambled* [14], ve kterém bylo zjištěno pomocí reverzního inženýrství aplikace Viber, že tato aplikace komunikuje přes protokol SSL.

Kapitola 3

WhatsApp

Whatsapp je velmi populární komunikační mobilní aplikace, kterou v současné době využívá asi 990 milionů uživatelů [19]. Whatsapp umožňuje posílat textové zprávy, obrázky, videa a volat.

V této kapitole rozeberu variantu šifrovacího algoritmu RC4, kterou používá aplikace k šifrování komunikace. Dále popíšu autentifikační procedury a signalizační protokol pro volání.

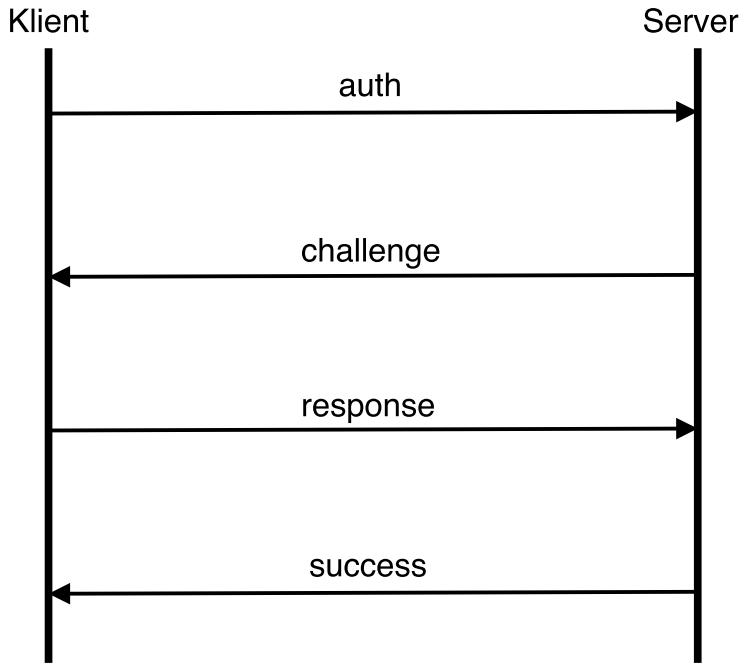
3.1 Autentifikace

Při autentifikaci pomocí procedury *full handshake* klient zašle zprávu auth na server. Tato zpráva není šifrovaná a obsahuje telefonní číslo klienta a autentifikační metodu, kterou klient používá. Poté mu server odpoví zprávou challenge, která obsahuje 20 bajtový řetězec pro vygenerování klíče pro šifrování komunikace. Klíče pro šifrování se generují za pomocí Password-Based Key Derivation Function 2 (PBKDF2) algoritmu, který používá heslo a 20 bajtový řetězec. Server i klient si generují klíče z hesla a 20 bajtového řetězce a vygenerované klíče jsou u serveru i klienta totožné. Celkem se generují čtyři klíče. Pro každý směr se generuje jeden klíč (pro směr od klienta k serveru a pro směr od serveru ke klientovi). Pro kontrolu integrity se vygenerují další dva klíče (opět jeden pro každý směr).

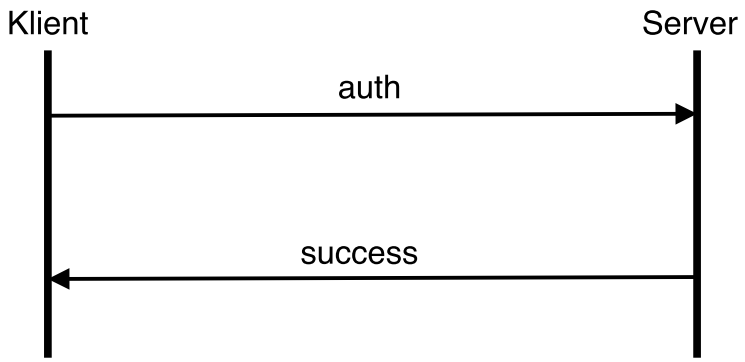
Následně klient vytvoří zprávu response, která je složena z telefonního čísla klienta v ASCII, 20 bajtového řetězce, který mu poslal server v binární podobě, časového razítka aktuálního času v ASCII a dalších dat, která popisují zařízení klienta (verzi OS Android, výrobce telefonu, model telefonu, ...). Tato zpráva je zašifrována pomocí vygenerovaných klíčů a předřadí se hash pro kontrolu integrity.

V případě, že registrace proběhne úspěšně, server odpoví zprávou success, která je šifrovaná, jinak server odpoví zprávou failure, která není šifrovaná. [11] Celý průběh full handshake autentifikace, jak jdou zprávy za sebou, můžeme vidět na obrázku 3.1.

Half handshake se skládá z auth zprávy, která obsahuje data z response zprávy a server odpoví success zprávou. Na průběh autetifikace se můžeme podívat na obrázku 3.2. Jelikož se při komunikaci neposílá zpráva challenge, ve které je 20 bajtový řetězec, pomocí kterého se vygenerují klíče pro šifrování komunikace, protože klient používá tento řetězec již z dřívější komunikace. Z tohoto důvodu nejsem schopen komunikaci dešifrovat, protože není možné určit klíče, které se používají k šifrování komunikace. [11]



Obrázek 3.1: Full handshake



Obrázek 3.2: Half handshake

3.2 Signalizace hovoru

WhatsApp klient se nejdříve připojí na server a provede autentifikaci. Jak můžeme vidět z průběhu signalizace hovoru na obrázku 3.3, tak po připojení na sever se klient zeptá na přítomnost volaného a pošle zprávu *offer* volanému a tím začne proces navazování hovoru. V této zprávě se poprvé vyskytuje vlastnost *call-id*. Tato vlastnost je stejná během celého procesu navazování hovoru a identifikuje hovor. Zpráva *offer* má vlastnost *audio*, ze které se dá zjistit, jaký kodek pro přenos zvuku volající nabízí. V tomto případě je to Opus, který

zvuková data vzorkuje pomocí dvou vzorkovacích frekvencí a to 8 kHz a 16 kHz. Dále také zahrnuje vlastnosti *p2p* a *srtp*, u kterých nejsem schopen určit význam. Tyto vlastnosti by mohli sloužit jako inicializační vektory pro šifrování zvukové komunikace anebo k popisu tohoto spojení. Poslední vlastnost, která se zde dá najít, je *te*, která obsahuje 6 bajtovou hodnotu, která určuje koncovou adresu (IP adresa a port), kterou klient používá pro mediální přenos.

Server odpoví zprávou *ack relay*, ve které se nachází vlastnost *token*, jejíž význam je neznámý. Dále obsahuje několik vlastností *te*, které oznamují koncové adresy přenosových serverů. Ještě se zde také vyskytují vlastnosti *encode*, *agc* a *ns*, které zřejmě slouží pro specifikaci kódování mediálního přenosu.

Také mohou být zachyceny zprávy příjmu *receipt* a *ack*. V těchto zprávách se nevyskytují pro nás žádné zajímavé informace.

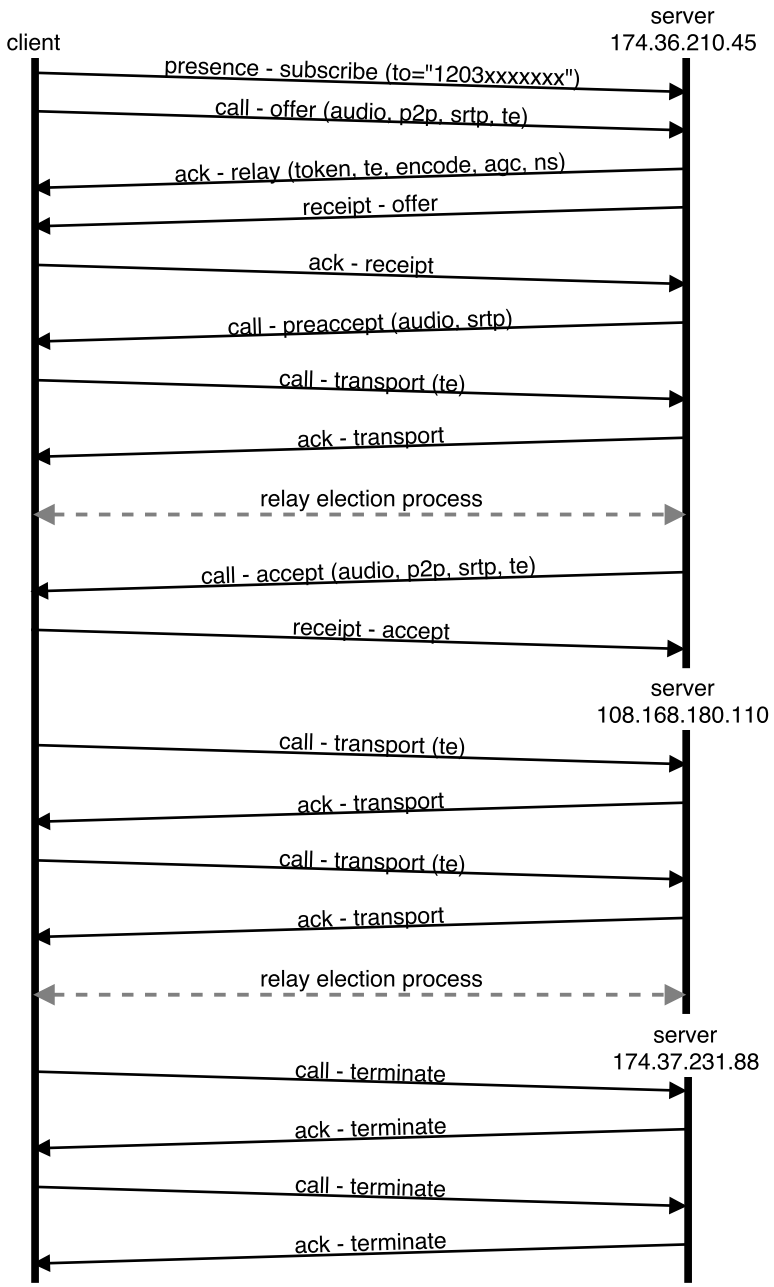
Od serveru přijde ke klientovi zpráva *preaccept*, která pomocí vlastnosti *audio* určí, jaký bude používat kodek pro zvukový přenos a jakou vzorkovací frekvenci pro vzorkování přenášeného zvuku. Je zde nalezena i vlastnost *srtp*, která má stejnou velikost jako ve zprávě *offer*, ale může nést jinou hodnotu.

Zpráva *transport* obsahuje klientskou veřejnou koncovou adresu. Pokud klient používá Traversal Using Relays around NAT (TURN) mechanismus, tak se zeptá TURN serveru jaká je jeho veřejná IP adresa. Její hodnota je odlišná od hodnoty udávané ve zprávě *offer*.

Potom následuje *ack* zpráva pro zprávu *transport*. Poslední zprávou v procesu navazování spojení je zpráva *accept*, která obsahuje vlastnost *audio*, pomocí které potvrzuje klient používání kodeku a vzorkovací frekvence. Ve zprávě se vyskytují vlasnosti *p2p*, *srtp* a dvě koncové adresy privátní a veřejná. Adresy jsou používané pro ustanovení přímého peer-to-peer (P2P) spojení. Zpráva *receipt* potvrzuje zprávu *accept*.

Následně je vytvořeno oboustranné spojení od volajícího k volanému, které je uskutečněno pomocí RTP. Adresy zjistím ze zprávy *offer* během volby přenosového serveru. Po 30 sekundách se klient spojí s jiným serverem a začne posílat signalizační zprávy přes tento server. Dále klient oznámí nové koncové adresy a po nové volbě přenosového serveru vytvoří nové spojení pro mediální přenos, které nahradí předchozí spojení za spojení s novou koncovou adresou.

Pokud si klient přeje ukončit hovor, tak pošle dvě identické zprávy *terminate* a hovor je ukončen. [11]



Obrázek 3.3: Signalizace volání ve WhatsApp [11]

3.3 PBKDF2

PBKDF2 (Password-Based Key Derivation Function 2) je funkce určená pro generování pseudonáhodných klíčů pro šifrování dat nebo také integritních klíčů. Funkce je definována standardy PKCS#5 v2.0 a RFC2898. Vstupem funkce je heslo, sůl a počet iterací, které funkce musí provést, abych dostal požadované heslo. V programovacím jazyce C# existuje implementace této funkce pomocí třídy *Rfc2898DerivateBytes*. Třída se nachází v knihovně *System.Security.Cryptography*. Bohužel tato implementace vyžaduje, aby velikost hodnoty soli byla alespoň osm bajtů. V tomto případě je zapotřebí, aby velikost hodnoty soli mohla být menší, než oněch osm bajtů. Konkrétně je zapotřebí, aby hodnota soli byla o velikosti čtyř bajtů. A to z důvodu, že tuto velikost má hodnota soli při extrahování hesla aplikace WhatsApp ze souboru uloženého na odposlouchávaném mobilním zařízení. Řešením je vytvořit vlastní implementaci funkce nebo najít na internetu implementaci, která by vyhovovala výše uvedeným požadavkům. Implementaci, která se v této bakalářské práci používá, pochází ze serveru stackoverflow.com[1]. Použití třídy *Rfc2898DerivateBytes* je velmi jednoduché. Při vytváření instance zadáváme konstruktoru tři parametry a to heslo, sůl a počet iterací. Pro získání potřených dat se zavolá nad instancí třídy *Rfc2898DerivateBytes* metoda *GetBytes()*. Jediným parametrem metody je velikost požadovaných dat uváděná v bajtech.

3.4 Symetricté algoritmy

Symetrické šifrovací algoritmy používají pro šifrování a dešifrování stejný klíč. Na rozdíl od asymetrických šifer, které používají pro šifrování a dešifrování odlišné klíče. Výhodou symetrických šifer oproti asymetrickým šifrám je, že nejsou tak náročné na výpočetní výkon. Nevýhodou je ovšem použití sdíleného klíče, pomocí kterého se šifrují a dešifrují data. Problém je, že se obě strany, příjemce i odesílatel, musí nejprve na tomto klíči domluvit.

Existují dvě skupiny symetrických šifer. První skupinou jsou proudové šifry, které šifrují po jednotlivých bitech. Druhou skupinou jsou blokové šifry, které naopak šifrují po blocích o pevně dané velikosti.

Proudové šifry jsou velmi důležitou skupinou šifrovacích algoritmů. Jsou schopny šifrovat jednotlivé znaky (obvykle v binární podobě) vstupní zprávy jeden po druhém. K tomu se využívá šifrovacích transformací, které se během šifrování mění. Na rozdíl od blokových šifrovacích algoritmů, které šifrují po skupinách znaků vstupní zprávy se během šifrování transformace nemění, je tedy fixní. Proudové šifry jsou obecně rychlejší než blokové šifry při hardwarové implementaci, protože pro implementaci použijí pár jednoduchých elektronických obvodů (například logická funkce XOR, ...). Proudové šifry jsou vhodné pro případy, kdy se data musejí zpracovávat ihned, protože je nelze ukládat do vyrovnávací paměti, ať už je omezena velikost vyrovnávací paměti nebo se při zpracování dat klade důraz na rychlost přenosu dat, například při telekomunikačních a jiných real-time přenosech. Další výhodou je, že proudové šifry mají omezenou nebo žádnou propagaci chyby. Tato vlastnost proudových šifer se velmi hodí v případě, že při přenosu dat dochází k chybám, například vlivem rušení.

Mezi zástupce skupiny proudových šifer patří RC4, FISH, Helix, SEAL, WAKE a další. Proudové šifry dělíme na dvě základní skupiny:

- synchronní proudové šifry

[1] Dostupné z: http://stackoverflow.com/questions/1046599/pbkdf2\-implementation-in-c-sharp-with-rfc2898derivebytes

- proudové šifry s vlastní synchronizací (asynchronní)

Synchronní proudová šifra je taková, která nevyužívá k šifrování text (původní ani šifrovaný). Data jsou šifrována pomocí klíče a stavu, ve kterém se funkce nachází. Při dešifrování je proto potřeba mít stejný klíč a postupně procházet ekvivalentními stavy. Pokud se při přenosu ztratí jediný bit (nebo přibude), pak se synchronizace ztratí. Pro zachování synchronizace se proto používají další mechanizmy.

Asynchronní proudové šifry využívají k šifrování i dešifrování mimo klíče i předchozích n bitů šifrovaného textu. Tím se zvyšuje bezpečnost, ale při ztrátě jednoho bitu se následujících n bitů zprávy dekóduje špatně. Po těchto n bitech se další data již dekódují opět správně. [13][1]

Symetrické blokové šifry jsou nejvýznamnějším a nejdůležitějším prvkem spousty kryptografických systémů. Samotné šifry poskytují možnost utajení. Jako základní stavební kámen umožňují konstrukci pseudonáhodných generátorů, proudových šifer, MAC (Message Authentication Code), hašovacích funkcí. Mohou také zastat roli ústřední komponenty pro techniky ověřování zpráv zajištění, integrity dat, digitální podpisy.

Žádná bloková šifra není natolik dokonalá, aby se dala použít ve všech případech, i když nabízí vysokou míru zabezpečení. Tato skutečnost vychází z toho, že každá aplikace vyžaduje jiné nároky určitých vlastností šifry. Těmito vlastnostmi mohou být například:

- rychlost šifrování
- velikost použité paměti (velikost kódu, velikost dat, ...)
- podmínky pro použití na určité platformě (hardware, software)
- použitý operační mód

Proto vždy musí být zvolen kompromis mezi efektivitou a mírou úrovně zabezpečení. Definice blokové šifry je následující. Bloková šifra je funkce, která převádí n-bitové vstupní bloky na n-bitové zašifrované bloky, kde n je délka jednoho bloku. Šifra může být vnímána jako jednoduchá substituční šifra s velkou velikostí znaku. Funkce je parametrizována k-bitovým klíčem K z množiny κ (klíčový prostor) všech možných k-bitových vektorů V. Obecně se klíč vybírá z této množiny náhodně. Použitím stejně velkých vstupních a výstupních bloků dat se předejde expanzi dat. Pro umožnění dešifrování musíme být schopni z šifrovací funkce vytvořit invertovanou funkci.

Nyní si řekneme něco o operačních módech. Blokové šifry šifrují vstupní data po blocích o pevně dané velikosti n. Pokud zpráva je větší než velikost bloku n, tak nejjednodušším řešením je zprávu rozdělit do bloků o velikosti n a šifrovat odděleně. Jednotlivé módy pak určují způsob šifrování. Nejběžnějšími módy jsou:

- ECB (electronic-codebook)
- CBC (cipher-block chaining)
- CFB (cipher feedback)
- OFB (output feedback)

Nyní se budu věnovat pouze popisu módu OFB, jelikož tento mód je využíván při extrahování hesla. Mód OFB je využíván aplikacemi, které musí předcházet chybám vzniklých přenosem. OFB mód je podobný módu CFB, který umožňuje šifrovat bloky o různé velikosti. Rozdíl je v tom, že blokový výstup šifrovací funkce slouží jako zpětná vazba. [13][12]

3.5 AES

AES (Advanced Encryption Standard) je symetrická bloková šifra, což znamená, že se pro šifrování a dešifrování používá stejný klíč a data se šifrují a dešifrují po blocích o pevně dané délce. AES šifra je rychlá v softwaru i hardwaru a na rozdíl od svého předchůdce DES nepoužívá Feistelovu síť. V současné době se šifra využívá například k zabezpečení komunikace na Wi-Fi sítích pomocí WPA2 (Wi-Fi Protected Access).

Belgičané Joada Daemena a Vincenta Rijmena jsou autoři této šifry, jejíž původní název zněl Rijndael. Název vznikl z přesmyčky jmen autorů šifry. Americký úřad pro standardizaci vyhlásil v roce 1997 veřejnou soutěž o federální šifrovací algoritmus AES. Do této soutěže byla přihlášena i šifra Rijndael, kterou v roce 2001 americký úřad pro standardizaci vyhlásil, jako vítězný návrh z patnácti přihlášených. AES je první šifra, jejíž specifikace je volně přístupna veřejnosti a zároveň byla používanou šifrou NSA (Národní bezpečnostní agenturou) k šifrování tajných dokumentů. V roce 2002 začala být používána jako federální standard v USA.

Ještě by nebylo od věci zmínit něco o rozdílech mezi šifrou AES a původním algoritmem Rijndael. U původní šifry Rijndael jsou velikost bloku a velikost klíče určeny násobky 32 bitů s minimální velikostí 128 bitů a maximální velikostí 256 bitů. U šifry AES je velikost bloku omezena pouze na 128 bitů a velikost klíče může nabývat pouze hodnoty 128, 192 nebo 256 bitů. [2]

3.6 RC4

RC4 (Rivest Cipher 4) je velmi populární symetrická proudová šifra. Šifru navrhl Ron Rivest v roce 1987 ve společnosti RSA Security. Specifikace šifry RC4 nebyla nikdy publikovaná oficiálně. V roce 1994 neznámý hacker zveřejnil její princip, který získal reverzním inženýrstvím programu BSafe společnosti RSA Security. Specifikaci anonymně zaslal emaily celé skupině Cypherpunks. Brzy někdo zveřejnil specifikaci na Sci.crypt, odkud se rozšířila na další internetové stránky.

Jelikož název RC4 je chráněn obchodní známkou, tak vznikl problém, jak tento algoritmus nazývat. Proto vznikly názvy ARCFOUR nebo ARC4 (Alleged RC4), které problém s obchodní známkou vyřešily. Délka klíče pro šifrování může být teoreticky v rozpětí od 1 bajtu do 256 bajtů. Stav RC4 šifry se uchovává v tzv. sboxu a dvou indexech i a j. [10]

Mezi velké výhody této šifry patří velmi jednoduchá implementace, ať už v hardwaru nebo v softwaru, a také její rychlost. Používá se například v TLS nebo pro zabezpečení bezdrátového přenosu pomocí WEP. Šifru používá pro přenos dat i WhatsApp, nicméně ne klasickou RC4, ale její modifikovanou verzi, kterou se bude tato kapitola zabývat. [18]

Variantu RC4 šifry, kterou využívá WhatsApp pro šifrování komunikace bude popsána pomocí algoritmu 1. Pro dešifrování jsem si vytvořil třídu DekoderRC4. V konstruktoru této třídy se provede potřebná inicializace a funkce *desifruj* zajišťuje dešifrování požadovaných dat.

Konstruktor nejprve provede inicializaci pole s, které má velikost 256 bajtů a to hodnotami od 0 do 255 vzestupně. Po inicializaci pole s dojde k promíchání prvků pole s. K promíchání prvků pole s je zapotřebí klíč, který je v tomto případě 20 bajtový. Jako poslední krok inicializace se provede vytvoření pomocného pole o velikosti, která může nabývat hodnoty 256 nebo 768. Následně dojde k naplnění tohoto pomocného pole. Po naplnění pomocného pole se použije jako vstup na funkci *desifruj*. Dešifrováním pomocného pole se

způsobí, že se pole s opět promíchá. Toto je jediný rozdíl, ve kterém se tato varianta liší od klasické verze RC4.

Při dešifrování se nejdříve provede prohození dvou prvků v poli s a následně se provede dešifrování velice jednoduchým způsobem, a to logickou funkcí XOR nad jedním bajtem vstupních dat a jedním bajtem pole s. Tato akce probíhá, dokud není celý vstup dešifrován.

Algoritmus 1: Třída Dekoder_RC4

```
s[256]
i = j = 0
Dekoder_RC4(klic, delka_klice, z, zd)
    for k = 0 to 255 do
        s[k] = k
    end for
    for k = 0 to 255 do
        k = klic[i mod delka_klice]
        j = (j + k + s[i]) mod 256
        prohod(i, j)
    end for
    i = j = 0
    pom[z]
    for k = 0 to z − 1 do
        if zd then
            pom[k] = 0
        else
            pom[k] = k

        end if
    end for
    desifruj(pom, z)

desifruj(data, delka)
    d = 0
    while dokud neni konec vstupnich dat do
        i = (i + 1) mod 256
        i = (j + s[i]) mod 256
        prohod(i, j)
        index = s[i] + s[j] mod 256
        vystup = data[d] XOR s[index]
        d = d + 1
    end while
    return vystup

prohod(i, j)
    pom = s[i]
    s[i] = s[j]
    s[j] = pom
```

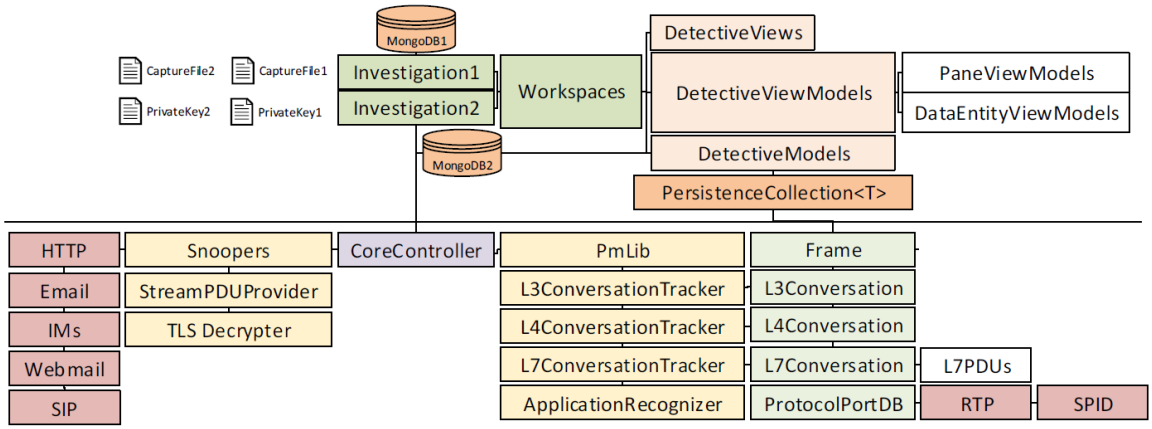
Kapitola 4

Implementace

4.1 Netfox Detective

Netfox Detective je nástroj pro forenzní analýzu síťového komunikace, který umožňuje snadnou extrakci obsahu z aplikačních protokolů. Aplikace je vyvíjena v rámci projektu SEC6NET na Fakultě informačních technologií Vysokého učení technického v Brně.

Netfox Detetective využívá pro zpracování vstupních PCAP souborů knihovnu *PmLib*. Knihovna podporuje spoustu formátů pro zaznamenávání síťové komunikace. Mezi podporovanými formáty jsou například Wireshark/TCPDump LibCap, Microsoft network monitor cap - verze 2.0 a PCAP-ng pcap - verze 1.0. *PmLib* zpracovává rámce do vyšších vrstev (L2, L3 a L4). K tomu používá PaketDotNet 0.13, který podporuje PPP, PPPoE, Ethernet, IP, UDP, TCP, atd. [17][16]



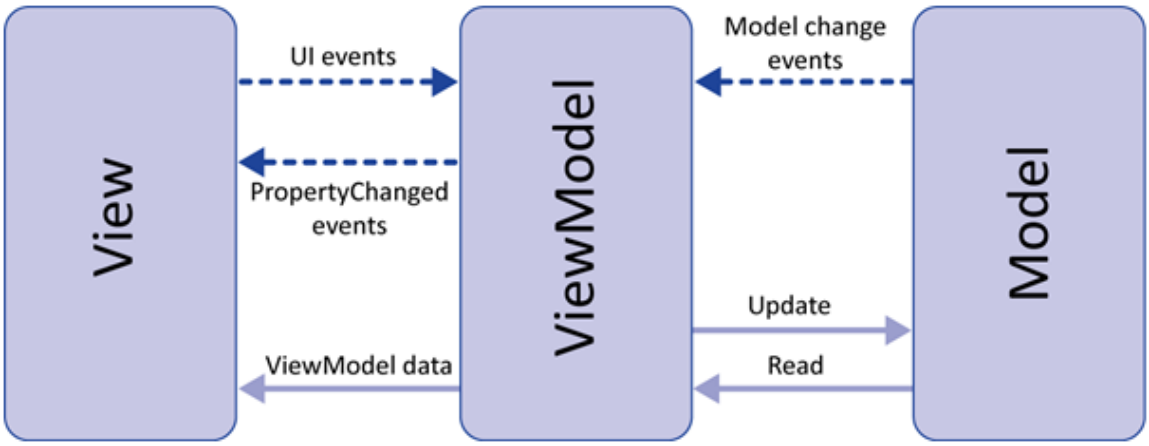
Obrázek 4.1: Archtektura a data model Netfox Detective [17]

Netfox Detective lze spouštět na platformě Windows 7 a novějších verzí. K vývoji aplikace je využit návrhový vzor MVVM (Model-View-ViewModel). MVVM (Model-View-ViewModel) je návrhový vzor pro WPF (Windows Presentation Forms) aplikace. Nabízí řešení, jak oddělit logiku aplikace od uživatelského rozhraní. Kódu je pak méně, vše je přehlednější a případné změny nejsou implementační noční můrou. MVVM odděluje data, stav aplikace a uživatelské rozhraní. Samotné WPF bylo vytvořeno tak, aby se v něm MVVM používal pohodlně. Proto se v něm využívá binding a command - náhrada za uživatelské roz-

hraní řízené událostmi. Netfox Detective lze spouštět na platformě Windows 7 a novějších verzí. K vývoji aplikace je využit návrhový vzor MVVM (Model-View-ViewModel). MVVM (Model-View-ViewModel) je návrhový vzor pro WPF (Windows Presentation Forms) aplikace. Návrhový vzor umožňuje oddělit logiku aplikace od uživatelského rozhraní. Díky tomuto faktu se poté aplikace snadněji udržuje v případě změny a zdrojové kódy jsou přehlednější. MVVM rozdělí aplikaci na vrstvy. Každá z těchto vrstev má na starost jinou věc (data, stav aplikace, uživatelské rozhraní)[7][17].

Nyní si uvedeme, co mají jednotlivé vrstvy na starost[7]:

- Model je vrstva, která je zodpovědná za data, se kterými aplikace pracuje. Tato vrstva neví, v jakém stavu se aplikace nachází.
- View se stará o uživatelské rozhraní aplikace. Uživatelské rozhraní se definuje pomocí jazyka XAML. Tato vrstva může definovat, jak bude vypadat webová stránka, okno aplikace či ovládací prvek.
- ViewModel je nejdůležitější vrstvou, protože propojuje vrstvy Model a View a uchovává stav aplikace. Ovládací prvky vrstvy View jsou propojeny s ViewModelem a dostávají od něj svá data. zda se také může provádět filtrování dat podle stavu aplikace.



Obrázek 4.2: Vztahy mezi vrstvami View, Viewmodel, Model [5]

Pro zpracování dat nacházejících se v aplikační vrstvě síťového modelu slouží dynamické knihovny, které se nazývají tzv. Snooper. Každá z těchto dynamických knihoven má na starost zpracování nějakého aplikačního protokolu. V současné době jsou v Netfox Detective naimplentovány knihovny pro protokoly HTTP, POP, IMAP, SMTP, SIP, RTP, FTP, Facebook, atd. [17]

Cílem této bakalářské práce je naimplementovat knihovnu aplikačního protokolu komunikační aplikace WhatsApp, a to pro verze protokolu 1.5, 1.4, 1.3 a 1.2. Jelikož se jedná o proprietární protokol, který se neustále vyvíjí, tak je možné, že se během psaní bakalářské práce mohou objevit novější verze protokolu, které nebudou kompatibilní s implementací, což znamená, že nebude možné je zpracovat.

4.2 Extrahování hesla

Aby bylo možné vygenerovat klíče pro dešifrování komunikace ze zprávy *challenge* při autentifikaci, je nutné znát heslo, které je uloženo v mobilním zařízení. Heslo se dá zjistit například pomocí aplikace WA PW extractor[1]. Velkou nevýhodou aplikace je to, že vyžaduje práva správce a tyto práva většinou nejsou k dispozici. Když tedy práva správce nejsou k dispozici, tak exituje možnost extrahovat heslo ze souboru uloženém na mobilním zařízení pomocí následujícího postupu. [3]

Nejprve je zapotřebí stáhnout soubor z mobilního zařízení, ze kterého se bude extahovat heslo. Cesta k souboru, kde se soubor nachází, je */data/data/com.whatsapp/files/pw*. Soubor se stáhne pomocí nástroje ADB (Android Debug Bridge), ve kterém se zadají tyto příkazy:

1. `adb shell`
2. `su`
3. `cp /data/data/com.whatsapp/files/pw /sdcard`
4. `exit`
5. `exit`
6. `adb pull /sdcard/pw`

Nyní když je soubor stažený z mobilního zařízení, tak z něho musí být vybrány tři hodnoty (klíč, sůl, inicializační vektor), které jsou zapotřebí pro extrahování hesla. V souboru se tyto hodnoty nacházejí od 30. bajtu a jsou přímo za sebou. První 4 bajtová hodnota určuje sůl, následující hodnota je 16 bajtová a určuje inicializační vektor, poslední hodnota je 20 bajtová a určuje klíč. K extrahování je ještě nutné vytvořit hodnotu, která bude vstupem pro algoritmus PBKDF2 (Password-Based Key Derivation Function 2). Hodnota se skládá z 30 bajtové konstanty a z telefonního čísla.

Poté co je vše připraveno, tak se může začít extrahovat heslo. K tomu je využit algoritmus PBKDF2, jehož vstupem bude vytvořená hodnota. Dále určí počet iterací. V tomto případě to bude 16 iterací. Ještě posledním vstupním parametrem je sůl, která byla zjištěna ze souboru. Na výstupu získáme 16 bajtovou hodnotu. Následně se použije šifra AES (Advanced Encryption Standard). Vstupními parametry jsou vstupní data, inicializační vektor a klíč. Jako vstupní data je klíč, který byl získán ze souboru, stejně tak inicializační vektor je ze souboru, klíčem je hodnota výstupu předešlého algoritmu PBKDF2.

Pro blokovou šifru AES v jazyce C# existují dvě varianty implementace třída *AesManaged* a třída *RijndaelManaged*. Obě třídy se vyskytují v knihovně *System.Security.Cryptography*. Rozdíl mezi těmito třídami je v tom, že třída *AesManaged* podporuje bloky o velikosti 128 bitů, na rozdíl od třídy *RijndaelManaged*, která podporuje i bloky o velikosti 128, 160, 192, 224 a 256 bitů. V této bakalářské práci je použita třída *RijndaelManaged*. U této třídy je následně potřeba nastavit vlastnost *IV*, která určuje inicializační vektor šifrovacího algoritmu a vlastnost *Key*, která určuje šifrovací klíč. Dále ještě se nastaví vlastnost *Mode*, která určuje, s jakým módem bude algoritmus pracovat. Zde je nastaven mód OFB. Kvůli tomuto módu jsem si musel vytvořit vlastní třídu *OFBStream*, protože v jazyce C#

[1] Dostupné z: https://www.mgp25.com/downloads/pw.apk

není podpora pro tento mód. *OFBStream* je převzata ze serveru stackoverflow[2]. Třída je potomkem abstraktní třídy *Stream*. K implementaci této třídy je nutné ještě vytvořit třídu *ZeroStream*, která je také převzata ze serveru stackoverflow[2]1 a potomkem abstraktní třídy *Stream*.

Konstruktor třídy *OFBStream* vyžaduje tři vstupní parametry. Prvním parametrem je objekt třídy *Memorystream*, který obsahuje zašifrovaná vstupní data. V tomto případě je to zašifrované heslo WhatsAppu. Druhým parametrem je objekt třídy *RijndaelManaged* s nastavenými výše uvedenými vlastnostmi. Poslední parametr je hodnota výčtového typu *CryptoStreamMode*, který určí, zda se bude do objektu třídy *OFBStream* zapisovat nebo se z něj bude číst. Při extrahování hesla je cílem dešifrovat heslo, tak je nutné nastavit mód pro čtení. Po vytvoření instance třídy *OFBStream* se dešifrovaný obsah získá tak, že se vytvoří nová instance třídy *MemoryStream*. Pomocí metody *CopyTo()* nad objektem třídy *OFBStream* se zkopírují dešifrovaná data do objektu *MemoryStream*, kde se s nimi může dále pracovat. Dešifrovaná data se převedou z objektu *MemoryStream* na bajtové pole. Toto bajtové pole se dále ještě musí převést na textový řetězec kódováním base64.

4.3 Generování klíčů pro dešifrování

Generování klíčů probíhá při zpracování zprávy *challenge* při autentizaci. Ve zprávě se nachází 20 bajtová hodnota. Tato hodnota se nazývá tzv. sůl. K tomu, aby bylo možné začít generovat klíče, je ještě potřeba znát heslo, jehož délka je také 20 bajtů.

Heslo lze zadat dvěma způsoby, a to buď již extrahované v textové podobě, nebo v podobě souboru, který byl stažen z odposlouchávaného mobilního zařízení. Z tohoto souboru bude heslo extrahováno způsobem, který je popsán zde 4.2. Heslo se musí nejprve převést z textové podoby do binární, protože heslo v textové podobě využívá kódování base64, které umožňuje binární data převést na textový řetězec obsahující tisknutelné znaky.

Dalším krokem je vytvoření pomocného bajtového pole o velikosti o jeden bajt větší než je délka hodnoty soli. Velikost je tedy 21 bajtů. Do pomocného pole jsou zkopírovány hodnoty soli. Hodnota posledního bajtu pomocného pole se bude v průběhu generování měnit, v závislosti na tom, jaký klíč se zrovna bude generovat. Celkem se generují čtyři klíče:

- klíč pro dešifrování komunikace ve směru od klienta k serveru; pro tento klíč je hodnota posledního bajtu v pomocném poli *0x01*
- klíč pro dešifrování komunikace ve směru od serveru ke klientovi; zde je hodnota posledního bajtu v pomocném poli *0x03*
- klíč pro kontrolu integrity ve směru k serveru; hodnota posledního bajtu v pomocném poli *0x02*
- klíč pro kontrolu integrity ve směru ke klientovi; hodnota posledního bajtu v pomocném poli *0x04*

Poté, co se nastaví příslušná hodnota posledního bajtu pomocného pole pro získání požadovaného klíče, se použije algoritmus PBRDF2 (Password-Based Key Derivation Function 2).

[2]Dostupné z: http://stackoverflow.com/questions/24515521/aes-ofb-encryption-for-rijndael\managed

Vstupními parametry jsou heslo, které bylo na začátku převedeno z textového řetězce na binární data a pomocné pole, které reprezentuje parametr sůl pro tento algoritmus. Posledním důležitým parametrem je počet iterací, které algoritmus má provést. V tomto případě to budou 2 iterace. Výstupem bude 20 bajtová hodnota požadovaného klíče. Celý postup je ještě naznačen pomocí pseudokódu Algoritmu 2. Při implementaci se pro práci s klíči používá třída *RC4Keys*, která obsahuje čtyři hodnoty. Každá z těchto hodnot reprezentuje jednotlivý klíč.

Algoritmus 2: Funkce pro generování klíčů pro dešifrování

```
GenerovatRC4Klice(sul, delka_soli)
    tmp[delka_soli + 1]
    klice[4]
    pocet_iteraci = 2
    for i = 0 to delka_soli do
        tmp[i] = k
    end for
    for i = 1 to 4 do
        tmp[delka_soli] = i
        klice[i − 1] = PBKDF2(tmp, delka_soli + 1, pocet_iteraci)
    end for
    return klice
```

4.4 Zpracování zpráv

Zpracování zpráv protokolu aplikace WhatsApp obstarává třída *SnooperWhatsApp*. Rodičem této třídy je abstraktní třída *SnooperBase*. Ve třídě je nutné implementovat metodu *RunBody()*. Pomocí této metody se aplikace připravuje na zpracování přidělené konverzace. Tato metoda se volá ve všech třídách, které dědí ze třídy SnooperBase. Poté se zavolá metoda *ProcessConversation()*. Učelem metody je exportovat objekty, které reprezentují jednotlivé zprávy signalizace hovorů. Pro účely analýzy bude stačit, když budou exportovány zprávy *offer*, *accept*, *terminate*.

Jak metoda *ProcessConversation()* postupuje při zpracování konverzace, si nyní popíšeme. Metoda si nejprve vytvoří instance třídy *PDUStreamBasedProvider* a následně ještě instanci třídy *PDUStreamReader*. Instance těchto tříd je nutné vytvořit z důvodu, aby bylo umožněno číst jednotlivé zprávy konverzace. Poté může započít zpracování jednotlivých zpráv nacházejících se v konverzaci. V každé zprávě se může vyskytovat více než jedna zpráva protokolu aplikace WhatsApp.

Poté se u těchto zpráv zjišťuje, zda jsou zašifrované, jejich velikost, obsah, časovou značku a směr komunikace. Takže výstupem třídy *WAMsg* je seznam objektů třídy *WAMessage*, která reprezentuje zprávy protokolu aplikace WhatsApp. Dále také třída *WAMsg* určí, zda se jedná o validní zprávu nebo zprávy aplikačního protokolu aplikace WhatsApp. Pokud se nejedná o validní zprávu, přeruší se navazující zpracování zprávy a začne se zpracovávat následující zpráva v konverzaci znovu. V případě, že je zpráva validní, pokračuje se ve zpracování zprávy.

Pokud je zpráva šifrovaná, dojde k pokusu o její dešifrování. Při dešifrování záleží na směru komunikace, protože pro každý směr je generován jiný klíč, jak bylo zmíněno výše

viz 3.1. Pro dešifrování algoritmem RC4, který byl také popsán výše viz 3.6, se využívá třídy *DecoderRC4*. Pro každý směr komunikace je vytvořena jedna instance této třídy. Pro oba směry se objekty instalují při rozpoznání zprávy *challenge*. Pokud tedy se bude vyskytovat zašifrovaná zpráva před autentizací, v tom případě před zprávou *challenge* nejsou objekty pro dešifrovaní instancovány. V důsledku této skutečnosti nelze dešifrovat zprávu. Musí se tedy přeskočit následující zpracování zprávy a začít zpracovávat další zprávu v pořadí.

V případě, že se jedná o nešifrovanou zprávu nebo dešifrování proběhlo v pořádku, tak je možné nyní určit typ zprávy. Typy zpráv, které jsou důležité, jsou hlavně zprávy týkající se autentizace (*auth*, *challenge*, *response*, *success*). Tento typ zpráv je významný kvůli klíčům, které je nutné znát pro dešifrování. Dalším typem zpráv, který je důležitý, je signalizace hovorů (*call*, *ack*, *receipt*).

Nyní když je určen typ zprávy, dá se vytvořit objekt pro exportování signalizačních zpráv a postupně do něj doplňovat potřebné informace. Exportují se pouze zprávy, které patří do kategorie signalizace hovoru. Z této kategorie se neexportují zprávy týkající se volby přenosového serveru. Reprezentaci exportovaných objektů zajišťuje třída *WaCallMsg*. Třída *WaCallMsg* uchovává následující informace:

- call ID (identifikační číslo hovoru)
- typ signalizační zprávy (*call - offer*, *call - terminate*, *receipt - accept*, ...)
- telefonní číslo volajícího
- telefonní číslo volaného
- časové razítko, kdy byla zpráva posílána
- zdroje exportovaného objektu

Nyní si popíšeme, jak byly tyto informace získány a u některých bude vysvětlen blíže jejich význam. Call ID určuje jednoznačně hovor, ke kterému se vztahuje poslaná zpráva. Další informací je typ signalizační zprávy. K zjištění, o jaký typ signalizační zprávy se přesně jedná, se použije metoda *getTypeCallMsg()*. Zatím je pouze určeno, zda se jedná o zprávu *call*, *ack* nebo *receipt*. Tato informace byla již dříve zjištěna pomocí metody *getTypeMsg()*. Vstupem metody *getTypeCallMsg()* je jeden parametr a to typu *TypeMsg*, který určuje, zda se jedná o zprávu (*call*, *ack*, *receipt*). Výstupem metody je textový řetězec, který obsahuje typ signalizační zprávy. Čísla volajícího a volaného se nachází v jiných zprávách. Jedno z telefonních čísel se určí při autentizaci ze zprávy *auth* a druhé telefonní číslo ze signalizační zprávy. Určení významu, zda telefonní číslo získané ze zprávy *auth* je telefonním číslem volajícího a telefonní číslo získané ze signalizační zprávy je telefonním číslem volaného nebo jestli je tomu právě naopak, se zjistí pomocí atributů *From*, *To* a typu signalizační zprávy. Jelikož třída *WaCallMsg* je potomkem třídy *SnooperExportedObjectBase*, tak je nutné naplnit požadovanými hodnotami proměnné, které třída *WaCallMsg* zdědila. Jedna z proměnných, kterou třída zdědila a je nutné určit její hodnotu je proměnná *Timestamp*. Pro tuto proměnnou byla hodnota již zjištěna při zpracování zprávy konverzace pomocí třídy *WaMsg*. V tom případě je tato informace uložena ve vlastnosti *Timestamp* právě zpracovávaného objektu *WAMessage*. Poslední vlastností, která nesmí být před exportováním ponechána prázdná, je seznam *ExportSources*. Tento seznam říká, z jaké konverzace nebo datového rámce je signalizační zpráva zpracována. Pokud není možno zadat datový rámec, který byl zpracován, tak se zadá zpracovávaná konverzace. V tomto případě je možno určit, z jakého datového rámce pochází právě zpracovávaná signalizační

zpráva. Po přidání všech zdrojů do *ExportSources* se objekt exportuje. To bude proveno tak, že se nad objektem *SnooperExport* zavolá metoda *AddExportObject()*. Vstupním parametrem metody je exportovaný objekt. Pokud se objekt neexportoval, zavolá se metoda *DiscardExportObject()*. Vstupním parametrem je opět exportovaný objekt. Po exportování objektu se pokračuje další zprávou v pořadí ve zpracovávané konverzaci. Jestliže jsou již zpracovány všechny, tak metoda *ProcessConversation()* pro zpracování konverzace ukončí své provádění.

4.5 Zpracování exportovaných objektů WaCallMsg

Poté co se provede zpracování zpráv konverzací a exportování objektů třídy *WaCallMsg*, které reprezentují signalizační zprávy hovoru WhatsApp, bude se pokračovat s jejich dalším zpracováním. Pokud by se nyní tyto objekty nezpracovávaly dále, tak by se v Netfox Detective zobrazovaly pouze obsahy jednotlivých signalizačních zpráv nebo seznam signalizačních zpráv vyskytujících se v určité konverzaci. Sice se může ze signalizačních zpráv zjistit průběh hovoru ručně, ale není to příliš pohodlné. Proto se exportované objekty zpracují ještě jednou a exportují se objekty třídy *WACall*, která bude reprezentovat jednotlivé hovory v zachycené komunikaci. Data z těchto objektů se v Netfox Detective zobrazí v záložce Call v okně pro exportované objekty třídy *SnooperWhatsApp*. V záložce poté uvidíme, kdy hovor započal, kdy byl ukončen, dále jeho délku, telefonní číslo volajícího, telefonní číslo volaného a stav hovoru. Stav značí například, zda se hovor navazuje, právě probíhá nebo již byl ukončen. Tento způsob je rozhodně lepší, než hledat tyto informace ručně ze signalizačních zpráv. Vzniká zde otázka, proč se data zpracovávají na dvakrát a ne najednou. Odpověď na tuto otázku je velmi jednoduchá. Důvodem, proč nelze data zpracovat najednou a exportovat pouze objekty *WACall*, je, že WhatsApp během své komunikace mění servery, kterým zasílá signalizační zprávy. To má za následek, že například signalizační zprávy o ustavování hovoru se vyskytují v jiné konverzaci než signalizační zprávy o ukončení hovoru. A pro zpracování každé konverzace se vytváří nová instance třídy *SnooperWhatsApp*. Instance se nemohu dostat k exportovaným objektům, které exportovaly předchozí instance třídy *SnooperWhatsApp*. Takže jediným řešením, jak se ke všem exportovaným objektům dostat, je, že se exportované objekty *WaCallMsg* zpracují znova. Zpracování objektů *WaCallMsg* se provádí také ve třídě *SnooperWhatsApp*. S rozdílem, že se k vytvoření instance třídy použije konstruktor, kterému se předává seznam s exportovanými objekty z dřívějšího zpracování místo přiřazené konverzace.

Než začnu popisovat zpracování objektů *WaCallMsg* a exportování objektů *WACall*, bylo by dobré si tento objekt napřed popsat, abychom měli představu o tom, jaké informace budou muset být zjistěny a jaké budou pouze převzaty z objektů *WaCallMsg*. Třída *WACall* je jako každý exportovaný objekt potomkem třídy *SnooperExportedObjectBase* odkud dědí například vlastnosti jako *Timestamp* (časové razítko) a *ExportedSources*, která určuje, pomocí jaké zprávy z konverzace se došlo k danému objektu. Dále třída *WACall* dědí od rozhraní *IWACall* a to vlastnosti *CallId* a *State*. Vlastnost *State* je definovaná hodnotami z výčtového typu *WACallState*. Hodnoty výčtového typu *WACallState* určují, v jakém stavu se hovor nachází. Dále ještě třída *WACall* zdědí další vlastnosti rozhraní *IWACall*, protože rozhraní *WACall* dědí vlastnosti od rozhraní *ICall*, u kterých je třeba nadefinovat jejich hodnoty. Zděděním rozhraní *ICall* se zajistí, že se data z exportovaného objektu třídy *WACall* zobrazí v záložce Call. Vlastnosti zděděné z tohoto rozhraní, které je nutné vyplnit, jsou následující:

- *From* - určuje telefonní číslo volajícího
- *To* - určuje telefonní číslo volaného
- *Start* - vlastnost, která sděluje, kdy byl hovor zahájen
- *End* - vlastnost sděluje, kdy byl hovor ukončen
- *Duration* - vyjadřuje délku hovoru

Nyní bylo popsáno, jaké informace nese objekt *WACall* a může se začít s popisem zpracování objektů *WaCallMsg*. Na začátku zpracování se nejprve vytvoří slovník *exportedCalls*. Klíčem ve slovníku bude textový řetězec, který bude reprezentovat *CallId* hovoru. Hodnotami ve slovníku budou objekty *WACall*. Do slovníku se budou ukládat jednotlivé objekty *WACall*, které se následně exportují. Při zpracování se prochází všechny objekty *WaCallMsg*, které byli vyexportovány. U každého objektu *WaCallMsg* se nejprve zkontroluje, zda objekt patří k nějakému již zpracovávanému hovoru.

To se zjistí použitím metody *ContainsKey()* nad slovníkem *exportedCalls*. Vstupním parametrem metody je klíč, u kterého je nutné zjistit, zda se ve slovníku vyskytuje. V tomto případě se za klíč dosadí hodnota *CallId* právě zpracovávané signalizační zprávy. Pokud výstupem metody bude, že zadaný klíč se ve slovníku nevyskytuje, tak to znamená, že signalizační zpráva patří k hovoru, který nebyl ještě zpracováván. Je tedy nutné vytvořit nový objekt *WACall* a přidat jej do slovníku pod příslušným *CallId*. Při vytváření objektu *WACall* dojde také k doplnění hodnoty *CallId* ve vytvářeném objektu. Pokud se ve slovníku klíč vyskytuje, žádný nový objekt *WACall* se nevytváří, protože ve slovníku již objekt *WACall* pro zadané *CallId* existuje.

Dalším krokem ve zpracování je zavolání metody *Update()* nad objektem *WACall*, ke kterému signalizační zpráva patří. Metoda *Update()* má za úkol aktualizovat informace, které objekt *WACall* nese, jako např. stav hovoru, časové údaje započetí a konce hovoru nebo telefonní čísla volajícího a volaného. Vstupem této metody je objekt *WaCallMsg* právě zpracovávané signalizační zprávy. Při provádění této metody se jako první příkaz zavolá metoda *ProcessFromTo()*. Tato metoda určí hodnoty vlastností *From* a *To*. Pokud jsou již tyto vlastnosti vyplněny, tak metoda ukončí své provádění. V případě, že by tyto informace nebyly ještě známy, tak se podle typu signalizační zprávy a hodnoty *From* a *To* objektu *WaCall*, určí volající a volaný. Pro zprávy typu *call - offer* a *receipt - accept* je význam vlastností *From* a *To* objektu *WaCallMsg* totožný s významem těchto vlastností u objektu *WACall*. To ale neplatí v případě, že se jedná o zprávy typu *receipt - offer*, *call - preaccept*, *call - accept*. Tady je nutno vlastnosti *From* a *To* prohodit. To znamená, že objektu *WACall* uložíme do vlastnosti *From* hodnotu *To* objektu *WaCallMsg* a do vlastnosti *To* uložíme hodnotu *From* objektu *WaCallMsg*. Po doplnění informací o volajícím a volaném metoda *Update()* zajistí přidání zdrojů do *ExportSources*. Ze zpracovávaného objektu *WaCallMsg* se přidávají pouze zdroje, které objekt *WACall* ještě neobsahuje. Dále se nastavuje hodnota *Timestamp*. Hodnota vlastnosti bude mít po zpracování všech signalizačních zpráv hovoru nejmenší možnou hodnotu z vlastnosti *Time* ze všech objektů *WaCallMsg*, které byly přiřazeny k tomuto danému hovoru. Což zjednodušeně znamená, že to bude čas, kdy byla zachycena první signalizační zpráva týkající se daného hovoru. Pomocí vlastnosti *Time* se také nastavuje vlastnost *End*.

Poslední věcí, kterou musí metoda *Update()* provést, je určení stavu hovoru. Pro tento účel jsou podstatné tyto signalizační zprávy:

- *call - offer*, tento typ signalizační zprávy určuje, že začalo navazování hovoru. Při zpracování této zprávy se nastaví stav hovoru na *WACallState.ESTABLISHING*.
- *receipt - accept*, z této signalizační zprávy se stanoví, kdy započal hovor. Stav hovoru se nastaví na hodnotu *WACallState.IN_PROGRESS*.
- *call - terminate*, z této signalizační zprávy se určí, kdy hovor byl ukončen a stav hovoru přechází na stav *WACallState.COMPLETE*.

Po provedení metody *Update()* se pokračuje zpracováním dalšího exportovaného objektu *WaCallMsg*. Po zpracování všech exportovaných signalizačních zpráv může započnout exportování objektů *WaCall* uložených ve slovníku exportedCalls. Exportování probíhá tak, že se prochází slovník exportedCalls prvek po prvku a každý objekt *WACall* se exportuje zavoláním metody *AddExportObject()*.

4.6 GUI pro vkládaní hesla

Jak je uvedeno výše, pro dešifrování je potřeba znát heslo, aby se mohli vygenerovat klíče pro dešifrování. Proto se tato podkapitola věnuje problematice jak předat třídě *SnooperWhatsApp* heslo pro generování klíčů. Za účelem zadávání hesel pro třídu *SnooperWhatsApp* byla přidána záložka WhatsApp Passwords do okna Global applications settings. K tomuto oknu se dá dostat kliknutím v menu na VIEW → Settings. Záložka WhatsApp Passwords obsahuje seznam, ve kterém jsou dva sloupce. Jeden sloupec zobrazuje telefonní čísla a v druhém sloupci je heslo, které patří k telefonnímu číslu. Dále záložka obsahuje ještě dvě tlačítka. Tlačítko Add, které při kliknutí zobrazí dialogové okno pro přidání hesla. Druhým tlačítkem je Remove, které slouží ke smazání hesla, které bude označeno v seznamu. Při dvojitém kliknutí na položku seznamu se otevře dialogové okno za účelem úpravy údajů vybrané položky.

Dialogové okno pro přidání hesla se skládá z totožných prvků jako okno pro editaci. Rozdíl je v tom, že při úpravě jsou textová pole pro telefonní číslo a pro heslo předvyplněná údaji z upravované položky. Dialogové okno se skládá ze tří tlačítek a tří textových polí. Jedno textové pole je určeno pro zadání telefonního čísla, ke kterému heslo patří. Zadání hesla je umožněno dvěma způsoby. Buď se vyplní textové pole určené pro heslo v textové podobě nebo se zadá cesta k souboru, ze kterého bude heslo extrahováno. Zde je nutné uvědomit si, že může být vyplněno buď pole určené pro zadání cesty k souboru s heslem, nebo textové pole pro heslo v textové podobě. Obě textová pole nelze vyplnit současně, protože by nebylo možno stanovit pomocí jaké volby heslo určit. Dialogové okno má tlačítko OK, kterým se potvrzuje přidání hesla nebo editace údajů. Toto tlačítko nemůže být stisknuto, pokud nejsou splněny všechny podmínky, aby mohlo být provedeno přidání nebo editace hesla. Dalším tlačítkem je Cancel. Při jeho stisku se dialogové okno zavře a neprovede se přidání nebo editace hesla. Posledním tlačítkem, které ještě zbývá popsat je tlačítko Browse, které při stisku otevře dialogové okno pro otevření souboru. Tímto oknem se pohodlně zadá cesta k souboru s heslem.

Nyní bylo zmíněno, jak grafické rozhraní vypadá a jak by se měly jednotlivé komponenty grafického rozhraní chovat. Jak už bylo v podkapitole 4.1 řečeno Netfox Detective je navržen pomocí návrhového vzoru MVVM. Tím pádem pro realizaci grafického rozhraní se budou muset vytvořit třídy, které budou obsluhovat vrstvy View a Viewmodel, jak pro záložku WhatsApp, tak pro dialogové okno. Tyto třídy jsou součástí vytvořené

knihovny pro rekonstrukci komunikace aplikace WhatsApp. Třída *PasswordSettingsTab* reprezentuje vrstvu View pro záložku WhatsApp Paswords. Tato třída je potomkem třídy *SettingsTabBase* a rozhraní *IPasswordSettingsTab*. Pro realizace seznamu s hesly je použita komponenta DataGrid. Vrstvu Viewmodel pro záložku zajišťuje třída *PasswordSettingsTabVm*. Třída *PasswordSettingsTabVm* pro reprezentaci dat používá seznam třídy *BindingList*. Jedná se o seznam, kde se ukládají položky s hesly a telefonními čísly. Pro reprezentaci položky seznamu slouží třída *PasswordItem*. Dále třída *PasswordSettingsTabVm* popisuje, co se má stát při kliknutí na tlačítko Add pomocí metody *Add()*. Metoda vytvoří objekt třídy *PasswordDialog*. Ta reprezentuje dialogové okno pro přidání či editaci údajů. Třída *PasswordDialog* bude popsána níže. Po vytvoření objektu se nastaví vlastnost *Header*, která určuje titulek okna. V tomto případě se nastaví hodnota "Add password". Následně se zavolá metoda *ShowDialog()*, která zobrazí dialogové okno. Po zavření se zkontroluje, zda dialogové okno bylo zavřeno pomocí tlačítka OK. Pokud bylo okno zavřeno pomocí tlačítka OK, tak se položka do seznamu *Password* přidá a obnoví se aktuální nastavení aplikace ve třídě *ApplicationSettings*. Pomocí této třídy se předávají hesla pro generování klíčů. Metoda *Edit()*, která má na starost úpravu údajů, je prakticky stejná jako metoda *Add()*. Rozdílem je, že vlastnosti *Header* se předává hodnota "Edit password"a vlastnostem *Phone* a *Password* se předávají hodnoty z označené položky v seznamu *Passwords*. Po zavření dialogového okna se nevytváří nová položka, ale upraví se hodnoty označené položky v seznamu. Metoda *Remove()* má za úkol při kliknutí na tlačítko Remove odstranit označenou položku v seznamu. Metoda *ToLog()* ukládá do *ApplicationsSettings* aktuální nastavení hesel ze seznamu *Passwords*, která byla převedena na textový řetězec. Textový řetězec následně nastaví hodnotu *WhatsAppPasswords* pomocí metody *SetValue()* nad třídou *ApplicationsSettings*.

O obsluhu vrstev View a Viewmodel pro dialogové okno se stará třída *PasswordDialog*. Třída obsahuje následující vlastnosti:

- *_phone*, určená k reprezentaci vyplněných dat v textovém poli pro telefonní číslo
- *_password*, uchovává vyplněná data z textového pole pro heslo v textové podobě
- *_path*, určené pro data z textového pole pro cestu k souboru s heslem
- *_status*, pomocí této vlastnosti se zjišťuje, jak bylo dialogové okno zavřeno. Jestli bylo zavřeno pomocí tlačítka OK, či jiným způsobem. Hodnoty vlastnosti se určí pomocí výčtového typu *DialogStatus*. Výčtový typ obsahuje hodnoty OK pro zavřené okno pomocí tlačítka OK a CANCEL pro zavření dialogového okna pomocí tlačítka Cancel nebo křížku.

Jelikož třída *PasswordDialog* zajišťuje obsluhu vrstvy Viewmodel, tak pro výše uvedené vlastnosti je třeba definovat getry a setry. Dále musíme ještě definovat obsluhu při kliknutí na tlačítka OK, Browse a Cancel. Při kliknutí na tlačítko Cancel se dialogové okno zavře pouze pomocí metody *Close()*. Při kliknutí na tlačítko OK se zavolá metoda *ClickOk()*. V metodě se nastaví vlastnost *_status* na hodnotu OK a poté se dialogové okno zavře zavoláním metody *Close()*. Na tlačítko OK lze kliknout, pouze když jsou splněny všechny potřebné podmínky. Jestli jsou všechny potřebné podmínky splněny, zjistíme metodou *ClickOkCanExecute()*. Metoda kontroluje, zda jsou potřebná textová pole vyplněná. Dále zda nejsou současně vyplněna textová pole pro heslo v textové podobě a pro cestu k souboru s heslem. Pokud je zadána cesta k souboru, tak je nutné zkontrolovat, zda uvedený soubor existuje. Kliknutím na tlačítko Browse se zavolá metoda *SetPath()*. Metoda

zobrazí dialogové okno pro otevření souboru pomocí metody *OpenFileDialog()*. Po zavření dialogového okna se nastaví hodnota vlastnosti *_path* cestou k souboru získanou z dialogového okna. Třída *PasswordDialog* ještě definuje metodu *NumberValidationTextbox()*, jejímž účelem je zajistit, aby textové pole pro zadání telefonního čísla obsahovalo pouze čísla a žádné jiné znaky.

Kapitola 5

Testování

Tato kapitola bude věnována testování částí implementace a sběru testovacích dat. Mezi testované části patří například testování třídy *ExtractPass*, která slouží pro extrahování hesla ze souboru. Další testovanou částí bude třída *SnooperWhatsApp*. Jednotlivé části jsou testovány pomocí tzv. unit testů. Unit testy pro knihovnu *SnooperWhatsApp* zajišťuje třída *SnooperWhatsAppTests*. Pro každou testovanou část v této třídě existuje metoda, která předá vstupní data a poté porovná výstupní data s daty očekávanými. Hned v úvodu této kapitoly bych chtěl upozornit na skutečnost, že bohužel pro testování třídy *SnooperWhatsApp* byla použita pouze jedna sada testovacích dat. Důvodem je, že WhatsApp již nepodporuje komunikaci pomocí starší verze protokolu než je verze 2.0. Při instalaci a pokusu o komunikaci pomocí starší verze, aplikace uživatele upozorní na to, že by si měl stáhnout novou verzi aplikace WhatsApp a neumožní mu vykonat žádnou komunikaci.

5.1 Testování extrahování hesla

V této podkapitole bude zmíněn postup testování třídy *ExtractPass*. Pro testování této části bylo použito mobilního zařízení s následující konfigurací:

- Samsung S II (GT - I9100)
- Android 4.1.2 s právy správce
- Password Extractor[1]

Pro účely testování je potřeba, aby mobilní zařízení mělo práva správce, a to z důvodu, aby bylo možné zjistit pomocí aplikace Password Extractor heslo. Získané heslo z aplikace se považuje za očekávaný výstup implementované třídy *ExtractPass*. Pro získání vstupních dat k testování se musí z mobilního zařízení stáhnout soubor *pw*. Postup stažení souboru z mobilního zařízení je uveden v podkapitole 4.2. Metoda *WhatsApptest_Password()* ve třídě *SnooperWhatsAppTests* je určena pro testování nyní popisované testované části. V metodě se nejprve založí nová instance testované třídy *ExtractPass*, které se předává telefonní číslo a cesta k souboru, ze kterého se heslo extrahuje. Poté se z vytvořeného objektu získá extrahované heslo v textové podobě pomocí metody *GetPass()*. Získané heslo se porovná s heslem, které bylo získáno z aplikace Password Extractor. Pokud se hesla shodují, znamená to, že test proběhl úspěšně. Tento postup jsem provedl celkem pětkrát nad odlišnými vstupními daty. Abych donutil aplikaci WhatsApp vytvořit soubor pw s novým heslem, tak

[1] Dostupné z: https://www.mgp25.com/downloads/pw.apk

je nutné ukončit aplikaci WhatsApp a ve Správci aplikací u WhatsAppu se musí vymazat data aplikace. Tím se smaže soubor pw. Při následném spuštění WhatsAppu, uživatel zadá telefonní číslo, na které mu bude zaslána SMS s ověřovacím kódem. Ověřování se provádí za účelem ověření, zda uživatel telefonní číslo opravdu vlastní. Po ověření WhatsApp vytvoří nový soubor s novým heslem.

Tabulka 5.1 obsahuje očekávané vstupy a dosažené výstupy pro jednotlivé testy třídy *ExtactPass*. Z této tabulky vyplývá, že u všech testů byl očekávaný výstup totožný s dosaženým výstupem, takže všechny testy proběhly úspěšně. Z toho lze odvodit, že úspěšnost při testování třídy *ExtractPass* byla 100 procent.

	Očekavaný výstup	**Dosažený výstup**
1	lqOPkcJxNGVqBQ7FQS8a1xFkaqE=	lqOPkcJxNGVqBQ7FQS8a1xFkaqE=
2	eVGqrhyzOemNDh3MPgmfITQBoME=	eVGqrhyzOemNDh3MPgmfITQBoME=
3	0q8lki2l8B4pjjRKZQpE4Ab/Jkg=	0q8lki2l8B4pjjRKZQpE4Ab/Jkg=
4	WwaZUJmUqej8jFSdVgmvj0f7h3Y=	WwaZUJmUqej8jFSdVgmvj0f7h3Y=
5	o6tpqRsFw4x7uJ/SvYV3aoiVHEU=	o6tpqRsFw4x7uJ/SvYV3aoiVHEU=

Tabulka 5.1: Tabulka s výstupy při tesování třídy *ExtractPass*

5.2 Testování rekonstrukce komunikace

Tato podkapitola je věnována popisu postupu testování funkčnosti třídy *SnooperWhatsApp*, které proběhlo pomocí metody *SnooperWhatsApptest_wh_15*. Na začátku metody se ve třídě *ApplicationSettings* nastaví hodnota *WhatsAppPasswords* na "420724874788,627XlMqch8i5Ncy2tRSbZLXs2m0=;". Tím předáme třídě *SnooperWhatsApp* heslo pro vygenerování klíčů pro dešifrování.

Následně se zavolá metoda *ProcessCapture()*. Tato metoda zpracuje vstupní soubor se zachycenou komunikací na jednotlivé konverzace. Jako vstupní soubor pro testování této části implementace byl použit soubor, který byl vytvořen v rámci práce *WhatsApp network forensics: Decrypting and understanding the WhatsApp call signaling messages* [11]. Tento soubor obsahuje komunikaci aplikace WhatsApp, ve které se uskutečnil hovor a aplikace WhatsApp komunikovala verzí protokolu 1.5. Důvod, proč nebyla vytvořena testovací data pro tuto testovanou část v této práci, byl již zmíněn na začátku kapitoly. Po ukončení metody *ProcessCapture()* se vytvořené konverzace uloží do proměnné *conversation*. Dále se zavolá medoda *ExportData()*, které pomocí proměnné *conversation* předáme konverzace ke zpracování. Zavoláním této metody dojde k prvnímu zpracování, kdy se ze zpráv konverzace exportují objekty *WaCallMsg*, které reprezentují jednotlivé signalizační zprávy týkající se hovorů. Po dokončení exportování všech objektů *WaCallMsg* se tyto objekty předají metodě *ExportData()*, která se zavolá ihned po ukončení první zavolané metody *ExportData()*. Druhé zavolání této metody zapříčiní, že se z exportovaných objektů WaCallMsg zjistí, zda se v zachycené komunikaci uskutečnil nějaký hovor či hovory. Pokud se v komunikaci uskutečnil nějaký hovor, tak se pro každý hovor exportuje objekt *WACall*. Po skončení druhé metody *ExportData()* se provede porovnání počtu exportovaných objektů s očekávaným počtem objektů a počet exportů s očekávaným počtem exportů. Pokud se při obou porovnáních dosažené počty rovnají s počty očekávanými, tak to znamená, že test z části proběhl úspěšně. Ještě je totiž nutné zkontrolovat, zda získané informace o hovoru, který

se vyskytuje v testované komunikaci, se shodují se skutečností. Očekávané počty a informace o hovoru jsou zjištěny analýzou zachycené komunikace pomocí programu Wireshark. Při testování se počty, ke kterým se dospělo zpracováním, nelišily od počtů, které byly očekávány. Dále je z tabulky 5.2 vidět, že informace o hovoru se také shodují. Takže tento test proběhl úspěšně, to však neznamená, že funkčnost třídy SnooperWhatsApp je 100 procentní, protože nebyly testovány všechny podporované verze protokolu a ani všechny stavy, které mohou nastat při hovoru, a to z důvodu, který byl uveden v úvodu této kapitoly.

	Očekavaný výstup	**Dosažený výstup**
Volající	420724874788	420724874788
Volaný	12035006106	12035006106
Začátek hovoru	5/15/2015 9:26:49 PM	5/15/2015 9:26:49 PM
Konec hovoru	5/15/2015 9:28:12 PM	5/15/2015 9:28:12 PM
Doba trvání hovoru	00:01:24.1376460	00:01:24.1376460
Stav	COMPLETE	COMPLETE

Tabulka 5.2: Tabulka s výstupy při tesování třídy *SnooperWhatsApp*

Kapitola 6

Závěr

Bakalářská práce se zabývá rekonstrukcí zachyceného provozu komunikačních nástrojů. Zaměřil jsem se na dvě velmi používané komunikační aplikace, a to Viber a WhatsApp.

V první části jsem u Vibru popsal, jak vypadá komunikace při odesílání a přijímání multimediálních dat a aktuální polohy v nezašifrované podobě. Zjistil jsem, že od verze 5 nejsou již multimediální data a poloha odesílány v nezašifrované podobě. Zachycená multimediální data jsou určitou dobu veřejně přístupná na internetu.

V další části bylo popsáno, jak probíhá autentifikace u WhatsAppu, jak vypadá signalizace volání, a vysvětlil jsem jakou variantu šifry RC4 používá WhatsApp pro šifrování komunikace. WhatsApp pro kódování zvuku používá kodek Opus se vzorkovací frekvencí 16 kHz nebo 8 kHz. Pokud nejsou k dispozici zachycená data z autentifikace pomocí full handshake, tak není možné dešifrovat komunikaci, protože pokud se zachytí autentifikace, která je prováděna pouze metodou half handshake, jenž neobsahuje zprávu challenge, ze které se zjistí 20 bajtový řetězec, pomocí kterého se generuje klíč pro dešifrování komunikace. U použité šifry RC4 jsem zjistil, že oproti klasické variantě se liší pouze v inicializační části, kde dešifruje data o délce 768 bajtů nebo 256 bajtů, čímž docílí dalšího promíchání keystreamu.

V rámci práce byla implementována knihovna pro rekonstrukci zachycené komunikace aplikace WhatsApp. Knihovna se ze zachycené komunikace snaží dostat informace důležité pro forenzní analýzu, konkrétně informace o uskutečněných hovorech. Aby bylo možné dešifrovat komunikaci, je třeba znát heslo, které WhatsApp používá ke generování šifrovacích klíčů. Toto heslo je uloženo na mobilním zařízení, které je odposlouchováno. Existují dvě metody jak získat heslo. Jedna z metod je použít aplikaci Password Extractor. Při použití této aplikace jsou potřeba práva správce na daném mobilním zařízení. To může být ve většině případů problém, protože tyto práva není možné vždy získat. Druhou metodou je si vytvořit obraz paměti mobilního zařízení a z obrazu si získat soubor s heslem. Z tohoto souboru je poté nutné heslo extrahovat. V implementaci této práce jsou umožněny obě varianty, jak varianta s ručním zadáváním hesla, tak varianta, kdy se heslo extrahuje ze souboru.

Při testování byla ověřena funkčnost třídy pro extrahování hesla ze souboru a třídy zajišťující rekonstrukci zachycené komunikace. Testování rekonstrukce komunikace proběhla na verzi protokolu 1.5, kde proběhlo testování úspěšně. Bohužel tato a starší verze nebyly více testovány z důvodu, že se již nedají opatřit testovací data pro testování, protože aplikace WhatsApp již komunikuje s novějšími verzemi protokolu. Při dokončování této práce WhatsApp používal verzi protokolu 2.0. Tato verze již nepoužívá šifrovací algoritmus RC4. To by mohl být důvod, proč se do budoucna tímto protokolem dále zabývat.

Nyní bych jěště zmínil výhody, kterých dosáhlo implementované rozšíření pro Netfox

Detective oproti Wiresharku s rozšířením pro rozpoznání protokolu WhatsApp. První výhoda oproti Wiresharku je, že v implementovaném rozšíření pro Netfox Detective je možné zadat heslo pro generování klíčů pro dešišifrování, jak ručně, tak zadáním cesty k souboru, ze kterého se heslo extrahuje. Ve Wiresharku je umožněno pouze zadat heslo ručně. Další podstatná výhoda oproti Wiresharku je zobrazení výpisu uskutečněných hovorů. Ve výpisu se vyskytují informace jako telefonní číslo volajícího či volaného, dále kdy hovor započal a kdy skončil, jak dlouho hovor trval a stav hovoru. Ve Wiresharku je nutné tyto informace o všech hovorech hledat ručně ze zpráv protokolu WhatsApp.

Literatura

[1] Šifrování. *FI WIKI* [online]. 2007-06-25 [cit. 2016-05-03].
URL https://kore.fi.muni.cz/wiki/index.php/%C5%A0ifrov%C3%A1n%C3%AD

[2] AES - Advanced Encryption Standard. *Guard Dynamics* [online]. 2016 [cit. 2016-05-03].
URL http://www.guard-dynamics.cz/aes-advanced-encryption-standard

[3] Extracting password from device. *GitHub* [online]. [cit. 2015-03-27].
URL
https://github.com/mgp25/Chat-API/wiki/Extracting-password-from-device

[4] Opening ports for Viber Desktop. *Viber* [online]. [cit. 2015-12-20].
URL https://support.viber.com/customer/portal/articles/1506350-opening-ports-for-viber-desktop#.VqePRVnIuAM

[5] Implementing the Model-View-ViewModel Pattern. *Microsoft* [online]. [cit. 2016-05-03].
URL https://msdn.microsoft.com/en-us/library/ff798384.aspx

[6] Berners-Lee, T.; Fielding, R. T.; Nielsen, H. F.: Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, RFC Editor, May 1996.
URL http://www.rfc-editor.org/rfc/rfc1945.txt

[7] Dajbych, V.: MVVM: Model-View-ViewModel. *dotnetportal.cz* [online]. 2009-04-21 [cit. 2016-05-03].
URL http://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel

[8] Fielding, R. T.; Gettys, J.; Mogul, J. C.; aj.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, June 1999.
URL http://www.rfc-editor.org/rfc/rfc2616.txt

[9] Freier, A.; Karlton, P.; Kocher, P.: The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101, RFC Editor, August 2011.
URL http://www.rfc-editor.org/rfc/rfc6101.txt

[10] Indra, M.: *Šifrovací algoritmy*. Pardubice, 2014. Bakalářská práce. Univerzita Pardubice, Fakulta elektrotechniky a informatiky.

[11] Karpíšek, F.; Baggili, I.; Breitinger, F.: WhatsApp network forensics: Decrypting and understanding the WhatsApp call signaling messages. *Digital Investigation*, ročník 2015, č. 15, 2015: s. 1–9, ISSN 1742-2876.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=10979

[12] Lauš, J.: *Moderní kryptoanalytické metody.* Brno, 2007. Diplomová práce. Masarykova Univerzita, Fakulta informatiky.

[13] Menezes, A. J.; Oorschot, P. C. v.; Vanstone, S. A.: *Handbook of applied cryptography.* Boca Raton, London, New York: CRC Press, 1996, ISBN 0–8493–8523–7.
URL http://cacr.uwaterloo.ca/hac/

[14] Michiel; Bosma, J.; Veerman, G.: Viber Communication Security unscramble the scrambled. 2011.
URL https://www.dropbox.com/s/0jbp62nu0f46nf1/Viber%20Communication%20Security%20unscrambIe%20the%20scrambIed.pdf?dl=0

[15] Paganini, P.: Viber vulnerable to MITM attack, million users at risk. *Security Affairs* [online]. 2014-04-24 [cit. 2016-01-10].
URL http://securityaffairs.co/wordpress/24305/hacking/viber-vulnerable-mitm.html

[16] Pluskal, J.: NetFox.Framework - The network forensic extandable analysis tool. In *Proceedings of the 20th Conference STUDENT EEICT 2014 Volume 2*, Brno University of Technology, 2014, ISBN 978-80-214-4923-7, s. 280–282.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=10692

[17] Pluskal, J.; Matoušek, P.; Ryšavý, O.; aj.: Netfox Detective: A tool for advanced network forensics analysis. In *Proceedings of Security and Protection of Information (SPI) 2015*, Brno University of Technology, 2015, ISBN 978-80-7231-997-8, s. 147–163.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=10863

[18] Polívka, J.: Pod pokličkou šifrovacího algoritmu RC4. *itnetwork.cz* [online]. [cit. 2016-01-20].
URL http://www.itnetwork.cz/algoritmy/ostatni/pod-poklickou-algoritmu-rc4/

[19] Smith, C.: 50 Amazing WhatsApp Statistics. *expandedramblings.com* [online]. 2016-01-22 [cit. 2016-01-28].
URL http://expandedramblings.com/index.php/whatsapp-statistics/

[20] Vlček, L.: Jak vytvořit Wifi hotspot pod Windows. *lukas-vlcek.cz* [online]. 2016-04-08 [cit. 2016-04-10].
URL http://www.lukas-vlcek.cz/2016/04/08/jak-vytvorit-wifi-hotspot-pod-windows/

[21] Walnycky, D.; Baggili, I.; Marrington, A.; aj.: Network and device forensic analysis of Android social-messaging applications. *Digital Investigation*, ročník 14, Supplement 1, 2015: s. S77 – S84, ISSN 1742-2876, doi:http://dx.doi.org/10.1016/j.diin.2015.05.009, the Proceedings of the Fifteenth Annual DFRWS Conference.
URL http://www.sciencedirect.com/science/article/pii/S1742287615000547