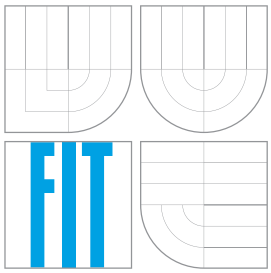
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

REKONSTRUKCE WEBMAILOVÉHO PROVOZU

WEBMAIL TRAFFIC RECONSTRUCTION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MIROSLAV SLIVKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. VLADIMÍR VESELÝ
SUPERVISOR

BRNO 2015

Abstrakt

Webmailové aplikace jsou dnes mezi lidmi velmi populární díky velké dostupnosti Internetu. Kromě běžného použití mohou být díky šifrovanému spojení využity ale i na nekalé účely spojené s únikem citlivých dat. Tato práce se zabývá analýzou webmailového provozu, typických znaků webmailových událostí a zpřístupněním dešifrovaného obsahu komunikace chráněné protokolem SSL/TLS. V práci budou navrženy a implementovány moduly pro Netfox.Framework, které zabezpečují zmiňované dešifrování SSL/TLS a analýzu webmailu. Závěrem práce jsou tyto části implementovány jako součást Netfox.Framework-u v rámci projektu bezpečnostního výzkumu SEC6NET na FIT VUT v Brně.

Abstract

Webmail applications are very popular these days. Besides typical usage, thanks to ciphered communication, they can be used for malicious activity like confidential data loss. This thesis discusses webmail events detection based on common webmail signatures in captured network traffic. Also there will be discussed SSL/TLS interception and decryption for further data analysis. The modules in this thesis are designed and implemented for Netfox.Framework forensics analysis tool. The Netfox project is developed at FIT BUT under security research project SEC6NET.

Klíčová slova

Netfox.Framework, e-mail, webmail, detekce webmailů, dešifrování SSL/TLS

Keywords

Netfox.Framework, e-mail, webmail, webmail detection, deciphering SSL/TLS

Citace

Miroslav Slivka: Rekonstrukce webmailového provozu, diplomová práce, Brno, FIT VUT v Brně, 2015

Rekonstrukce webmailového provozu

Prohlášení

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením Ing. Vladimíra Veselého s použitím literatúry, ktorú uvádzam v zozname.

.....
Miroslav Slivka
24. května 2015

Poděkování

Týmto by som rád poďakoval svojmu vedúcemu Ing. Vladimírovi Veselému a konzultantovi Ing. Janovi Pluskalovi za odbornú pomoc.

Pudingový koláč:
cesto: 5 vajec,
20 dkg práškový cukor,
25 dkg polohrubá múka,
1 dcl voda,
1 dcl olej,
1 prášok do pečiva,
1 vanilkový cukor.
Do cesta sa vlieva uvarený puding:
1 L mlieka,
4 lyžice cukru,
2 čokoládové pudingy.
Na upečený a vychladnutý koláč rozotrieme pochúťkovú smotanu:
3 pochúťkové smotany,
3 lyžice práškový cukor,
1 vanilkový cukor.

Obsah

Úvod

Motivácia pre túto prácu je adaptovať nástroj *Netfox.Framework* novým potrebám, ktoré vznikajú pri jeho implementácii. *Netfox.Framework* je nástroj určený k forenznej analýze zachytenej komunikácie v počítačových sieťach. *Netfox.Framework* poskytuje prístup k pokročilému získavaniu aplikačných dát v zachytenej komunikácii založený na konverzáciách. Tento nástroj bol vyvinutý ako diplomová práca [13] na Fakulte Informačních Technologií, Vysokého Učení Technického v Brně pod záštitou výskumnej skupiny SEC6NET a jeho vývoj ďalej pokračuje ako súčasť rodiny nástrojov *Netfox*. Nástroj *Netfox.Framework* sa snaží poskytnúť jednoduchý systém vývoja špecifických aplikácií určených k forenznej analýze. Cieľom tejto práce je vyvinúť moduly implementujúce dešifrovanie SSL/TLS komunikácie a detekciu webmailovej komunikácie pre tento framework.

Webmail je emailový klient implementovaný ako webová aplikácia bežiaca na webovom serveri. Ako u väčšiny webových aplikácií, najväčším plusom webmailov pred použitím desktopových klientov je možnosť posielať a prijímať emaily odkiaľkoľvek iba s použtím webového prehliadača. Hlavnou nevýhodou webmailových aplikácií je nutnosť pripojenia k Internetu. Všetky webové aplikácie sú odkázané na používanie Hypertext Transfer Protokolu (HTTP). HTTP pracuje na princípe žiadosť/odpoveď, kde klient posiela žiadosti na server a server ju spracuje a pošle odpoveď. V predvolenom nastavení HTTP pracuje na porte 80. HTTP žiadosti a odpovede sú označované ako správy. Protokol používa metódy na vykonanie určitých operácií. Z pohľadu tejto práce budú najzaujímavejšie metódy `GET` a `POST`. `GET` slúži na získanie dát od serveru a `POST` na odoslanie dát na server.

Stále väčšie množstvo webovej prevádzky je dnes prenášané šifrovane pomocou SSL/TLS. To môže spôsobiť problémy pri jej analýze. K zavedeniu šifrovania viedlo niekoľko faktorov:

- priemyselné a vládne regulačné opatrenia vyžadujúce, aby citlivé údaje boli prenášané šifrovane,
- ľudia, ktorý potrebujú pristupovať na web anonymne, napríklad obyvatelia Číny alebo inej krajiny s cenzúrovaným internetom,
- útočníci snažiaci sa zakryť svoje aktivity a komunikačné kanály.

Ako je vidieť v tomto výpise okrem bezpečnosti a anonymity, ktoré poskytuje šifrovanie komunikácie, tak šifrovaná komunikácia môže byť rovnako aj skrytou hrozbou.

Transport Layer Security (TLS) protokol je IETF štandard navrhnutý na poskytnutie bezpečnej komunikácie cez Internet. Predchodcom TLS je Secure Socket Layer (SSL) protokol. TLS bol navrhnutý ako vylepšenie SSL a stal sa široko používaným protoklom poskytujúcim bezpečnú komunikáciu pre webové aplikácie. V predvolenom nastavení HTTP over TLS/SSL čiže HTTPS počúva na porte 443.

V Kapitole 1 je popísaná analýza webmailov a protokolu TLS/SSL. Následne Kapitola 2 a Kapitola 3 sa zaoberá návrhom a implementáciou vyššie spomínaných modulov. V

Kapitole 4 je ukázaný proces testovania. Záver zhodnocuje doposiaľ dosiahnuté výsledky práce a diskutuje možné rozšírenia.

Dostupné riešenia

V nasledujúcej časti sú preskúmané dostupné riešenia poskytujúce potrebnú funkcionalitu pre nástroj *Netfox.Framework*. V tomto prípade ide o dešifrovanie SSL/TLS komunikácie. Mnoho monitorovacích nástrojov obsahuje možnosť dešifrovania SSL/TLS komunikácie, mimo iné aj Wireshark. Vo Wireshark-u je ale problém s poskytnutím takto dešifrovanej komunikácie nástroju *Netfox.Framework*, pretože uloženie dešifrovanej komunikácie podporované nie je.

Ďalej v tejto kapitole budú rozobraté dostupné riešenia detekcie webmailových udalostí v zachytenej komunikácii. Takéto nástroje patria často bohužiaľ do komerčnej sféry a ich *know-how* je pre túto prácu skryté.

SSL/TLS dešifrovanie

Riešenie od spoločnosti Gigamon[1] na dešifrovanie SSL/TLS komunikácie je hardwarový nástroj nazvaný *GigaSMART*. Toto riešenie poskytuje okrem dešifrovania SSL/TLS relácií aj iné funkcie na podporu monitorovania sieťovej prevádzky.

Hlavnou výhodou v riešení, ktoré poskytuje tento nástroj je, že nespôsobuje spomalenie výkonu samotnej monitorovacej aplikácie. Na to, aby dešifrovanie pomocou tohto nástroja fungovalo je nutné mať prístup k privátnym kľúčom. Tieto je nutné nahrať do tohoto nástroja.

Hlavnou nevýhodou tohto riešenia je cena. Okrem ceny mínusom je aj obmedzená podpora šifrovacích algoritmov a verzií SSL/TLS protokolu. V obdobý vytvorenia tejto práce na stránkach Gigamonu bola vypísaná podpora pre protokoly SSL3, TLS v1.0 a TLS v1.1. Podpora asymetrických algoritmov zahŕňa iba algoritmus RSA. Podporované symetrické algoritmy sú AES, DES a 3DES.

Iným riešením by sa mohlo zdať nasadenie SSL proxy. Tieto nástroje ale väčšinou postrádajú možnosť replikovania prevádzky smerom na monitorovací nástroj.

Detekcia webmailovej komunikácie

Jedným z nástrojov, ktoré dokážu detekovať webmailovú komunikáciu je napríklad nástroj *NetResident*[2] od spoločnosti *TamoSoft*. *NetResident* je nástroj na analýzu obsahu sieťovej komunikácie v reálnom čase, ale aj offline. Okrem detekcie a rekonštrukcie webmailovej komunikácie podporuje taktiež detekciu a rekonštrukciu VoIP, IM a ďalších aplikačných protokolov.

Nevýhodou tohto produktu je opäť cena. Okrem ceny, zo špecifikácie produktu nie je známa podpora šifrovanej komunikácie. Bez toho bude potrebné uvažovať o ďalšom nástroji, ktorý poskytne nástroju *NetResident* už dešifrovanú komunikáciu.

Iné nástroje, ktoré podporujú detekciu webmailovej aktivity sa často zaoberajú zabezpečením proti úniku citlivých dát[3]. Takéto nástroje často bývajú vo forme hardware[4].

[1] https://www.gigamon.com/products/technology/ssl-decryption
[2] http://www.tamos.com/products/netresident/
[3] http://www.codegreennetworks.com/products/network-data-loss-prevention/
[4] http://zecurion.com/zgate/

Taktiež je možné nájsť štúdie zaoberajúce sa detekciou webmailovej prevádzky z netflow dát [16]. Detekcie na základe netflow dát pracujú ale so štatistickou analýzou. To v tejto práci nie je uplatniteľné.

Kapitola 1

Analýza

Táto kapitola sa zaoberá analýzou komunikácie webmailových aplikácií so svojím *backend*-om. Okrem protokolov používaných samotnými aplikáciami sa čitateľ oboznámi s protokolmi zabezpečujúcimi bezpečné spojenie medzi aplikáciami (SSL/TLS) a niektorými najpoužívanejšími šiframi či autentizačnými algoritmami, ktoré sa dnes používajú. Taktiež sa v tejto kapitole diskutujú možné bezpečnostné problémy týchto mechanizmov.

1.1 Analýza webmailov

Webmail je webová aplikácia poskytujúca služby e-mailového klienta. Webmailová aplikácia býva zviazaná priamo s poskytovateľom mailových služieb ako napríklad Seznam.cz, Google, Microsoft, Yahoo a podobne. Existujú však aj webmailové aplikácie, ktoré nie sú zviazané so žiadnym poskytovateľom a je možné si ich nainštalovať na domáci či firemný server a používať s vlastným mailovým serverom. Takýmito aplikáciami sú napríklad Roundcube[1], Horde[2] a mnoho ďalších.

Webové aplikácie môžu byť napísané v rôznych jazykoch, ale všetky riešia nejakým spôsobom komunikáciu medzi klientom a serverom. Základný princíp fungovania webmailovej aplikácie je nasledujúci (viď Obrázok 1.1):

- Po prihlásení sa užívateľovi zobrazí obsah zložky inbox,
  - klient pošle žiadosť na webový server, že chce zobraziť obsah tohto prečinku,
  - webový server dá žiadosť na mail server, kde sa nachádza konto užívateľa, na zobrazenie obsahu daného priečinku,
  - výsledok operácie mailového serveru sa vhodne zakóduje a pošle klientovi.
- Užívateľ posiela e-mail,
  - klient stlačil tlačidlo odoslať správu,
  - webovému serveru sa posiela žiadosť na odoslanie správy obsahujúca všetky atribúty mailovej správy,
  - webový server posiela vhodne zakódovanú správu na odoslanie mailovému serveru,

[1] http://roundcube.net/
[2] http://www.horde.org/

– webový server informuje o úspechu danej operácie.

Použitý formát kódovania správ sa môže líšiť, všetky aplikácie ale odosielajú polia e-mailovej správy.

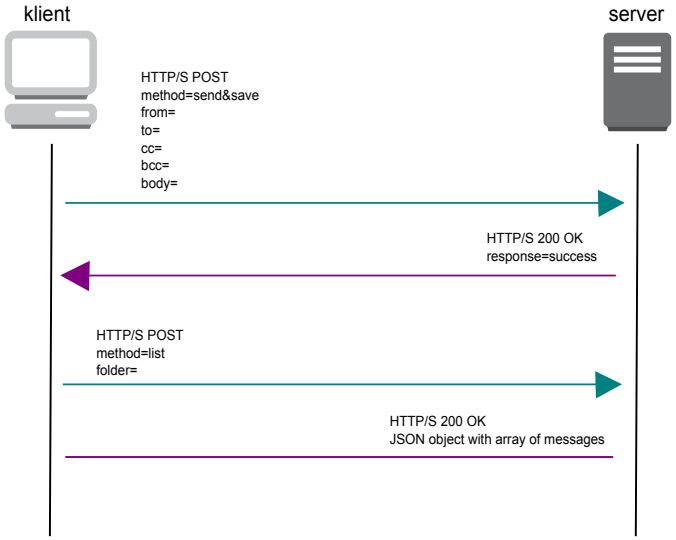
V nasledujúcich tabuľkách je prehľad analýzy komunikácie vybraných webmailových aplikácií. Každá tabuľka sa začína názvom analyzovanej služby. Následuje riadok *URL aplikácie*, ktorý obsahuje identifikátor, na základe ktorého vieme povedať, že sa komunikuje práve s analyzovanou webmailovou aplikáciou. Riadok *Kódovanie správy* popisuje ako je prenášaná správa kódovaná. Ostatné riadky popisujú, čo sa deje pri vykonávaní jednotlivých metód. Údaje ktoré popisujú nasledujúce tabuľky boli získané v rozmedzí 2/2015 až 4/2015.

Gmail	
URL aplikácie	/mail/.*
Kódovanie správy	Formulár zakodovaný v URL ako `key=value` páry
Nová správa	Správa obsahujúca prázdne polia mailu s novým `composeid`
Zobrazenie obsahu priečinku	*Request URI* obsahuje položku *search*; Odpoveďou je štruktúra polí
Zobrazenie správy	

Tabuľka 1.1: Analýza komunikácie webmailu gmail.

Microsoft Live Maill	
URL aplikácie	ol/mail.fpp
Kódovanie správy	Formulár zakódovaný ako `key=value` páry v URL
Nová správa	Pár `mn="SendMessage_ec"` a `d="obsah,oddelený,čiarkami"`; V obsahu nie sú pomenované polia predmetu, tela správy, komunikujúcich strán ani ostatné
Zobrazenie obsahu priečinku	`mn="GetInboxData"`; Odpoveď je javascript objekt, ktorý má v parametroch pole správ na zobrazenie.
Zobrazenie správy	

Tabuľka 1.2: Analýza komunikácie webmailu Microsoft Live Mail.



Obrázok 1.1: Základný princíp komunikácie webmailových aplikácií.

Yahoo Mail	
URL aplikácie	parameter v url `appid=YahooMailNeo`
Kódovanie správy	JSON alebo JSON zapúzdrený v MIME
Nová správa	JSON objekt zakódovaný v MIME multipart. `requests.payloadParts.payload` obsahuje správu a `requests.requests.id` obsahuje typ `SendMessage`
Zobrazenie obsahu priečinku	`requests.payload.method.ListMessagesInThread` `requests.payload.method.params` je pole obsahujúce `fid` (folder id) listovaného priečinku; Odpoveď je každá správa vo vlákne/konverzácii zakódovaná v JSON a každý JSON objekt je jedna časť MIME multipart
Zobrazenie správy	

Tabuľka 1.3: Analýza komunikácie webmailu Yahoo Mail.

Seznam	
URL aplikácie	email.seznam.cz
Kódovanie správy	FastRPC principiálne XML-RPC zakódované v base64
Nová správa	`user.message.send` nasledované položkami správy
Zobrazenie obsahu priečinku	`user.list.messages` - odpoveď obsahuje celé správy
Zobrazenie správy	

Tabuľka 1.4: Analýza komunikácie webmailu seznam.cz.

Atlas.cz/ centrum.cz	
URL aplikácie	gm-pool.centrum.cz
Kódovanie správy	Formulár zakódovaný v URL ako `key=value` páry
Nová správa	*Request URI* obsahuje parametre `m=newmsg` a `op=send`
Zobrazenie obsahu priečinku	metódou `GET`, parameter `m=list`; odpoveď je JSON objekt, `Content-Type` je ale *text/plain*
Zobrazenie správy	

Tabuľka 1.5: Analýza komunikácie webmailu atlas/centrum

Horde	
URL aplikácie	
Kódovanie správy	Formulár zakódovaný v URL ako `key=value` páry alebo formulár v MIME multipart
Nová správa	*Request URI* obsahuje `compose.php`
Zobrazenie obsahu priečinku	*Request URI* obsahuje `folders.php` odpoveď je HTML stránka; neobsahuje celé správy
Zobrazenie správy	

Tabuľka 1.6: Analýza komunikácie webmailu Horde

Roundcube - roundcubemail 0.8.5	
URL aplikácie	
Kódovanie správy	Formulár zakódovaný v URL ako `key=value` páry alebo JSON
Nová správa	obsah `POST` metódy je správa spolu s `_task=mail _action=send`
Zobrazenie obsahu priečinku	`_task=mail _action=list` v parametroch *Request URI*; `Content-Type` je *text/plain* obsah je ale JSON; neobsahuje celé správy
Zobrazenie správy	

Tabuľka 1.7: Analýza komunikácie webmailu Roundcube.

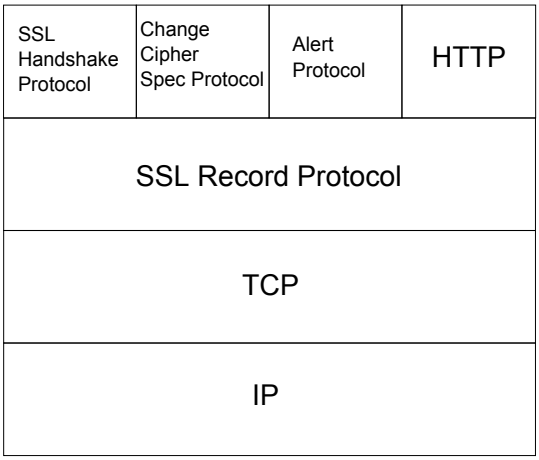
Z analýzy komunikácie bolo ďalej zistené, že v prípade webmailov Seznam, Gmail, Yahoo a Microsoft Live prebieha celá komunikácia šifrovane už od prístupu na stránku. V prípade webmailu Centrum/Atlas šifrovane prebieha iba prihlásenie či registrácia a ďalšia komunikácia prebieha nešifrovane. U webmailových aplikácií Roundcube a Horde šifrovanie závisí na nasadení.

Väčšina vyššie popísaných aplikácií pri odosielaní novej správy používa v obsahu HTTP pomenované polia e-mailovej správy. To bude možné využiť pri návrhu detekcie webmailových udalostí. Ostatné udalosti prebiehali pomerne rôznorodo. Pri udalostiach ako *zobrazenie priečinku* je možné sa spoľahnúť, že sa v nejakom kontexte vyskytne slovíčko *list*.

Tabuľky neobsahujú kompletne všetky akcie, ktoré je možné vykonávať nad webmailovou aplikáciou. Chýbajúce udalosti boli vyhodnotené ako nevhodné na extrakciu, pretože neniesli dostatok informácií. Ide o udalosti typu *presun správy medzi zložkami* či *vymazanie správy*. Tieto udalosti zvyčajne využívajú identifikátor správy a preto nebolo možné z takýchto udalostí zistiť podstatné informácie ako obsah správy, príjemca, predmet a podobne.

1.2 Analýza protokolu SSL/TLS

Transport Layer Security (TLS) protokol je IETF štandard navrhnutý na poskytnutie bezpečnej komunikácie cez Internet (najnovšie v RFC 5246[6]). Predchodcom TLS je Secure Socket Layer (SSL) protokol (viď RFC 6101 [10]). TLS bol navrhnutý ako vylepšenie SSL a stal sa široko používaným protokolom poskytujúcim bezpečnú komunikáciu pre webové aplikácie, elektronickú poštu, internetový fax a ďalšie dátové prenosy.



Obrázok 1.2: Protokolový zásobník SSL. Zdroj: W. Stallings, 1998 [17].

Vo vrstvovej architektúre TLS je obvykle implementovaný nad niektorým transportným protokolom a zapúzdruje aplikačné protokoly ako sú HTTP, FTP, SMTP, NNTP či XMPP. Historicky bol používaný so spoľahlivými transportnými protokolmi ako je napríklad TCP. Avšak existuje aj implementácia, ktorá pracuje s datagramovými transportnými protokolmi, ako je UDP. Táto implementácia bola štandardizovaná pod názvom Datagram Transport Layer Security (DTLS) [15].

SSL/TLS relácia je asociácia medzi klientom a serverom. Jedna SSL/TLS relácia môže spravovať viacero spojení. SSL/TLS je vrstvový protokol, ktorý definuje ďalšie protokoly (viď Obrázok 1.2):

- záznamová vrstva (*Record Layer*),
- *Change Cipher Spec* protokol,
- *Alert* protokol,
- *Handshake* protokol.

Record protokol

Úlohou *Record Layer* je zobrať dáta od aplikácie, rozdeliť ich na menšie bloky, môže ich skompirmovať, ak bol dohodnutý komprimačný algoritmus (viď RFC 3749 [11]), aplikuje Message Authentication Code (MAC)[3], zašifruje dáta a výsledok odošle.

Samotné dáta prenášané v ustanovenej relácii sú chránené buď blokovou, prúdovou alebo tzv. Authenticated Encryption with Associated Data (AEAD) [12] šifrou. AEAD je mód

[3]MAC algoritmus, alebo kľúčovaná hash funkcia, poskytuje prenášanej informácii pravosť a integritu.

blokovej šifry, ktorý poskytuje dôvernosť, integritu a autenticitu dát. V TLS napríklad Galois/Counter Mode (GCM) [8] či Counter with CBC-MAC Mode (CCM) [18]. Pri prúdovej šifre je štruktúra šifrovaných dát jednoduchá, tj. obsah a MAC hodnota. Od TLS v1.1 [5] v režime CBC[7] blokových šifier sa pred šifrovaný obsah predsúva explicitný inicializačný vektor.

Alert protokol

Hlavnou úlohou *Alert* protokolu je oznamovať chyby pri *Handshake*-u. Definované sú rôzne úrovne chýb. Najčastejšie sa asi stretneme s úrovňou *fatal*, kedy je *Handshake* okamžite ukončený a relácia zrušená. Ďalšia významná správa typu *alert* je *Closure alert*, ktorá sa posiela po skončení relácie. Táto správa musí byť zaslaná vždy na konci relácie, vďaka nej sa zamedzuje *truncation* [4] útoku. Všetky správy prijaté po *Closure* sú ignorované.

Change Cipher Spec protokol

Protokol *Change Cipher Spec* obsahuje jedinú správu, ktorá oznamuje zmenu kryptografických parametrov. Táto správa sa posiela vždy pred správou *Finished* v rámci *Handshake* protokolu alebo sa môže poslať hocikedy v rámci aktívnej relácie chránená touto reláciou.

Handshake protokol

Na vyjednanie parametrov šifrovanej relácie sa používa *Handshake* protokol, ktorý môžme rozdeliť na tri časti: inicializačnú, autentizačnú a ukončenie *Handshake*-u (viď Obrázok 1.3). Pomocou *Handshake* protokolu sa vyjednajú nasledujúce parametre:

- ID relácie,
- certifikáty peerov,
- kompresný algoritmus,
- CipherSpec - šifrovací algoritmus, MAC a dĺžka MAC, u TLS je to aj *Pseudorandom Function* (PRF),
- master secret,
- značka znovupoužitelnosti.

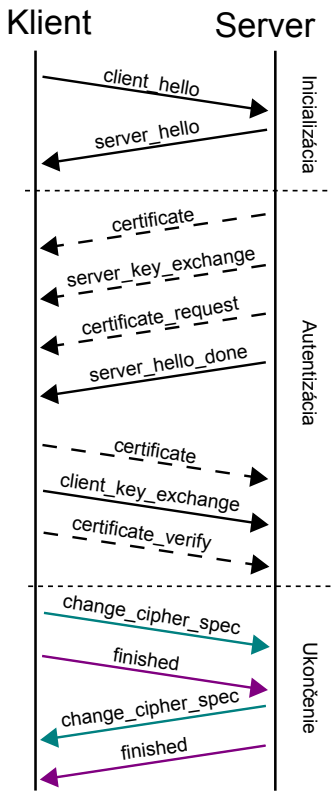
Tieto parametre sa potom použijú na vygenerovanie zdieľaného kľúča, ktorý sa následne použije na šifrovanie všetkých správ prenášaných v danej relácii.

Hello správy V inicializačnej fáze klient posiela správu *Client Hello*, na ktorú server musí odpovedať správou *Server Hello*. Tieto spávy sú použité na zvolenie najbezpečnejších možností, ktoré podporujú obe strany. Vyjednajú sa nasledujúce parametre:

- verzia protokolu,
- ID relácie,

[4] http://documentation.microfocus.com/help/index.jsp?topic=%2Fcom.microfocus.eclipse.infocenter.edtest%2FBKCJCJATTAS006.html

- `CipherSuite`,
  - metóda na výmenu kľúčov,
  - autentizačná metóda,
  - symetrická šifra,
  - Message Authentication Code (MAC), na kontrolu integrity,
- kompresný algoritmus,
- náhodné hodnoty (`client.random` a `server.random`)



Obrázok 1.3: Prehľad komunikácie *SSL/TLS Handshake*

Po výmene *Hello* správ nasleduje autentizačná časť. Táto fáza je v RFC definovaná ako nepovinná. Autentizačnú časť môžeme rozdeliť na autentizáciu serveru a autentizáciu klienta. Autentizácia serveru je obalená medzi správy *Server Hello* a *Server Hello Done*. Autentizácia klienta prebieha vždy až po autentizácii serveru.

Certificate Server teda pokračuje v posielaní správy *Certificate*. Táto správa obsahuje sekvenciu certifikátov. Na začiatku sekvencie je odosielateľov certifikát a každý ďalší certifikát musí autentizovať ten predchádzajúci. Keďže validácia certifikátov je založená na koreňových certifikátoch, ktoré nie sú podpísané, tak koreňový certifikát v tejto sekvencii nie je. Očakáva sa, že druhá strana tento certifikát má.

Server Key Exchange Následne môže byť odoslaná správa *Server Key Exchange*, ak server nemá certifikát alebo certifikát je určený len pre podpisovanie. Táto správa teda obsahuje nejaký verejný kľúč, ku ktorému má server súkromný kľúč.

Certificate Request Server môže vyžiadať autentizáciu po klientovi pomocou správy *Certificate Request*. Následne server odošle správu *Server Hello Done*, ktorá indikuje ukončenie inicializačnej fázy *Handshake* protokolu.

Client Key Echange Na správu *Certificate Request* klient musí odpovedať správou *Certificate*. Inak je odoslaná správa *Client Key Exchange*, ktorej obsahom je zašifrovaná hodnota *pre-master secret* verejným kľúčom algoritmu zvoleného v inicializačnej fáze. V prípade použitia Diffie-Hellman Ephemeral [14] správa obsahuje zašifrovaný verejný exponent tohto algoritmu. Pri použití statickej verzie Diffie-Hellman sa posiela certifikát obsahujúci statický exponent a táto správa musí byť prázdna.

Pomocou hodnoty *pre-master secret* sa vygeneruje *master secret* používaný na vypočítanie tzv. *key material* (viď Kapitolu 1.2.1). Z *key material* sa potom časť použije ako kľúč symetrickej šifry, časť ako inicializačný vektor a časť ako kľúč HMAC funkcie.

Server by z bezpečnostných dôvodov nemal reagovať na zlyhanie spracovania *pre-master secret*. Naopak pokračuje sa s náhodne vygenerovanou hodnotou.

Certificate Verify Správa *Certificate Verify* sa posiela ak klient poslal certifikát s možnosťou podpisu. Touto správou klient potvrdí, že je držiteľom súkromného kľúča tohto certifikátu. V tejto správe sa použijú všetky správy *Handshake*-u odoslané a prijaté do tejto chvíle.

Change Cipher Spec V ukončovacej fáze sa odosiela správa *Change Cipher Spec*, ktorá oznamuje, že nasledujúce správy budú odosielané šifrovane. Táto správa nepatrí do *Handshake* protokolu.

Finished Správa *Finished* je poslednou správou *Handshake* protokolu a zároveň prvou šifrovanou správou. Je šifrovaná šifrou, ktorá je inicializovaná dohodnutými parametrami. Ako odpoveď server odošle tiež správu *Change Cipher Spec* a rovnako aj *Finished*. Po tejto výmene klient a server majú nadviazavané šifrované spojenie a môžu pomocou neho komunikovať.

1.2.1 Matematika v SSL/TLS

Pseudonáhodná funkcia Pseudonáhodná funkcia (PRF) v TLS je založená na HMAC. Pre všetky `CipherSuites` (metóda výmeny kľúčov, šifra a MAC) založené na RFC 5246 [6] platí nasledujúca PRF s použitím SHA256 ako *P_hash* (1.1). Všetky nové `CipherSuites` musia PRF explicitne špecifikovať.

$$\begin{aligned} P\_hash(secret, seed) = HMAC\_hash(secret, A(1) + seed) + \\ HMAC\_hash(secret, A(2) + seed) + \\ HMAC\_hash(secret, A(3) + seed) + ... \end{aligned} \tag{1.1}$$

A() je definované podľa Rovnice 1.2

$$\begin{aligned} A(0) &= seed \\ A(i) &= HMAC\_hash(secret, A(i-1)) \end{aligned} \tag{1.2}$$

PRF je vytvorená aplikovaním *P_hash* podľa Rovnice 1.3.

$$PRF(secret, label, seed) = P\_ < hash > (secret, label + seed) \tag{1.3}$$

kde *label* je ACSII string.

Podľa W. Stallings, 1998 [17], aby bola PRF čo najbezpečnejšia, používa dva hash algoritmy 1.4. To sa vzťahuje k TLS v1.0 a TLS v 1.1.

$$PRF(secret, label, seed) = P\_MD5(S1, label + seed) \oplus P\_SHA - 1(S2, label + seed) \tag{1.4}$$

kde S1 a S2 je *secret* (*pre-master* alebo *master*) hodnota rozdelená na dve polovice.

Master secret *Master secret* sa vypočíta podľa rovnice 1.5.

$$master\_secret = PRF(pre\_master\_secret, \text{"}master\,secret\text{"}, ClientHello.random + ServerHello.random)[0..47] \tag{1.5}$$

Reťazec *"master secret"* sa použije ako časť semienka v PRF (viď Rovnica 1.3). Po vypočítaní *master secret* by sa mal *pre-master secret* z bezpečnostných dôvodov čo najskôr odstrániť z pamäte. *Key material* sa vypočíta podľa Rovnice 1.6.

$$key\_block = PRF(master\_secret, \text{"}key\,expansion\text{"}, server\_randomclient\_random) \tag{1.6}$$

Opäť reťazec *"key expansion"* sa použije ako časť semienka PRF. Časť tohto *key_block*-u sa potom použije ako MAC kľúč, kľúč symetrickej šifry a inicializačný vektor. Pre každú stranu, klient a server, je vybratá samostatná časť *key_block*-u ako kľúč. To znamená, že celková dĺžka *key_block*-u závisí od zvoleného `CipherSuite` podľa Rovnice 1.7.

$$key\_block\_len = 2 * MAC\_key\_len + 2 * key\_len + 2 * IV\_len \tag{1.7}$$

kde *IV_len* sa rovná veľkosti bloku šifrovacej funkcie.

U FORTEZZA algoritmu výmeny kľúčov v SSLv3 sa *master secret* použije len na MAC, kľúče relácie sú generované klientom a odoslané v *Client Key Exchange* správe.

Dĺžka *master secret* hodnoty je vždy 48 bajtov, to je znázornené aj v Rovnici 1.5. *Pre-master secret* na rozdiel od *master secret* môže byť rôzne dlhé podľa toho, aká metóda výmeny kľúčov bola použitá. Napríklad pri Diffie-Hellman sa z vypočítanej hodnoty odstrihnú počiatočné nuly a iba to, čo ostalo sa použije ako *pre-master secret*. U RSA sa vygeneruje 48 bajtová hodnota, ktorá sa zašifruje verejným kľúčom serveru.

1.3 Analýza šifier na výmenu kľúčov relácie

Existujú dva prístupy výmeny kľúčov. Môže sa jednať o distribúciu verejných kľúčov asymetrických šifier alebo využitie asymetrickej kryptografie na výmenu tajných kľúčov. Tajný kľúč je kľúč použitý v symetrickej kryptografii. Používa sa pomenovanie tajný kľúč, pretože má ostať utajený pre každého, okrem komunikujúcich strán. V SSL/TLS sa používajú oba prístupy.

Prvý prístup, teda výmena verejných kľúčov asymetrickej kryptografie sa využíva pri autentizácii. Deje sa to, keď server/klient posiela správu *Certificate*. Certifikát obsahuje okrem iného, verejný kľúč serveru/klienta podpísaný súkromným kľúčom CA, ktorej dôveruje protistrana. Súkromný kľúč CA je podpísaný inou CA a tak ďalej až ku koreňovej CA, ktorej súkromný kľúč ostáva nepodpísaný. Tomu sa hovorí *chain-of-trust*.

Druhý prístup, výmena tajných kľúčov pomocou asymetrickej kryptografie je v SSL/TLS vidieť v správe *Client Key Exchange*. Zo správy *Certificate*, či *Server Key Exchange* sa získal verejný kľúč serveru a týmto kľúčom sa zašifroval *pre-master secret*, z ktorého sa potom vypočíta tajný kľúč.

Najčasťejšie používanými sú dnes algoritmy RSA a Diffie-Hellman. V prípade použitia Diffie-Hellman na výmenu kľúčov sa tento algoritmus kombinuje s iným algoritmom, ktorý zabezpečuje o autentizáciu. V tejto časti sa čitateľ oboznámi s princípom algoritmu RSA, nasledovať bude algoritmus Diffie-Hellman a jeho varianta nad eliptickými krivkami.

1.3.1 Algoritmus RSA

Algoritmus pomenovaný po jeho stvoriteľoch - The Rivest-Shamir-Adleman podporuje šifrovanie, podpisovanie a výmenu kľúčov. Má nasledujúcu schému:

- p, q (ľubovoľne zvolené prvočísla, súkromné),
- $n = pq$ (verejne známy modulus),
- e s gcd($\Phi(n)$,e) = 1; $1 < e < \Phi(n)$ (verejný kľúč, zvolený),
- $d \equiv e^{-1} \bmod \Phi(n)$ (súkromný kľúč),

kde $\Phi(n)$ je počet kladných celých čísel menších ako n a zároveň je to vzhľadom k n relatívne prvočíslo.W. Stallings, 1998 [17] v Kapitole 7 ukazuje, že pre prvočísla p, q platí rovnosť $\Phi(n) = (p-1)(q-1)$.

Súkromný kľúč pozostáva z {d,n}, verejný kľúč z {e,n}. Ak používateľ A zverejní svoj verejný kľúč užívateľovi B, ktorý chce poslať užívateľovi A správu M, potom zašifrovaná správa C sa vypočíta ako $C = M^e \bmod n$. Užívateľ A potom zašifrovanú správu rozšifruje takto $M = C^d \bmod n$.

1.3.2 Algoritmus Diffie-Hellman

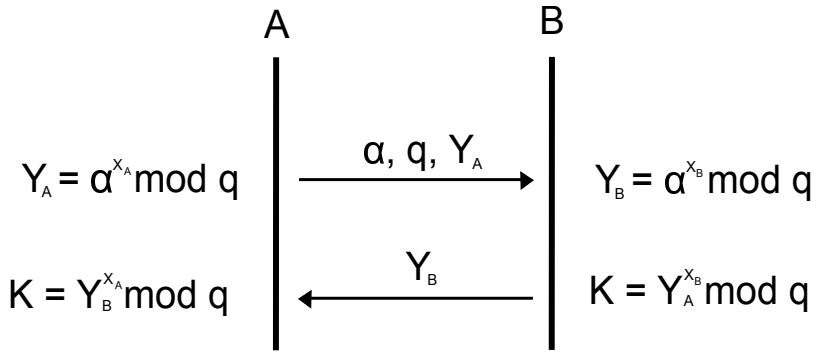
Účelom tohto algoritmu je bezpečná výmena tajných kľúčov cez nezabezpečené médium. Algoritmus má nasledujúcu schému:

- obe strany získajú globálne verejné prvky q a α, kde q je prvočíslo a α je jeho primitívnym koreňom (napríklad z certifikátu),
- strana A vygeneruje súkromný exponent X_A a vypočíta verejné číslo $Y_A = \alpha^{X_A} \bmod q$, ktoré odošle strane B,
- strana B vygeneruje svoj súkromný exponent X_B a vypočíta verejné číslo $Y_B = \alpha^{X_B} \bmod q$, ktoré odošle strane A,
- strana A vypočíta tajný kľúč ako $K = (Y_B)^{X_A} \bmod q$,
- strana B vypočíta tajný kľúč ako $K = (Y_A)^{X_B} \bmod q$.

Podľa pravidiel modulárnej aritmetiky obe strany takto získajú rovnaký kľúč $K = \alpha^{X_A * X_B} \bmod q$. Tento algoritmus sa spolieha na zložitosť výpočtu diskrétneho logaritmu, tzn. ak útočník chce získať súkromný exponent, musí vypočítať diskrétny logaritmus z verejného čísla so základom α vzhľadom k modulu q. Akonáhle strana vypočíta hodnotu K mala by svoj súkromný exponent čo najskôr odstrániť z pamäte.

Kryptografia nad eliptickými krivkami je definovaná pomocou aritmetických operácií nad nejakou zvolenou krivkou. Pri variante algoritmu Diffie-Hellman nad eliptickými krivkami je schéma nasledujúca:

- výber parametov a a b, ktoré určujú eliptickú krivku a generujúci bod G z tejto skupiny (globálne známe parametre),
- strana A si zvolí celé súkromné číslo $n_A < n$ a vypočíta verejné číslo $P_A = n_A \times G$,
- podobne strana B si zvolí svoje súkromné číslo n_B a vypočíta P_B,



Obrázok 1.4: Schéma algoritmu Diffie-Hellman

- súkromný kľúč je potom vypočítaný ako $K = n_A \times P_B$, respektíve $K = n_B \times P_A$,

kde n dostaneme z $nG = O$, kde O je tzv. *zero point* alebo bod aditívnej identity. Bezpečnosť algoritmov nad eliptickými krivkami sa spolieha na to, že je pomerne ťažké vypočítať k zo známych hodnôt G a kG. Tento problém je známy ako problém logaritmu eliptických kriviek.

1.3.3 Získanie prístupu k šifrovaným dátam

Pre získanie prístupu k dátam prenášaným v SSL/TLS existujú dva základné prístupy, ktoré spomínajú S. Davidoff a J. Ham, 2012 [3]:

- Prvou možnosťou je získať privátny kľúč serveru na získanie kľúčov relácie a dešifrovanie obsahu. To ale závisí na algoritme použitom na výmenu kľúčov a taktiež je nutný prístup k súkromnému kľúču serveru.
- Druhou možnosťou je ukončiť TLS/SSL reláciu niekde na proxy a nadviazať novú SSL/TLS reláciu smerom od proxy ku klientovi. Pri tomto prístupe je dôležité mať prístup ku klientskej stanici za účelom nainštalovania certifikátu proxy serveru. Inak užívateľ klientskej stanice môže dostať upozornenie od systému na potenciálny útok Man-In-The-Middle.

Ak existuje prístup k súkromnému kľúču serveru, pomocou neho je možné dešifrovať hodnotu *pre-master secret* odoslanú klientom a získať kľúče relácie. Na to je potrebné mať prístup ku kompletnému obsahu *Handshak*-u SSL/TLS a následnej výmene dát vytvorenou reláciou. Súkromný kľúč serveru pri dešifrovaní prevádzky nepomôže, ak bol na výmenu kľúčov zvolený algoritmus Diffie-Hellman Ephemeral. Ten používa špeciálnu schému generovania hodnoty *pre-master secret*, kde pre každú novú reláciu generuje nové súkromné hodnoty. Táto verzia sa používa v kombinácii s iným asymetrickým algoritmom, ktorý zabezpečí autentizáciu. Teda verejné hodnoty Diffie-Hellman sú podpísané súkromným kľúčom.

Princíp s proxy funguje tak, že celá komunikácia s vonkajším svetom smeruje cez proxy. Ak chce server komunikovať cez SSL/TLS s nejakým serverom, tak SSL/TLS odpovede serveru sú ukončené na proxy a tým pádom proxy udržuje šifrované spojenie so serverom. Na strane ku klientovi proxy poskytuje falošný certifikát serveru a zostaví druhý SSL/TLS tunel. Potom proxy môže prehliadať prevádzku v dešifrovanej podobe alebo posielať túto prevádzku na iný systém k ďalšej analýze.

SSL/TLS je navrhnuté na ochranu proti tomuto typu útoku. Ide vlastne o Man-In-The-Middle. Teda certifikát proxy serveru nebýva podpísaný dôveryhodnou CA a používateľovi na klientskej stanici sa zobrazujú varovania. Tomu sa dá predísť nainštalovaním certifikátu proxy servera medzi dôveryhodné koreňové certifikáty na klientovi (obeti).

SSLsplit[5] je aplikácia vytvorená Daniel Roethlisbergerom a distribuovaná pod zjednodušenou BSD licenciou. SSLsplit ukončuje SSL/TLS spojenia a vytvára nové smerom k pôvodnej cieľovej adrese a zároveň zaznamenáva celú prevádzku. Je možné si zvoliť šifry, ktoré sa majú používať rovnako aj privátny kľúč, ktorým sa majú podpisovať falšované certifikáty. Na správne fungovanie je potrebné nastaviť *IP forwarding*[6].

SSLstrip[7] je nástroj vytvorený Moxie Marlinspike a je distribuovaný pod licenciou GNU GPLv3. Implementuje útok podobný ako SSLsplit, s tým rozdielom, že relácia medzi obeťou a útočníkom prebieha nešifrovane.

Charles Proxy[8] je webové proxy, distrubuované ako shareware. Voľne dostupná je 30-dňová verzia. Na zachytenie SSL/TLS komunikácie využíva princíp Man-In-The-Middle, kedy falšuje pôvodné cerifikáty od serverov a podpisuje ich vlastným certifikátom.

Fiddler[9] je taktiež webové proxy, na rozdiel od Charles Proxy je voľne dostupné, avšak určené pre platformu Windows. Na zachytenie SSL/TLS komunikácie využíva rovnaký princíp ako Charles Proxy.

[5] https://www.roe.ch/SSLsplit
[6] https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt
[7] http://www.thoughtcrime.org/software/sslstrip/
[8] http://www.charlesproxy.com/
[9] http://www.telerik.com/fiddler

Kapitola 2

Návrh

V tejto kapitole bude popísaný návrh algoritmu použitého pre detekciu webmailovej prevádzky. Na začiatku bude v krátkosti popísaný tok dát v nástroji *Netfox.Framework* pre lepšiu predstavu, čo bude vstupom a výstupom jednotlivých modulov. Následne bude popísaný návrh algoritmu. Na konci kapitoly bude navrhnuté grafické rozhranie pre zobrazovanie detekovaných udalostí.

2.1 Návrh algoritmu

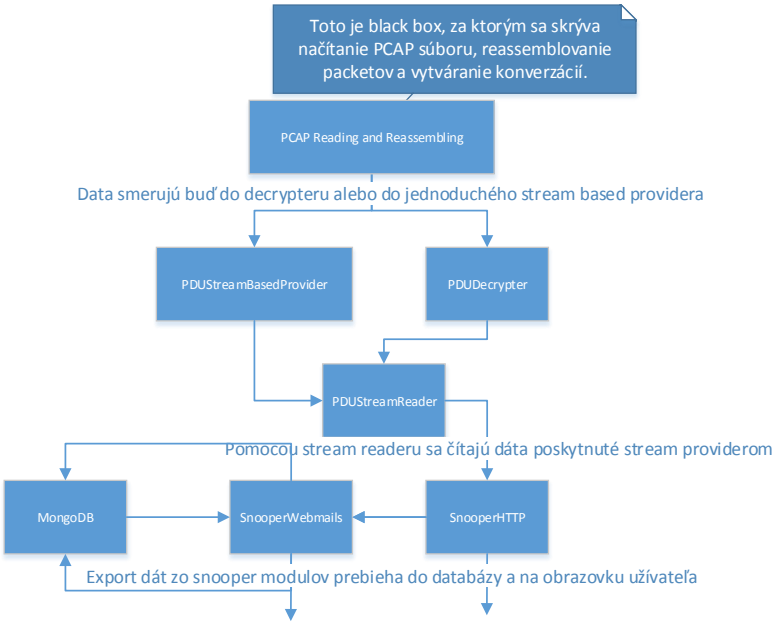
Základný princíp algoritmu skrývajúci sa za *Netfox.Framework*-om je na Obrázku 2.1, pre lepšiu predstavu je v Prílohe B.2 sekvenčný diagram. Je možné si všimnúť množstvo väzieb medzi `SnooperHTTP` a `SnooperWebmails` a databázou. Tieto väzby sú tam preto, lebo webmaily komunikujú protokolom HTTP. `SnooperHTTP` teda exportuje HTTP správy do databázy a `SnooperWebmails` potom môže pracovať nad exportami z databázy alebo priamo nad dátami z modulu `SnooperHTTP`.

Z Obrázku 2.1 je zrejmé, že modul `PDUDecrypter`-u bude v spracovaní postavený na rovnakú úroveň ako modul `PDUStreamBasedProvider`. Konkrétne bude z tejto triedy dediť, aby poskytol modulom využívajúcim jeho služby rovnaké rozhranie. Vnútorne bude `PDUDecrypter` implementovaný ako konečný automat, ktorý bude reagovať na správy SSL *Handshake* protokolu. Keďže SSL/TLS poskytuje množstvo kombinácií algoritmov na výmenu kľúčov a šifrovacích algoritmov, bude vhodné vytvoriť pre všetky algoritmy z jednej kategórie jedno rozhranie kvôli jednoduchej rozšíriteľnosti. Základné dve rozhrania/kategórie, na ktoré je potrebné brať ohľad sú:

- algoritmy na výmenu kľúčov, v zásade asymetrické šifry,
- šifrovacie algoritmy, symetrické šifry - spolu s hash funkciou poskytujúcou MAC.

Okrem týchto dvoch základných kategórií pri šifrovacích algoritmoch je nutné rozlišovať typ šifry, teda či je šifra prúdová alebo bloková. Pri blokovej šifre je ďalej nutné rozlíšiť režim, v ktorom šifra pracuje. Z Kapitoly 1.2 sú známe režimy CBC a GCM prípadne CCM. Posledné dva spomínané módy blokových šifier je možné zlúčiť do jednej kategórie s názvom AEAD. Teda vyššie definované rozhranie pre šifrovacie algoritmy bude musieť poskytnúť rozhranie ako pre prúdové šifry, tak pre blokové šifry v CBC režime a rovnako aj pre blokové šifry v režime AEAD.

Algoritmus detekcie webmailových udalostí v zachytenej komunikácii bude postavený na spoločných znakoch týchto webových aplikácií. Spoločné znaky boli vybrané na základe



Obrázok 2.1: Základný koncept spracovania zachytenej komunikácie *Netfox.Framework*-om.

analýzy z Kapitoly 1.1. Je možné, že niektoré webmaily budú vyžadovať zvláštny prístup, pretože podľa vybraných znakov sa udalosť nedetekuje. Ako vhodné webmailové udalosti boli vybrané nasledujúce:

- nová správa,
- zobrazenie priečinku,
- zobrazenie správy.

Udalosti boli zvolené na základe množstva zaujímavých informácií, ktoré bolo možné z takejto udalosti vyexportovať. Udalosti typu *presun správy medzi zložkami* či *vymazanie správy* sa dejú pomocou identifikátoru správy a preto z takýchto udalostí nebolo možné exportovať zaujímavé informácie. Naviac pri udalostiach typu zobrazenie správy či zobrazenie priečinku mnoho webmailových aplikácií posiela časť kódu, ktorý je problematické tokenizovať. Preto sa pri niektorých detekciách môže stať, že sa udalosť detekuje, ale obsah udalosti nie je nastavený. Zvolené spoločné znaky webmailovej komunikácie budú uložené v bázovej abstraktnej triede analyzátoru, kde ich bude možné prípadne dopĺňať. V Tabuľke 2.1 sú ukázané vybrané znaky.

Algoritmus sa bude spoliehať na funkciu webmailových analyzátorov. Tieto analyzátory vyhodnotia každú HTTP správu, ktorá im bude poslaná a rozhodnú, či ide o webmailovú udalosť alebo nie.

Obsahom HTTP správy môžu byť rôzne štruktúry, napríklad JSON, FastRPC, a pod. Za účelom prehliadania rôznorodého obsahu, analyzátory použijú ďalší submodul pozorovateľa. Pozorovateľ obsahu HTTP správy sa bude dať využiť aj na extrakciu požadovaných častí správy. Bohužiaľ nebude známa kompletná funkcionalita kvôli rôznorodosti protokolov

udalosť	vybrané znaky	
nová správa	názvy polí emailu: "*from*", "*to*", "*subject*", "*body/content*", "*simpleBody*"	
zobrazenie priečinku	volanie metódy "*method*"	názov metódy "*ListMessagesInThread*"
	volanie metódy "*method*"	názov metódy "*ListFolderThreads*"
	volanie metódy "*search*"	názov metódy "*"
	volanie metódy "*m*"	názov metódy "*list*"
	volanie metódy "*user.listMessages*"	názov metódy "*"
zobrazenie správy	volanie metódy "*method*"	názov metódy "*GetDisplayMessage*"
	volanie metódy "*user.message.getAttributes*"	"názov metódy *"

Tabuľka 2.1: Vybrané znaky udalostí spoločné pre vybrané emaily.

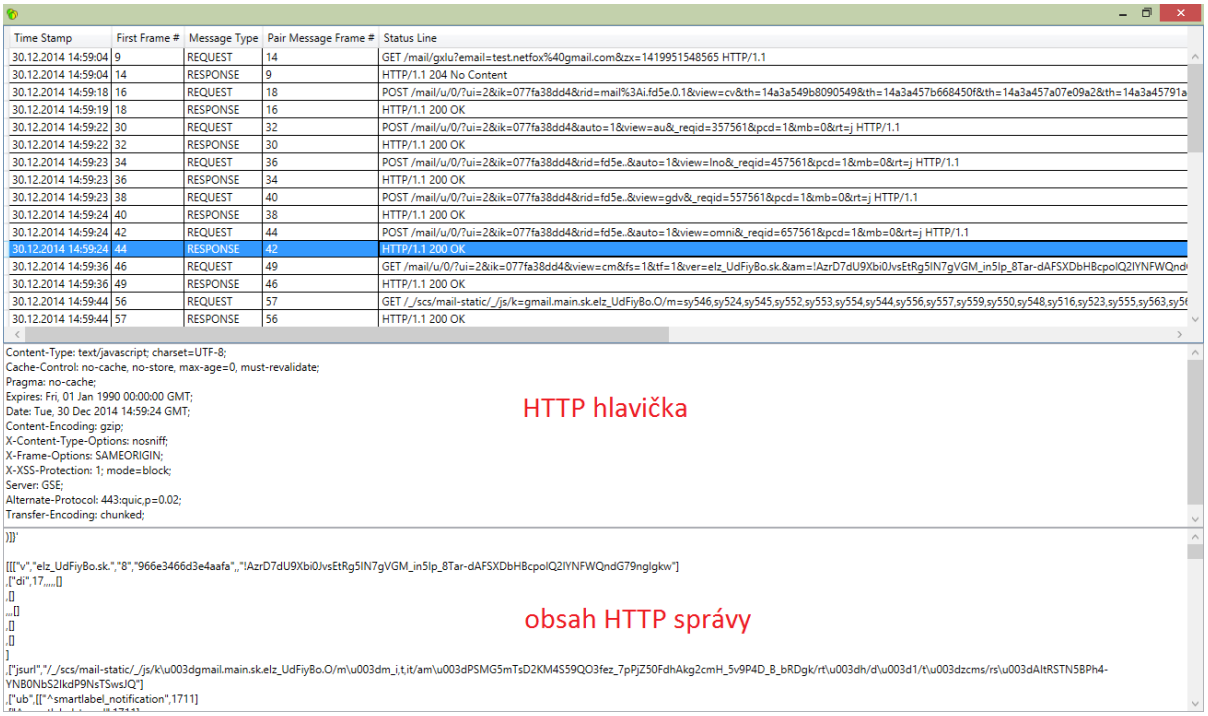
použitých jednotlivými aplikáciami. To je vyriešené návrhovým vzorom Návštevník (bližšie popísaný v Kapitole 3.3.2).

Z textu vyššie a Obrázku 2.1, je zrejmé, že pred samotnou detekciou webmailových udalostí musí prebehnúť detekcia a export HTTP správ. V prípade šifrovanej komunikácie sa táto komunikácia musí ešte dešifrovať. Tieto úlohy majú na starosti moduly `SnooperHTTP` a `PDUDecrypter`. Implementačné detaily týchto modulov sú v Kapitole 3.

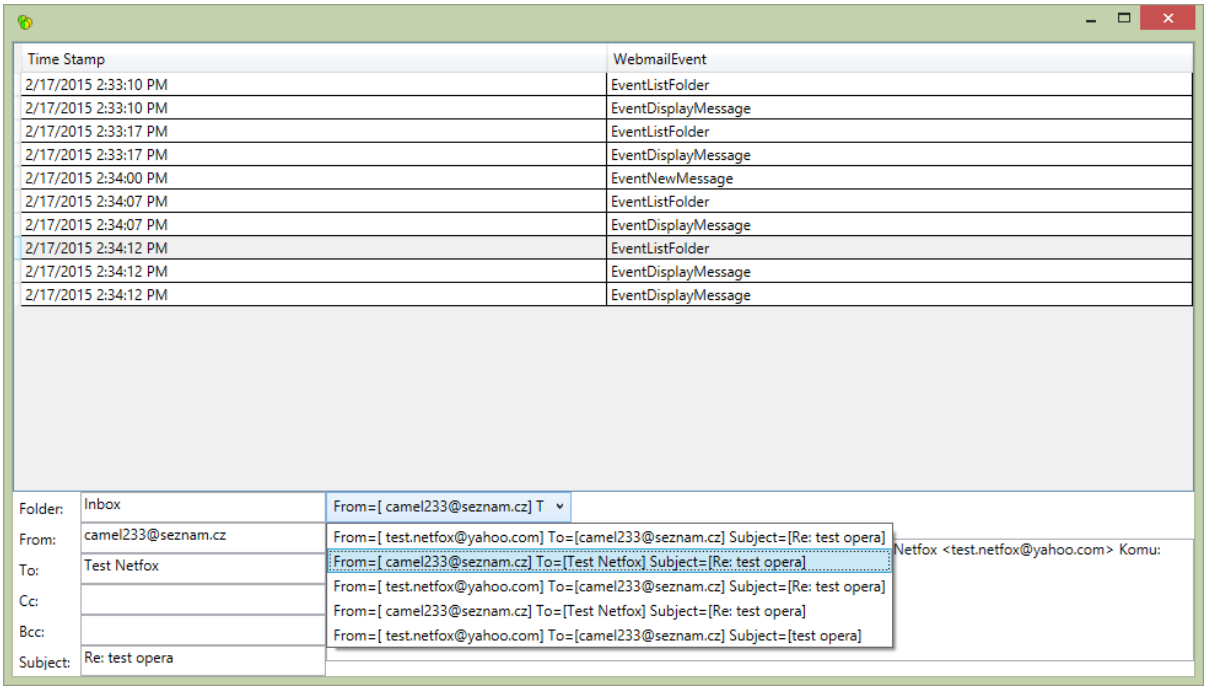
2.2 Návrh užívateľského rozhrania

Užívateľské rozhranie pre HTTP *Snooper* je pomerne jednoduché. Skladá sa z `DataGrid`-u, v ktorom sa zobrazujú jednotlivé HTTP správy. Pod touto tabuľkou sa nachádzajú dve textové polia. V prvom sa zobrazuje obsah HTTP hlavičky a v druhom samotný obsah HTTP správy.

Užiíateľské rozhranie pre Webmailový *Snooper* je taktiež pomerne jednoduché. Rovnako obsahuje `DataGrid`, kde sa zobrazujú detekované udalosti. Po výbere niektorej z udalostí sa naplní `ComboBox` správami, ktoré obsahuje vybraná udalosť. Po výbere niektorej správy zo spomínaného `ComboBox`-u sa naplnia polia zobrazujúce obsah správy.



Obrázok 2.2: Ukážka rozhrania HTTP *Snooper*-a.



Obrázok 2.3: Ukážka rozhrania *Snooper*-a webmailov.

Kapitola 3

Implementácia

Projekt *Netfox.Framework* [13] je modulárny systém, ktorý zabezpečuje spracovanie PCAP súborov a ich analýzu. Súčasťou frameworku je *reassembling* packetov (spojenie súvisiacich rámcov do jednej štruktúry, vysporiadanie sa so znovuzasielaním a pod.) a vytváranie konverzácií. Konverzácia v *Netfox.Framework*-u je identifikovaná zdrojovou a cieľovou IP adresou, zdrojovým a cieľovým portom a protokolom sieťovej vrstvy. Konverzácie sú aproximácie TCP relácií, získaných z komunikácie, ktorá môže byť aj nekompletná. Cieľom tejto kapitoly je oboznámiť čítateľa s konkrétnou implementáciou modulov, ktorých návrh bol popísaný v Kapitole 2.

3.1 SSL Decrypter

Pred vstupom do modulov, ktoré analyzujú danú komunikáciu sa reassemblované PDU načítajú pomocou modifikovanej triedy `System.IO.Stream`. Trieda implementujúca dešifrovanie bude postavená na rovnaké miesto v spracovaní. Vstupom teda sú reassemblované L7 PDU, ktoré sú obsahom konverzácie. A výstupom sú dešifrované dáta ako prúd bytov.

Ako implementáciu konkrétnych šifrovacích algoritmov je možné použiť C# *namespace* `System.Security.Cryptography`[1], ktorý poskytuje množstvo kryptografických algoritmov. Prednosť ale bola daná knižnici Bouncy Castle [2]. Táto knižnica okrem jednoduchého rozhrania pre kryptografické algoritmy poskytuje naviac, na rozdiel od *namespace* `Sustem.Security.Cryptography` taktiež rozhranie pre parsovanie PEM formátu súborov. Na Obrázku 3.1 je stavový diagram, ktorý reprezentuje princíp algoritmu. Vstupným bodom je metóda `NewMessage()`, ktorá pri každom zavolaní dešifruje jednu správu a pripraví ju na čítanie. Návratová hodnota metódy je `true`, ak existuje ďalšia dešifrovaná správa na čítanie, `false` ak nebolo možné dešifrovať žiadnu ďalšiu správu. Táto metóda principiálne implementuje konečný automat, aktuálne so štyrmi stavmi:

- `Init` - decrypter zatiaľ nie je nainicializovaný parametrami relácie,
- `Negotiation` - decrypter je nainicializovaný algoritmami, ale čaká sa na vyjednanie tajného kľúča (správa *Client Key Exchange*),
- `Intermezzo` - medzistav po inicializácii, v ktorom sa dešifruje prvá správa,
- `Data Exchange` - v tomto stave sa dešifrujú správy.

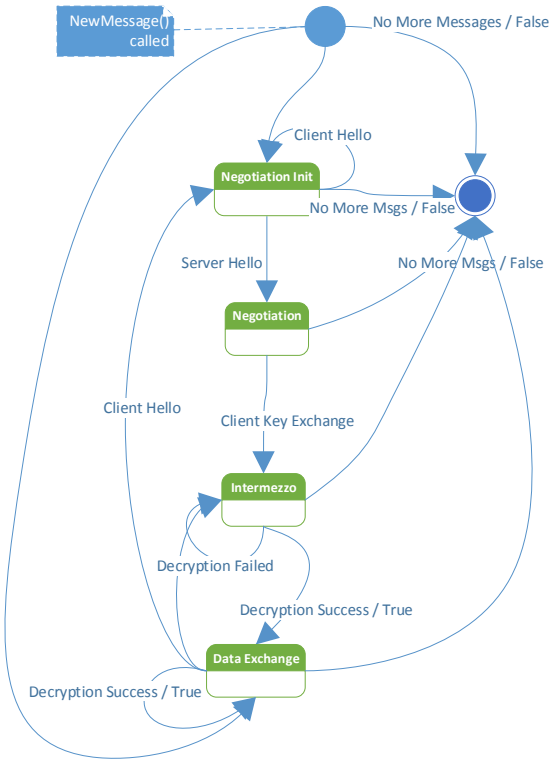
[1] https://msdn.microsoft.com/en-us/library/system.security.cryptography%28v=vs.110%29.aspx

Ak bude potrebné zohľadniť autentizačnú časť, je možné pridať ďalší stav pred `Intermezzo`, v ktorom sa budú spracovávať požadované správy. V momentálnej implementácii je podporovaný algoritmus na výmenu kľúčov RSA, pri ktorom z hľadiska dešifrovania autentizačnú časť nie je potrebné riešiť.

Tento stavový automat je uzavretý v cykle, ktorý sa ukončí po prechode do stavu `Data Exchange`, aby pri každom zavolaní metódy `NewMessage()` bola dešifrovaná nejaká správa.

V prvom stave `Init` sa čaká na *Hello* správy od klienta a od serveru, z ktorých sa získajú potrebné parametre (náhodné čísla, a *CipherSuite*). Z popisu *CipherSuite* sa nainicializuje trieda `KeyDecrypter`, zabezpečujúca dešifrovanie kľúča (*pre-master secret*) a trieda `DataDecrypter`, ktorá zabezpečí dešifrovanie samotných správ. Tieto dve triedy sú abstraktné a ich implementácia závisí od konkrétneho šifrovacieho algoritmu. Ako bolo vyššie spomenuté, ako implementácia šifrovacích algoritmov bola použitá knižnica BouncyCastle [2].

Po prijatí správy *Server Hello* sa prechádza do stavu `Negotiation`. V tomto stave automat zotrvá po dobu, kým nedostane správu *Client Key Exchange*. Pomocou triedy `KeyDecrypter` sa dešifruje hodnota *pre-master secret*. Z tejto hodnoty sa pomocou PRF popísanej v RFC 2246 [4] pre TLS v1.0 a TLS v1.1, a v RFC 5246 [6] pre TLS v1.2 získa tzv. *key material*. Pomocou tohto materiálu sa nainicializujú HMAC a symetrická šifra v triede `DataDecrypter`.



Obrázok 3.1: Stavový automat modulu `PDUDecrypter`.

Medzi prechodom z inicializačných stavov do stavov, ktoré dešifrujú komunikáciu, je potrebné dešifrovať taktiež správu *Finished*. To je dôležité pri použití prúdových šifier, napr. RC4, ktorých vnútorný stav sa mení (de)šifrovaním každého bytu. *Finished* správa je šifrovaná, jediné čo je možné zistiť po prijatí je, že patrí do *Handshake* protokolu. Z Kapitoly 1.2 vieme, že je odosielaná bezprostredne po správe *Change Cipher Spec*. Na to sa spolieha aj algoritmus.

Po tomto kroku algoritmus prechádza do stavu `Intermezzo`. Tento stav vykonáva rovnaké kroky ako stav `Data Exchange`. Ak by automat prechádzal zo stavu `Negotiation` priamo do stavu `Data Exchange`, cyklus v ktorom je stavový automat uzavretý (a metóda `NewMessage()`) by mohol skončiť bez dešifrovania správy. Zároveň tento stav rieši situáciu ak sa do konverzácie dostane nejaká nedešifrovateľná správa. Teda automat sa neprepne do stavu `Data Exchange`, kým sa nepodarí dešifrovať nejakú správu.

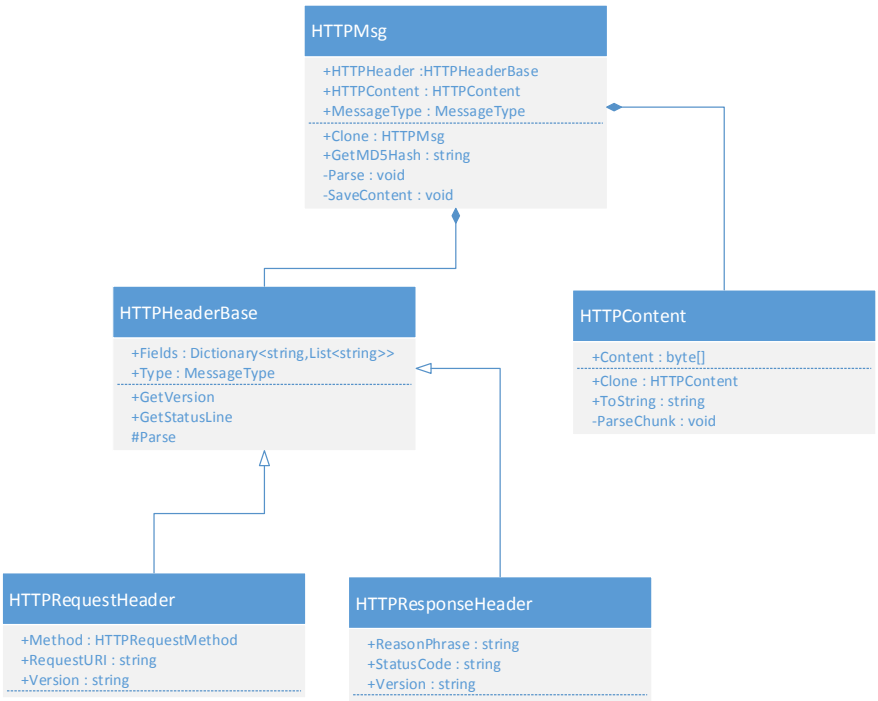
Po prechode do stavu `Data Exchange` sa cyklus automatu skončí. Stav však ostane zachovaný a pri ďalšom volaní metódy `NewMessage()` sa vykoná cyklus iba raz a jeho výstupom bude dešifrovaná správa. Automat však môže prejsť do iného stavu aj keď je už

v `Data Exchange`. Napríklad v prípade SSL *Alert* protokolu, či zmene parametrov relácie.

Pri parsovaní SSL/TLS správy modul zisťuje koľko bytov SSL/TLS správa obsahuje. Tu sa môže vyskytnúť situácia, kedy v reassemblovanom PDU nie je dostatok bytov, respektíve je ich viac. Popisovaný modul túto situáciu rieši tak, že skontroluje ohlásenú dĺžku SSL/TLS správy a porovná ju s množstvom bytov, ktoré dostane zo správy. Ak ich nie je dosť, dáta uloží do dočasného poľa (`ContinuationData`) a dáta z nasledujúceho paketu sa potom prilepia za ne. Podobne je vyriešené, ak je dát v správe viac ako je treba, s tým rozdielom, že dáta z pôvodnej správy sa orežú a prebytočné dáta sa po prijatí nasledujúcej správy prilepia pred ňu.

3.2 HTTP Snooper

HTTP Snooper je jeden z modulov *Netfox.Framework*-u, ktoré analyzujú zachytenú komunikáciu. Úlohou tohto modulu je zo zachytenej komunikácie získať časť, ktorá používa protokol HTTP. Vstup tohto modulu typu *Snooper* je štruktúra konverzácie, ako bolo popísané na začiatku Kapitoly 3. Výstupom je množina rozparsovaných HTTP správ pripravených na export do databázy.



Obrázok 3.2: Diagram tried pre HTTP *Snooper*.

Na začiatku analýzy sa *Snooper* musí rozhodnúť, či sa budú dáta konverzácie dešifrovať alebo nie. Toto rozhodnutie spraví na základe existencie priradeného privátneho kľúča ku skúmanej konverzácii. Následne vytvorí inštanciu triedy `PDUStreamReader`, ktorá zabezpečí čítanie daného streamu. Pri čítaní streamu z `PDUDecrypter`-u a z klasického `PDUStreamBasedProvider` je rozdiel v tom, že pri použití `PDUDecrypter`-u je prvá správa pripravená na čítanie až po volaní metódy `NewMessage()`. Samotné parsovanie správ sa

potom odohráva v triede `HTTPMsg`, ktorá prijíma v konštruktore práve inštanciu triedy `PDUStreamReader` a zložku na export dát, ktoré sa nebudú ukladať do databázy.

Algoritmus za triedou `HTTPMsg` je pomerne jednoduchý. Základný koncept použitých tried môžeme vidieť na Obrázku 3.2. Na začiatku prebieha detekcia, či vlastne ide o HTTP správu. To sa deje na základe prvého riadku správy. Prvý riadok HTTP správy je tzv. *status line*. Tento riadok môže mať rôznu podobu v závislosti od toho, či ide o žiadosť alebo odpoveď.

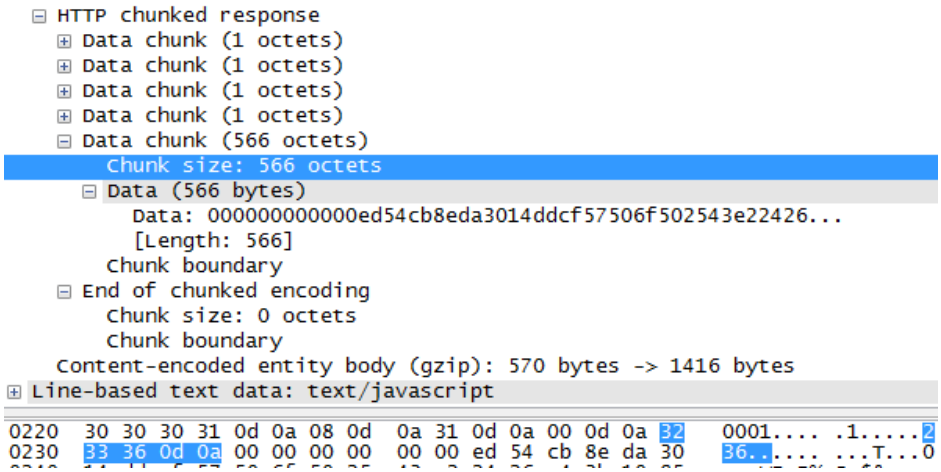
V prípade žiadosti sa stavový riadok začína názvom HTTP metódy. V RFC 2616 [9] je ich definovaných niekoľko, najčastejšie sa stretneme s metódami `GET`, `POST`, `PUT`, `DELETE`, `OPTIONS`, `CONNECT`. Nasleduje *Request URI* a verzia protokolu. Jednotlivé časti sú oddelené medzerou. Celý riadok je ukončený znakmi `CR LF`.

V prípade odpovede sa stavový riadok začína verziou protokolu, nasleduje *status code* a *status phrase*. Rovnako ako pri žiadosti je riadok ukončený znakmi `CR LF`.

Ak sa prvý riadok nezhoduje z očakávaným vzorom, algoritmus prehlási, že nejde o HTTP správu a parsovanie sa ukončí. Inak sa pokračuje v parsovaní HTTP hlavičky. Tá sa skladá z množiny hodnôt oddelených znakmi `CR LF`. Posledná hodnota je od obsahu správy oddelená dvojicou znakov `CR LF`.

Po načítaní hlavičky je ešte potrebné vyparsovať obsah správy. Nie všetky HTTP správy ale majú nejaký obsah. Okrem toho, samotný obsah či dokonca prenos môže byť kódovaný. V prípade kódovania prenosu hlavička správy obsahuje položku `Transfer-Encoding`. Dnes sa najčastejšie stretneme s kódovaním označovaným *chunked* prípadne *identity*. Ukážka ako vyzerá *chunked* prenos je na Obrázku 3.3.

Na začiatku každého *chunk*-u je údaj o tom, koľko bytov je v danom *chunku* nasledovaný samotnými dátami. Každá časť *chunk*-u je oddelená znakmi `CR LF`. Posledný prenášaný *chunk* má veľkosť 0 bytov. V prípade, keď sa nepoužilo kódovanie prenosu, môžme v hlavičke čakať pole `Content-Length` s informáciou koľko bytov má obsah.



Obrázok 3.3: Ukážka *chunked* prenosu.

Z poľa `Content-Encoding` zistíme kódovanie obsahu. Najčastejšie používané kódovanie je *gzip* alebo *deflate*. Pre polia `Content-Length` a `Transfer-Encoding` sú definované isté pravidlá v RFC 2616 [9], z ktorých vyplýva, že ak je prenášané pole `Transfer-Encoding`, ktoré neobsahuje hodnotu *identity*, tak pole `Content-Length` nie je prenášané vôbec.

Keďže obsahom HTTP správy môže byť relatívne veľké množstvo dát, nie je vhodné ho ukladať do databázy. Vhodnejšie je obsah uložiť v nejakej podobe na disk a potom

v prípade potreby ho odtiaľ získať. Pre tento účel sa v triede `HTTPMsg` nachádza metóda `GetMD5Hash()`, aby bolo možné pomenovať súbory s obsahom jednotlivých správ jednoznačne. Tento hash sa prepočítava z reťazca vytvoreného z časového razítka, čísel rámcov, ktoré obsahujú danú HTTP správu a zo *status line* tejto HTTP správy. Ďalej bolo nutné upraviť spôsob, akým sa získava člen `HTTPContent` z triedy `HTTPMsg`.

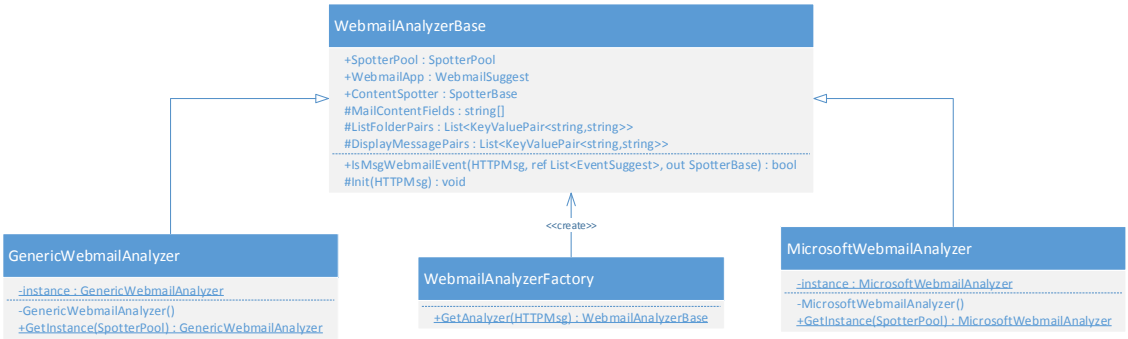
3.3 Webmails Snooper

Webmailový *Snooper* je ďalší z modulov *Netfox.Framework*-u, ktorý analyzuje zachytenú komunikáciu. Tento modul má za úlohu detekovať v komunikácii webmailovú aktivitu a v prípade úspešnej detekcie takýchto udalostí ich exportovať a interpretovať užívateľovi. Vstupom do modulu sú HTTP správy vyexportované modulom `SnooperHTTP`. Výstupom sú štruktúry webmailových udalostí. Pri detekcii webmailových udalostí sú pre tento modul zaujimavé iba HTTP žiadosti, konkrétne metódy `POST` a `GET`, ako vyplýva z Kapitoly 1.1. Takže zo všetkých vstupných správ si *Snooper* vyberá iba žiadosti.

S každou novou správou sa `SnooperWebmails` pýta analyzátoru, či daná správa je webmailová udalosť. Bázová trieda analyzátoru si uchováva zvolené spoločné znaky webmailových aplikácií a na tieto znaky sa vhodne dotazuje pozorovateľa. V prípade zhody sa uloží typ detekovanej udalosti, ktorý si vyzdvihne *Snooper* a na základe týchto detekcií vytvorí exportovateľné štruktúry typu `WebmailEvent`. Kompletný diagram tried je možné vidieť v Prílohe B.1.

3.3.1 Analyzátor webmailov

V okamihu, keď sa vybrala vhodná HTTP správa, môže sa pristúpiť k výberu správneho analyzátoru. Vzhľadom na to, že sa analyzátor vyberá pre každú správu, je vhodné uvažovať o obmedzenom počte inštancií rôznych analyzátorov. Keďže analyzátor má v podstate iba jednu funkciu a to zistiť, či správa je webmailová udalosť, `IsMessageWebmailEvent()`, je možné využiť návrhový vzor *jedináčik*. Výber analyzátoru prebieha podľa detekcie webmailovej aplikácie, pre ktorú sa implementoval špecifický analyzátor. Takúto detekciu musí zaistiť užívateľ (ten, kto implementoval nový analyzátor). Detekcia bude zrejme prebiehať na základe obsahu HTTP správy. Vďaka implementácii oproti rozhraniu je možné výber vhodného analyzátoru skryť v jednoduchej továrni, `WebmailAnalyzerFactory`.



Obrázok 3.4: Diagram tried balíku `WebmailAnalyzers`.

Všetky analyzátory musia dediť z triedy `WebmailAnalyzerBase`, aby poskytli rovnaké rozhranie pre ich užívateľa:

- metóda `Init(HTTPMsg)` - metóda slúži na inicializovanie analyzátoru, teda inicializuje potrebného pozorovateľa obsahu, ak je čo pozorovať, je to protected člen,
- metóda `IsMsgWebmailEvent()` - abstraktná metóda slúži na detekciu, či daná HTTP správa je webmailová udalosť, z tejto metódy by sa mala volať metóda `Init()`. Zároveň je v tejto metóde priestor na nastavenie parametru `WebmailApp`. Návratový typ je `bool` a ak je `true`, tak HTTP správa obsahuje webmailovú udalosť, inak je `false`. Ako druhý parameter sa tejto metóde posiela štruktúra zoznam (ideálne prázdny), ktorý táto metóda naplní. A tretím parametrom je výstupný parameter typu pozorovateľ, čím sa užívateľovi vráti nainicializovaný pozorovateľ použitý na detekciu,
- člen `SpotterPool` - fond pozorovateľov, z ktorého sa získavajú pozorovatelia v danom analyzátore,
- člen `WebmailApp` - indikuje konkrétnu webmailovú aplikáciu za účelom lepšej extrakcie obsahu,
- člen `ContentSpotter` - naposledy použitý spotter v danom analyzátore.

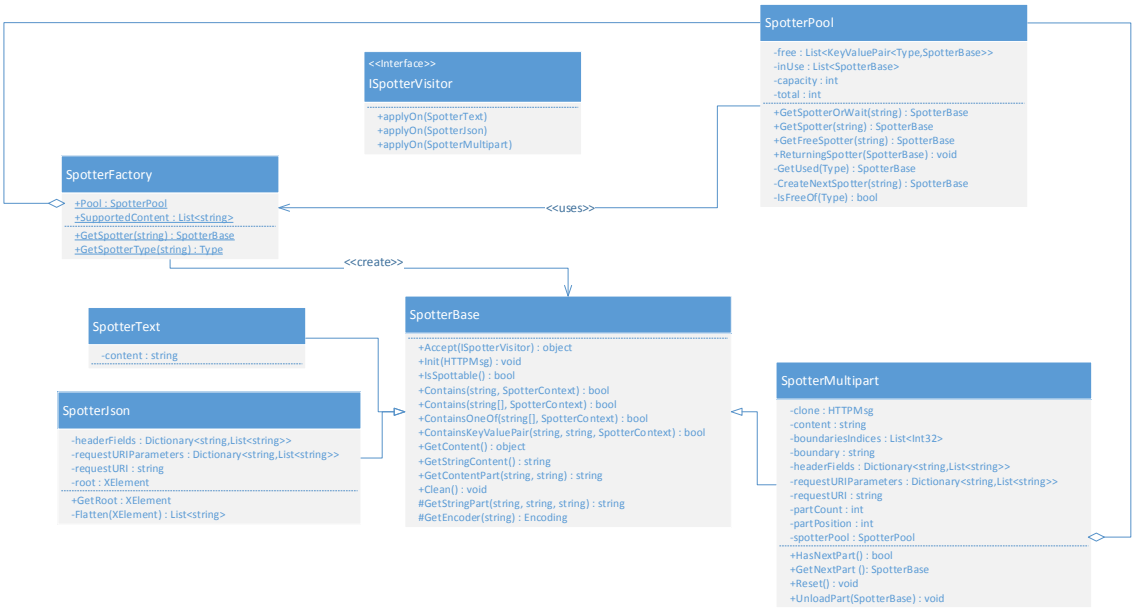
3.3.2 Pozorovatelia obsahu

Nevyhnutnou súčasťou webmailového analyzátoru je pozorovateľ obsahu, *spotter*. Pozorovateľ ma za úlohu prehliadať obsah a zistiť, či v ňom existujú určité chcené znaky. V extrémnom prípade sa môže pozorovateľ pokúsiť časť obsahu extrahovať.

Rovnako ako webmailový analyzátor by sa pozorovateľ mal vytvoriť pre každú HTTP správu. To by pri dlhej konverzácii mohlo mať nežiadúce účinky na pamäť. U pozorovateľov je ale potrebné aspoň čiastočné uchovanie stavu. Pri použití návrhového vzoru *jedináčik* by mohol byť problém pri viacerých vláknach. Preto bol zvolený návrhový vzor *fond* (viď Obrázok 3.5). S inicializáciou fondu bude možné nastaviť maximálny počet inštancií, ktoré sa môžu vytvoriť a zároveň sa budú už vytvorené inštancie opätovne používať, keď nebudú potrebné. Trieda `SpottersPool` poskytuje takéto rozhranie:

- metóda `GetSpotterOrWait(string)` - vráti správneho pozorovateľa, ak je nejaký voľný alebo ak kapacita fondu ešte nebola dosiahnutá, inak musí čakať, kým sa nejaký pozorovateľ uvoľní. Správneho pozorovateľa vyberie na základe parametru, ktorý by mal obsahovať hodnotu poľa `Content-Type` z HTTP správy,
- metóda `GetSpotter(string)` - podobne ako metóda `GetSpotterOrWait(string)`, ale ak sa dosiahla kapacita fondu a nie je žiaden voľný pozorovateľ, vyhodí výnimku,
- metóda `GetFreeSpotter(string)` - ak existuje voľný pozorovateľ chceného, druhu tak ho vráti, inak vráti null. Nepokúša sa vytvoriť nového pozorovateľa,
- metóda `ReturningSpotter(SpotterBase)` - vráti použitého pozorovateľa do fondu.

Cieľom je, aby každý analyzátor používal rovnaký/jeden fond. Najjednoduchšie riešenie bolo spraviť tento fond ako statický člen triedy `SpotterFactory`. Každému analyzátoru sa potom musí fond *spotter*-ov injektovať. To sa deje pri získavaní inštancie jedináčka analyzátoru metódou `GetInstance()`, ktorá prijíma parameter typu `SpotterPool`.



Obrázok 3.5: Diagram tried pozorovateľov využívajúcich *fond* a *továreň*.

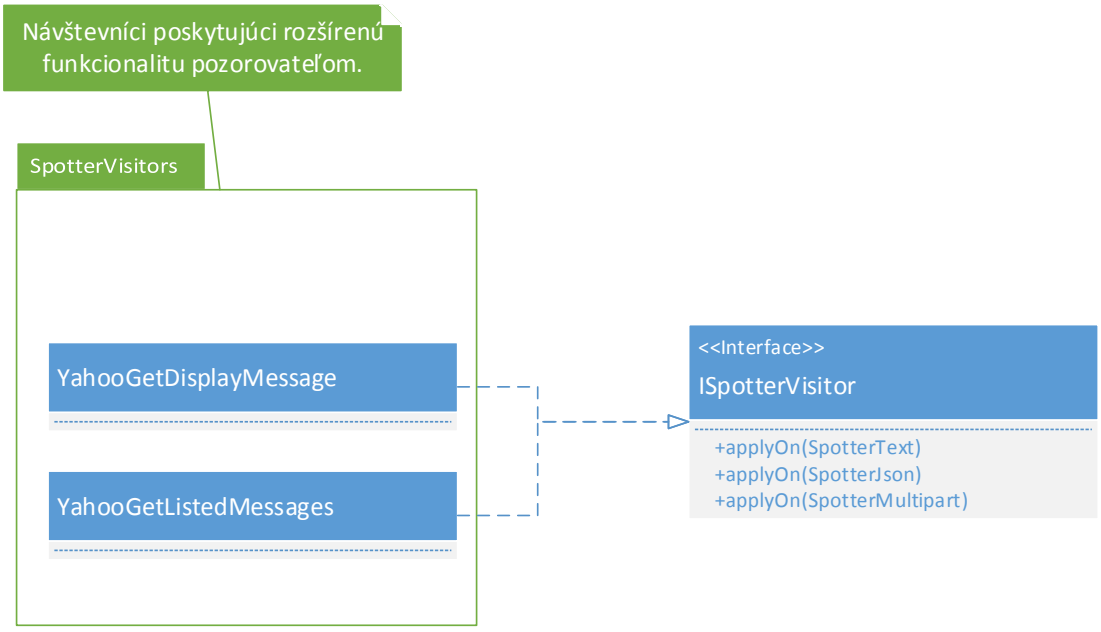
Pri spomínanej implementácii môže nastať jeden problém. Nastáva, keď počas celého života fondu sa nevytvorí pozorovateľ každého druhu a kapacita sa naplní. Môže prísť požiadavka na pozorovateľa, ktorý nebol vytvorený a program ostane čakať bez zjavnej chyby (keďže sa volá metóda `GetSpotterOrWait(string)`). Tento problém sa dá vyriešiť napríklad tak, že pri inicializácii fondu sa vytvorí z každého druhu aspoň jeden pozorovateľ dopredu.

Ako si je možné všimnúť na Obrázku 1.1, metóda `IsMsgWebmailEvent()` obsahuje návratový parameter pozorovateľa. Tento pozorovateľ je jeden z pozorovateľov z fondu. Keďže ho táto metóda vracia užívateľovi, nemôže ho zároveň vrátiť do fondu, pretože by hrozilo, že by sa použil znovu pre inú správu. Vrátenie pozorovateľa do fondu teda musí vyriešiť užívateľ.

Pozorovaný obsah môže byť pre viaceré správy zakódovaný v rovnakom formáte. Teda môže to byť JSON, či FastRPC, no štruktúra sa môže líšiť u každej aplikácie. Líšiť sa môže preto, lebo spomínané formáty reprezentujú iba generický spôsob zápisu dát. Tu nastáva problém so samotnou extrakciou relevantných častí obsahu. Pozorovateľ je generický modul nezávislý na aplikácii, ktorej obsah pozoruje. Implementovať doň funkcionalitu závislú od konkrétnej aplikácii by bolo nevhodné.

Táto situácia bola vyriešená návrhovým vzorom *návštevník*. Každý návštevník implementuje rozhranie `ISpotterVisitor` (viď Obrázok 3.6), ktoré definuje metódy `applyOn()` pre každý druh pozorovateľa, tzn. parametrom tejto metódy je pozorovateľ. Do bázovej triedy pozorovateľa sa pridá metóda `accept(ISpotterVisitor)` (viď Obrázok 3.5), ktorá slúži ako vstup pre návštevníkov. Každý pozorovateľ túto metódu implementuje rovnakým spôsobom a to zavolaním metódy `applyOn()` daného návštevníka. Takto bude možné jednoducho doimplementovať prípadne ďalšie extraktory obsahu pre aplikácie, ktoré to budú vyžadovať.

Z analýzy webmailovej komunikácie (Kapitola 1.1) vieme, že niektoré znaky webmailových udalostí sa dajú rozoznať najlepšie z HTTP hlavičky. Preto pozorovateľ musí vedieť



Obrázok 3.6: Diagram návštevníkov pozorovateľov.

pozorovať aj iné časti HTTP správy. To, na ktorú časť sa má pozorovateľ zamerať je možné odlíšiť pomocou príznaku `SpotterContext`. Konkrétny pozorovateľ potom musí dediť z abstraktnej triedy `SpotterBase`. Táto trieda obsahuje väčšinou iba abstraktné metódy až na dve:

- metóda `GetStringPart(string,string,string)` - je *protected* člen tejto triedy. Táto metóda získa podreťazec daného reťazca, ktorého hranice určujú iné podreťazce,
- metóda `GetEncoder(string)` - je ďalší *protected* člen a vracia inštanciu triedy `Encoding` na základe poľa `Content-Type` z HTTP hlavičky,
- metóda `Accept(ISpotterVisitor)` - je metóda, ktorou sa aplikuje návštevník na konkrétneho pozorovateľa,
- metóda `Init(HTTPMsg)` - inicializuje pozorovateľa danou HTTP správou,
- metóda `IsSpottable()` - vráti `true`, ak HTTP správa obsahuje pozorovateľnú časť,
- metóda `Contains()` - vráti `true`, ak HTTP správa obsahuje v danom kontexte daný reťazec/reťazce,
- metóda `ContainsOneOf(string[], SpotterContext)` - vráti `true`, ak HTTP správa obsahuje aspoň jeden zo zadaných reťazcov v danom kontexte,
- metóda `ContainsKeyValuePair(string,string, SpotterContext)` - vráti `true` ak HTTP správa obsahuje daný pár,
- metóda `GetContent()` - vráti inštanciu objektu reprezentujúcu obsah správy,

- metóda `GetStringContent()` - vráti reťazec reprezentujúci obsah správy,
- metóda `GetContentPart(string, string)` - vráti časť obsahu správy na základe dvoch vzorov, kde jeden reprezentuje kľúč a druhý hodnotu,
- metóda `Clean()` - vyčistí pozorovaeľa od aktuálne pozorovanej správy.

Konkrétny pozorovateľ potom môže implementovať aj ďalšie metódy, ktoré môžu byť využité pri aplikovaní návštevníka na konkrétneho pozorovateľa. Takto to je možné nájsť napríklad u pozorovateľa `SpotterMultipart`, ktorý okrem zdedených metód bázovej triedy implementuje ešte metódy:

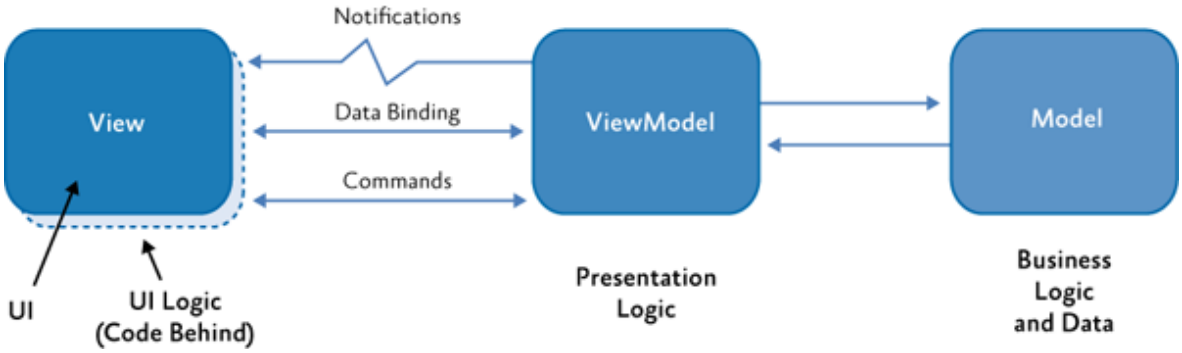
- `HasNextPart()` - vráti true ak obsah má ďalšiu časť,
- `GetNextPart()` - vráti pozorovateľa ďalšej časti,
- `UnloadPart(SpotterBase)` - vráti pozorovateľa do fondu
- `Reset()` - resetuje počítadlo častí.

3.4 Užívateľské rozhranie

Užívateľské rozhranie bolo implementované s využitím architektonického návrhového vzoru *Model/View/ViewModel* (MVVM) (viď Obrázok 3.7), ktorý vyžaduje aj samotný nástroj *Netfox.Detective*. Ako píše John Gossman, 2005 [1] vo svojom blogu, kde prvýkrát odhalil MVVM vzor, tento vzor je variácia vzoru *Model/View/Controler* (MVC), ktorý je prispôsobený moderným požiadavkám oddelenosti vývoju pohľadu a modelu, kedy pohľad vyvíja UI dizajnér, ktorý je viac umelec ako vývojár.

Model je v MVVM definovaný rovnako ako v MVC, teda dáta alebo logika programu, ktorá je kompletne oddelená od užívateľského rozhrania. Zabezpečuje riešenie domény problému.

Pohľad pozostáva z grafických elementov (tlačidiel, okienok a iných komponent GUI). Interaguje so vstupnými zariadeniami (čo je zodpovednosť controléru v MVC vzore).



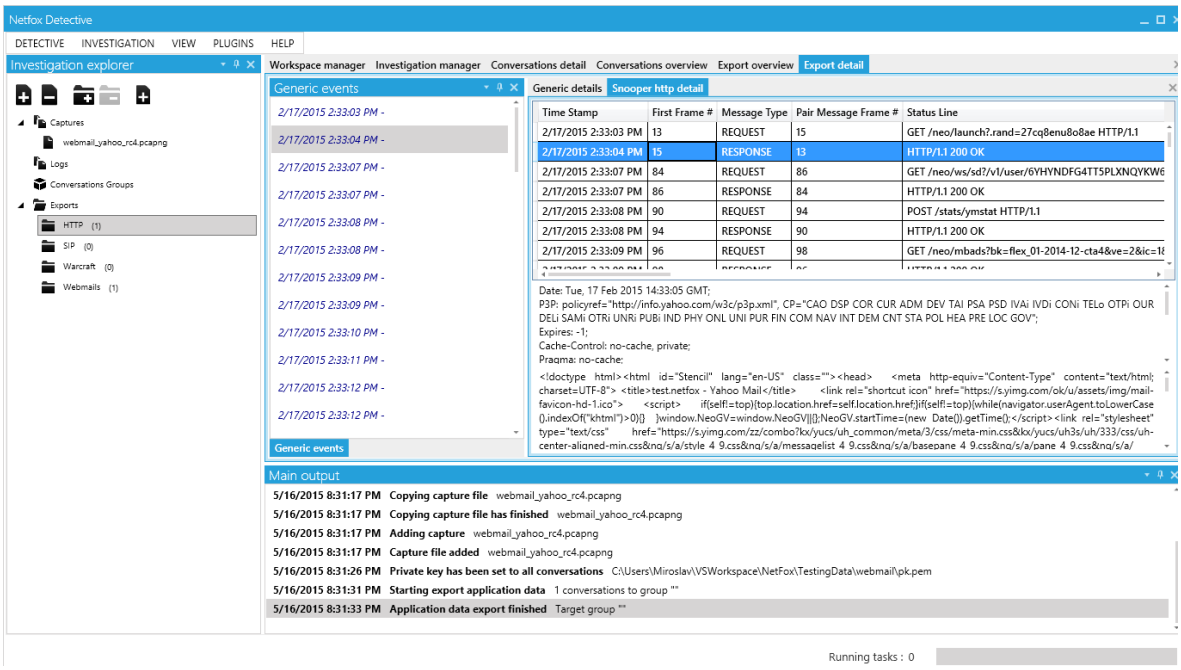
Obrázok 3.7: Interakcie v modeli MVVM. Zdroj: https://msdn.microsoft.com/

ViewModel je komponenta, ktorá odhaľuje dáta z *modelu* do *pohľadu*, prípadne môže udržiavať stav aplikácie. Môžeme sa naň pozerať ako na abstrakciu *pohľadu* alebo tiež špecializáciu *modelu*, ktorú môže *pohľad* použiť na *Data Binding*. V tejto úlohe *ViewModel*

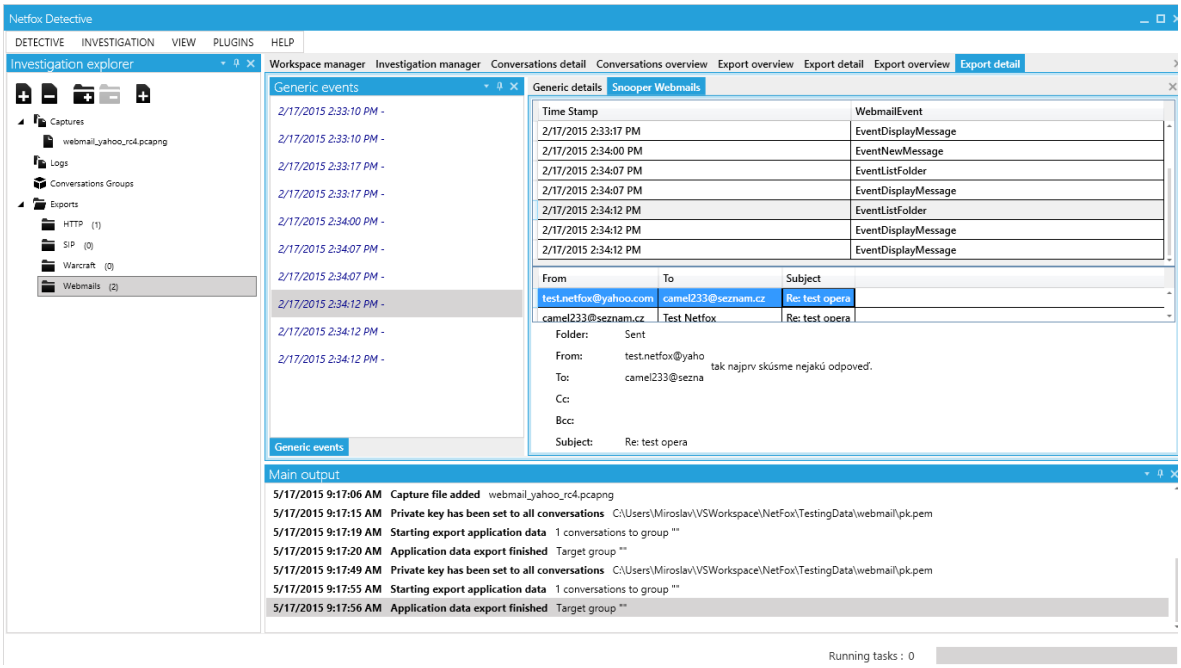
obsahuje transformátory dát, ktoré konvertujú typy *modelu* na typy *pohľadu* a obsahuje príkazy, ktoré môže pohľad použiť na interakciu s *modelom*.

Pre implementáciu užívateľského rozhrania bolo nutné vytvoriť rozhranie pre *pohľad*, aby ho bolo možné dynamicky načítať z *Netfox.Detective*-u. Okrem rozhrania musí pohľad dediť z triedy *DetectiveExportDetailPaneViewBase*. Komponenta *ViewModel* potom musí dediť z triedy *DetectiveExportDetailPaneViewModelBase*, ktorá poskytla konštruktor a prístup k exportovaným dátam.

Finálne implementované *pohľady* pre moduly `SnooperHTTP` a `SnooprtWebmails` je možné vidieť v akcii na Obrázkoch 3.8 a 3.9.



Obrázok 3.8: Finálna podoba rozhrania HTTP *Snooper*-a integrovaného v *Netfox.Detective*.



Obrázok 3.9: Finálna podoba rozhrania Webmailového *Snooper*-a integrovaného v *Netfox.Detective*.

Kapitola 4

Testovanie

Testovanie implementovaného riešenia prebehlo pomocou Unit Testov. Pre každý implementovaný modul bol vytvorený jeden *Test Suite*, ktorý obsahuje všetky implementované testy.

Priebeh SSL/TLS komunikácie sa dá rozdeliť na niekoľko testovateľných častí, ktoré preveria správne dešifrovanie SSL/TLS relácie:

- **Výmena kľúčov**. SSL/TLS protokol môže použiť na výmenu kľúčov rôzne algoritmy asymetrickej kryptografie. Pre každý implementovaný algoritmus je vytvorený osobitný test. V rámci testu sa triede implementujúcej daný algoritmus pošle správa *Client Key Exchange* a preverí sa správnosť získaného kľúča na základe informácií z Wireshark-u.

- **Generovanie kľúča symetrickej kryptografie**. Kľúč symetrickej kryptografie sa generuje v dvoch krokoch, v oboch sa ale používa rovnaká funkcia (PRF). Implementácia tejto funkcie sa mení podľa verzie protokolu.

- **Dešifrovanie správy**. Testovanie samotnej implementácie šifrovacieho algoritmu nemá zmysel, keďže ako implementácia týchto šifrovacích algoritmov bola využitá externá knižnica. Má ale zmysel otestovať, že sa implementovanými krokmi skutočne dostaneme k dešifrovanej správe. Na overenie správnosti implementácie triedy `PDUdecrypter` sa otestuje dešifrovanie jednej relácie SSL/TLS. Overí sa počet dešifrovaných správ a vybratá časť dešifrovaného prúdu dát. Tento postup sa použije pre všetky implementované symetrické šifry.

Vstupné a referenčné hodnoty testov boli získané z debugovacích výstupov Wireshark-u, konkrétne modulu *SSL dissector*. *SSL dissector* je súčasťou Wireshark-u a zabezpečuje práve dešifrovanie SSL/TLS relácií. Vo Wireshark-u nastavenia tohto modulu nájdeme pod nastaveniami protokolu SSL. Tam je možné pridať ako privátne kľúče serverov pod položku *RSA key list*, tak aj klientské kľúče uložené v súbore vo formáte *NSS Key Log*[1] pod položku *(Pre)-Master-Secret log filename*. Rovnako v tejto obrazovke je možné nastaviť, kam sa majú ukladať debugovacie výpisy. V debugovacích výpisoch je potom možné nájsť hodnoty *(Pre)-Master secret*, získané náhodné hodnoty zo správ *Client* a *Server Hello* či výstupy PRF.

Testovacie dáta (viď Príloha A) boli nazbierané využitím znalostí z Kapitoly 1.3.3.

Úspešné testy zobrazuje Tabuľka 4.1.

[1] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format

Popis testu
Test chovania PRF metódy v TLS v1.0 a TLS v1.1
Test chovania PRF metódy v TLS v1.2
Získanie hodnoty *pre-master secret* zo správy *Client Key Exchange* - RSA
Test dešifrovania - AES v CBC režime
Test dešifrovania - RC4

Tabuľka 4.1: Úspešné testy modulu PDUDecrypter

Neúspešné testy zobrazuje Tabuľka 4.2.

Neúspešne dopadlo dešifrovanie komunikácie šifrovanej blokovou šifrou v GCM režime. Dešifrovanie zlyhalo, pretože použitá knižnica v režime GCM podporuje MAC do dĺžky 128 bitov. V SSL/TLS sa ale používa minimálna dĺžka MAC 256 bitov. Na doimplementovanie tejto funkcionality bude potrebné využiť inú knižnicu alebo počkať na verziu knižnice BouncyCastle, v ktorej už podpora bude.

Popis testu	Dôvod zlyhania
Test dešifrovania - AES v režime GCM	Obmedzenie knižnice BouncyCastle (využiť inú knižnicu)

Tabuľka 4.2: Neúspešné testy modulu PDUDecrypter

SnooperHTTP Testovanie HTTP analyzátora prebehlo na základe počtu vyexportovaných HTTP správ zo zvolenej komunikácie. Reálny počet správ v danej komunikácii bol dopredu známy vďaka nástroju Wireshark. Okrem toho prebehlo aj výkonnostné testovanie. Testovala sa doba vykonávania modulu `SnooperHTTP` v závislosti na počte konverzácií v PCAP súbore. Za týmto účelom bol vytvorený jeden test, v ktorom sa obmieňali vstupné dáta. Graf 4.1 a Tabuľka 4.3 ukazujú výsledky tohto testovania. V teste je vidieť istý neprirodzený výkyv pri hodnote 1253 konverzácií, respektíve 5182 konverzácií. Toto mohlo byť spôsobené veľkým množstvom chybne vyparsovaných správ pri hodnote 5182 konverzácií, a to spôsobilo rýchlejšie ukončenie. Podobne nízke hodnoty v prvých dvoch meraniach mohli byť spôsobené menším počtom HTTP správ v súbore.

Parsovanie HTTP správy sa deje v jednom prechode čítania správy. Keď sa zanedbá to, že veľkosť každej HTTP správy môže byť rôzna, potom vyvodená časová zložitosť algoritmuje $O(n)$, kde n je počet HTTP správ v skúmanej konverzácii. Štruktúry, ktoré sa používajú v tomto algoritme reprezentujú práve HTTP správu, takže priestorová zložitosť algoritmu je $O(n)$, kde n je počet HTTP správ v konverzácii.

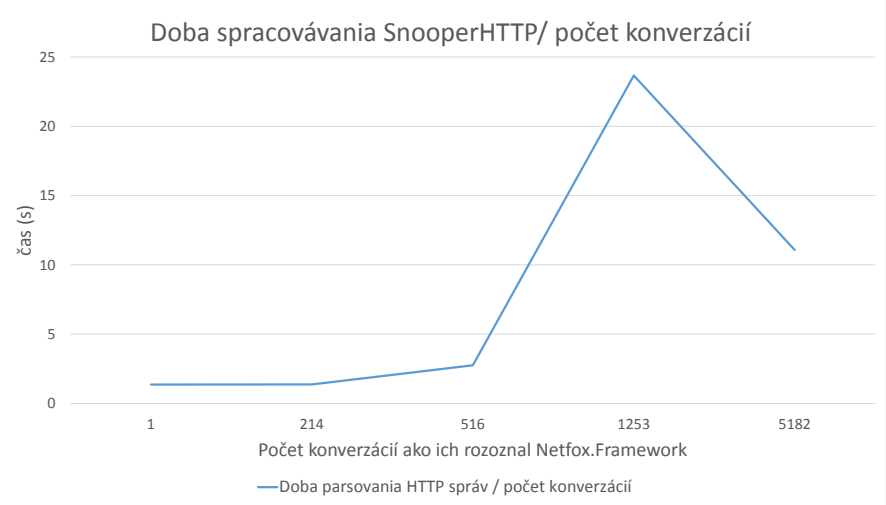
Z testovania je možné vyvodiť, že pri správnom vstupe algoritmus produkuje správny výstup. Pri nesprávnom alebo nepodporovanom vstupe, výstup bude prázdny alebo nekompletný.

SnooperWebmails Ako je zrejmé z Kapitoly 3.3, celá detekcia webmailových udalostí stojí na pozorovateľoch obsahu. Preto bol vytvorený test pre pozorovateľa obsahu každého typu. Každý test pozorovateľa je koncipovaný rovnako, iba s malými zmenami.

- Najprv sa musí pustiť `SnooperHTTP` nad vhodným súborom, ktorý nainicializuje štruktúry `HTTPMsg`,

Počet konverzácií	čas (s)
1	1,3660889
214	1,373759
516	2,7592433
1253	23,668021
5182	11,0780134

Tabuľka 4.3: Výkonnostný test modulu SnooperHTTP



Obrázok 4.1: Graf výkonnostného testu modulu SnooperHTTP.

- v ďašom kroku sa vyberie niektorá vhodná správa s dopredu známym obsahom, napríklad z pozorovania vo Wireshark-u,
- inicializuje sa ňou testovaný pozorovateľ,
- testujú sa pozorovacie úlohy s očakávaným výsledkom `true` nad rôznym kontextom,
- testujú sa pozorovacie úlohy s očakávaným výsledkom `false` nad rôznym kontextom.

Pre testovanie webmailového *snooper*-a boli vytvorené testy, ktoré overujú, že boli detekonované nejaké udalosti. Minimálny počet bol vyhodnotený podľa analýzy obsahu HTTP správ vo Wireshark-u. Detekcia bola testovaná nad štyrmi webmailovými aplikáciami a to Gmail, Microsoft Live, Yahoo Mail a Mail Seznam.cz. U ostatných webmailov je najpravdepodobnejšia detekcia udalosti typu *Nová správa*, keďže mnoho webmailových aplikácií posiela v obsahu bežné polia s podobným pomenovaním: *odosielateľ*, *príjemca*, *predmet správy*, *obsah správy*.

Pre testovanie všetkých implementovaných modulov boli použité testovacie dáta z Prílohy A, zložky TestingData/webmail. Dáta označené ako *whole_conversation* boli použité na výkonnostné testovanie modulu `SnooperHTTP`. Okrem testov v tejto zložke boli na výkonnostné testovanie použité aj dáta TestingData/http_caps/http_all.cap a TestingData/pcap_mix/isa-http.pcap.

4.1 Integračné testovanie

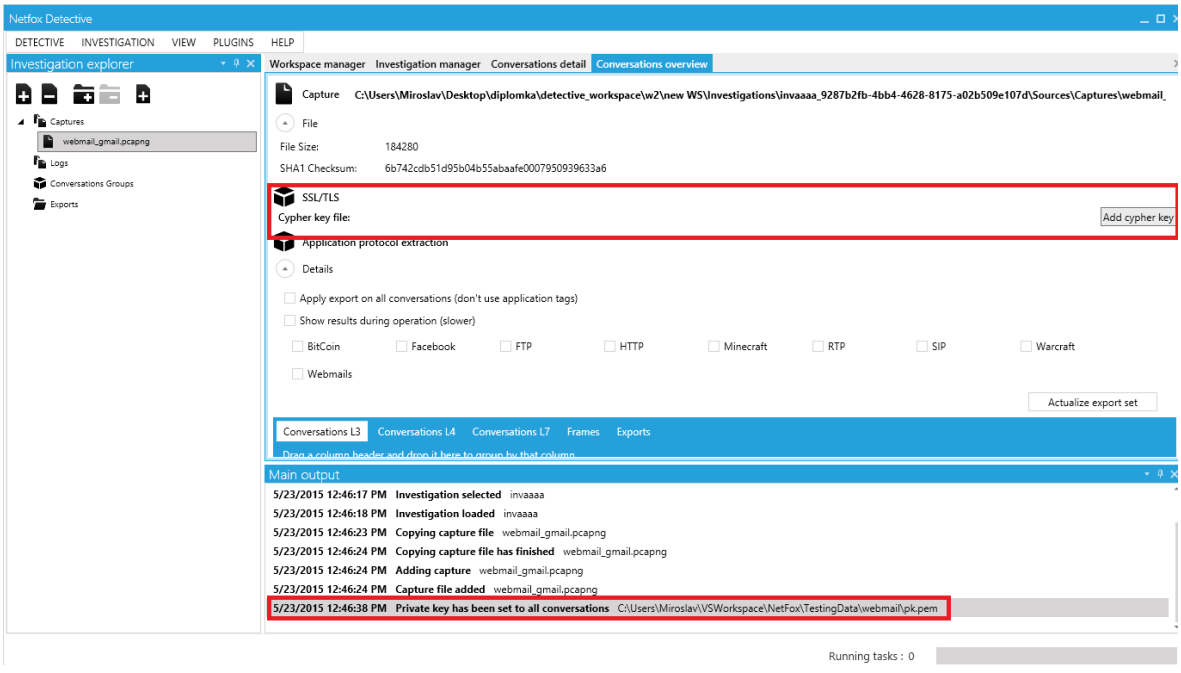
Po integrovaní všetkých implementovaných modulov do *Netfox.Framework*-u prebehlo testovanie integrovaných častí priamo v aplikácii *Netfox.Detective*.

Pri tomto testovaní sa použila rovnaká časť zachytenej komunikácie ako v unit testoch, plus boli otestované PCAP súbory obsahujúce kompletnú komunikáciu. Všetky tieto súbory obsahujú nejakú HTTP komunikáciu, ktorá je nešifrovaná, no väčšina relevantnej komunikácie, ktorá obsahuje webmailové udalosti je šifrovaná.

V prostredí *Netfox.Detective*-u je možné nastaviť privátny kľúč serveru pre daný PCAP súbor (viď Obrázok 4.2). Toto nastavenie sa aplikuje pre všetky nájdené konverzácie v analyzovanom súbore. Testovanie odhalilo problém, ktorý sa objaví, keď sú splnené nasledujúce podmienky:

- v PCAP súbore sa nájde viac konverzácií,
- PCAP súboru sa priradí súkromný kľúč.

V tomto prípade sa `SnooperHTTP` pokúsi na každú konverzáciu (s nastaveným privátnym kľúčom) použiť `PDUDecrypter`. To znamená, že s takýmto nastavením sa nedetekuje HTTP protokol v nešifrovanej podobe.



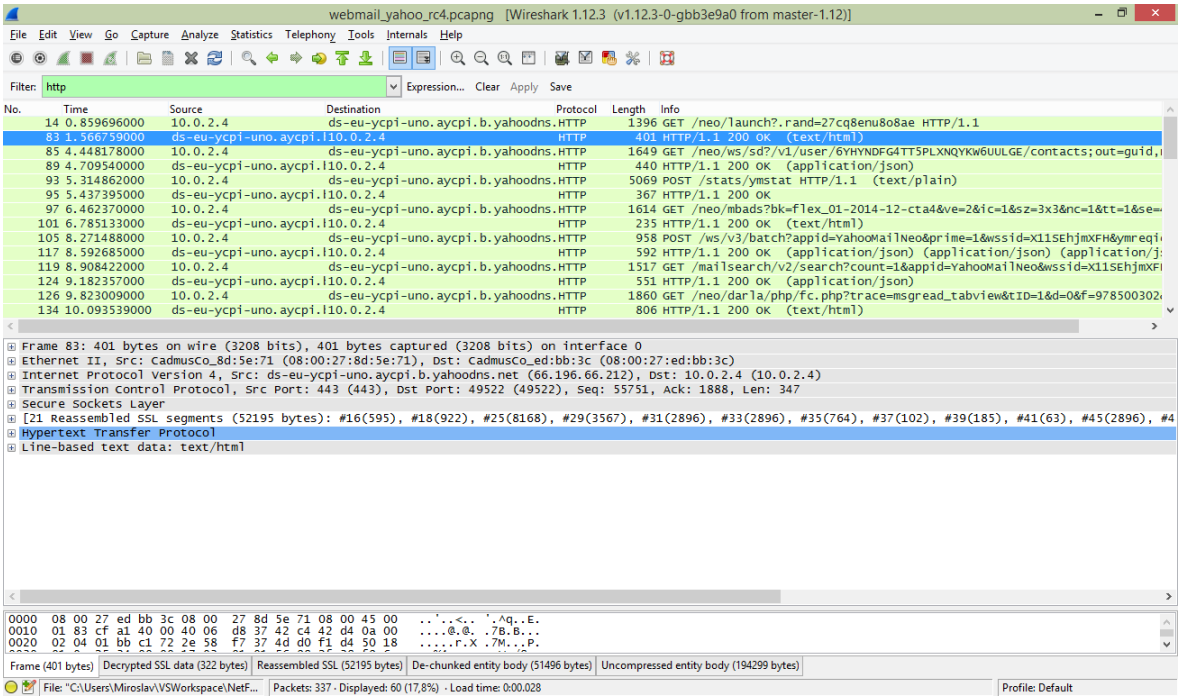
Obrázok 4.2: Vkladanie privátneho kľúča do *Netfox.Detective*.

Ďalší odhalený problém súvisí s tým, ako *Netfox.Framework* vidí a spravuje konverzácie. Opäť musia byť splnené vyššie uvedené podmienky. V tomto prípade sa neodhalia žiadne

šifrované HTTP udalosti. To môže byť spôsobené tým, že konverzácie neobsahujú kompletný *SSL Handshake*.

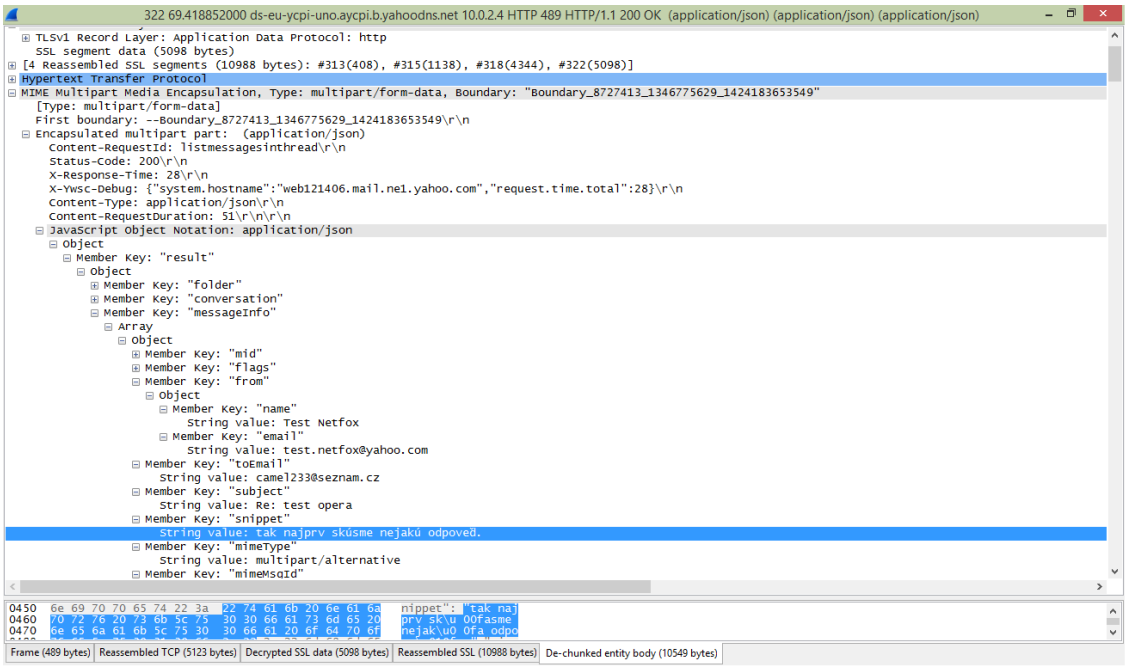
Konečným dôsledkom týchto problémov je potreba implementovať spôsob priradenia privátneho kľúča serveru na špecifickú konverzáciu. Dočasným riešením je vo Wireshark-u vybrať jeden TCP *stream* a tento uložiť. V *Netfox.framework*-u sa potom takto uložený jeden *stream* premietne presne do jednej konverzácie (ktorá už bude obsahovať kompletný *SSL Handshake*) a je ju možné analyzovať v prostredí *Netfox.Detective*.

Obrázok 4.3 ukazuje, ako vidí HTTP správy Wireshark. Tento náhľad je možné porovnať s náhľadom na Obrázku 3.8, kde je zobrazená rovnaká komunikácia v *Netfox.Detective*-ovi. Je možné si všimnúť, že niektoré správy majú iné číslo rámca. To je preto, lebo Wireshark na tomto mieste ukazuje posledný rámec TCP segmentu na rozdiel od pohľadu modulu `SnooperHTTP`, ktorý ukazuje prvý rámec TCP segmentu.



Obrázok 4.3: Ukážka zobrazenia HTTP správ v nástroji Wireshark.

Pohľad modulu `SnooperWebmails` nie je možné porovnať s nástrojom Wireshark, pretože ten nerozlišuje webmailové udalosti. Integračné testovanie v tomto prípade prebehlo preskúmaním analyzovaného TCP *stream*-u vo Wireshark-u a priradením detekovaných udalostí k HTTP správam. Napríklad vybraná udalosť na Obrázku 3.9 korešponduje s udalosťou zobrazenou na Obrázku 4.4.



Obrázok 4.4: Ukážka analyzovanej HTTP správy obsahujúcej webmailovú udalosť.

Záver

Na základe analýzy webmailovej komunikácie bolo zistené, že mnoho webmailov komunikuje šifrovane pomocou SSL/TLS. Taktiež sa našli podobné znaky významných webmailových udalostí, ktoré sa neskôr použili na ich detekciu v nástroji *Netfox.Framework*. Kým sa bolo možné dostať k samotnej implementácii modulu na detekovanie a extrakciu webmailových udalostí, bolo nutné sa vysporiadať s dešifrovaním.

Na základe analýzy protokolu SSL/TLS (viď Kapitola 1.2) sa implementoval modul `PDUDecrypter`, ktorý zabezpečuje dešifrovanie skúmanej komunikácie. V aktuálnej verzii podporuje jeden algoritmus na výmenu kľúčov (RSA) a dva algoritmy použité na šifrovanie dát AES a RC4. Šifra AES je z triedy blokových šifier a na šifrovanie prúdu dát používa jeden z módov blokových šifier spomínaných v Kapitle 1.2. V module `PDUDecrypter` je aktuálne implementovaná podpora pre mód CBC. `PDUDecrypter` si poradí s verziami TLS 1.0 až TLS 1.2.

Dešifrovať pomocou nástroja *Netfox* s využitím modulu `PDUDecrypter` je možné, ak máme zachytenú kompletnú komunikáciu a prístup k privátnemu kľúču serveru. Privátny kľúč bežne nie je dostupný, ale pomocou niektorého penetračného nástroja či proxy spomínaného v Kapitole 1.3.3 je možné použitím sfalšovaného certifikátu nadviazať novú SSL/TLS reláciu smerom ku klientovi.

Ďalším krokom k detekcii webmailových udalostí je export HTTP správ. Toto zabezpečuje implementovaný modul `SnooperHTTP`. Jeho modelom je HTTP správa. Detekcia sa robí na základe stavového riadku HTTP protokolu. Po úspešnej extrakcii HTTP správ je možné z týchto správ detekovať webmailové udalosti.

V tejto práci opisovaný modul `SnooperWebmails` je postavený na detekcii na základe spoločných znakov väčšiny webmailových aplikácií. Spoločné znaky boli vybrané na základe analýzy z Kapitoly 1.1 a sú prispôsobené veľkým webmailovým aplikáciám ako sú Gmail, Yahoo, Seznam, Centrum... To vo výsledku znamená, že udalosti týchto webmailových aplikácií budú s určitosťou detekované, ak sa vyskytujú v danej komunikácii.

Rozšíriteľnosť implementovaných modulov je na dobrej úrovni vďaka implementácii oproti rozhraniam. Týmto spôsobom boli implementované triedy poskytujúce algoritmy na výmenu kľúčov a šifrovanie dát pre modul `PDUDecrypter`. Ďalšia práca na module `PDUDecrypter` je rozšíriť ho o verziu protokolu SSL 3.0, implementácia podpory kompresných metód či implementácia autentizačnej časti *Handshake*-u.

Rozšíriteľnosť modulu `SnooperHTTP` spočíva v implementácii ďalších kompresných metód obsahu HTTP správy (aktuálne je podporovaná metóda *gzip* a *Deflate*) či implementovaní detekcie novej verzie protokolu HTTP 2.0.

Modul `SnooperWebmails` je možné rozšíriť o znaky udalostí typu *zobrazenie priečinku* a *zobrazenie správy* pridaním daných znakov do triedy `WebmailAnalyzerBase`. V tejto triede existujú slovníkové štruktúry popisujúce práve spomínané udalosti. Ďalšou možnosťou rozšírenia je implementácia špecifického analyzátoru nejakej webmailovej aplikácie, ktorej uda-

losti nie je možné detekovať jednoduchým pridaním znakov do slovníku. Inšpiráciu špecifického analyzátoru je možné hľadať v triede `MicrosoftWebmailAnalyzer`. V neposlednom rade je možné taktiež rozšíriť rodinu pozorovateľov. Pozorovatelia sú určení k pozorovaniu obsahu HTTP správ a preto celý balíček pozorovateľov je implementovaný so zreteľom na prenositeľnosť celého balíčku. Pozorovatelia sú závislí na štruktúre `HTTPMsg` implementovanej v module `SnooperHTTP`. Preto je otázka, či by nebolo vhodnejšie ich umiestniť do tohto modulu. Ideálne umiestnenie by bolo v nejakej medzivrstve medzi modulom `SnooperHTTP` a ostatnými modulmi typu *Snooper*, ktoré využívajú `SnooperHTTP`.

Kapitola 4 na záver ukazuje úspešnosť jednotlivých implementovaných modulov v práci, pre ktorú boli navrhnuté.

Časť tejto práce bola prijatá na konferencii Excel@FIT 2015, v ktorej získala piate miesto v kategorii *Inovačný potenciál* a dve ceny sponzorov.

Literatúra

[1] Introduction to Model/View/ViewModel pattern for building WPF apps. http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx, 2005 cit. 2015-05-17.

[2] The Legion of the Bouncy Castle. https://www.bouncycastle.org/, 2013 cit. 2015-03-23.

[3] Davidoff, S.; Ham, J.: *Network Forensics*. Prentice Hall, 2012, ISBN 0-13-256471-8.

[4] Dierks, T.; Allen, C.: The TLS Protocol Version 1.0. RFC 2246, RFC Editor, January 1999.
URL http://www.rfc-editor.org/rfc/rfc2246.txt

[5] Dierks, T.; Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, RFC Editor, April 2006.
URL http://www.rfc-editor.org/rfc/rfc4346.txt

[6] Dierks, T.; Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, RFC Editor, August 2008.
URL http://www.rfc-editor.org/rfc/rfc5246.txt

[7] Dworkin, M. J.: SP 800-38A 2001 Edition. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. Technická zpráva, Gaithersburg, MD, United States, 2001.

[8] Dworkin, M. J.: SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Technická zpráva, Gaithersburg, MD, United States, 2007.

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

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

[11] Hollenbeck, S.: Transport Layer Security Protocol Compression Methods. RFC 3749, RFC Editor, May 2004.
URL http://www.rfc-editor.org/rfc/rfc3749.txt

[12] McGrew, D.: An Interface and Algorithms for Authenticated Encryption. RFC 5116, RFC Editor, January 2008.
URL http://www.rfc-editor.org/rfc/rfc5116.txt

[13] Pluskal, J.: *Framework for Captured Network Communication*. diplomová práce, FIT VUT v Brně, Brno, 2014.

[14] Rescorla, E.: Diffie-Hellman Key Agreement Method. RFC 2631, RFC Editor, June 1991.
URL http://www.rfc-editor.org/rfc/rfc2631.txt

[15] Rescorla, E.; Modadugu, N.: Datagram Transport Layer Security Version 1.2. RFC 6347, RFC Editor, January 2012.
URL http://www.rfc-editor.org/rfc/rfc6347.txt

[16] Schatzmann, D.; Mühlbauer, W.; Spyropoulos, T.; aj.: Digging into HTTPS: Flow-based Classification of Webmail Traffic. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, New York, NY, USA: ACM, 2010, ISBN 978-1-4503-0483-2, s. 322–327, doi:10.1145/1879141.1879184.
URL http://doi.acm.org/10.1145/1879141.1879184

[17] Stallings, W.: *Cryptography and Network Security*. Prentice Hall, 1998, ISBN 0-13-869017-0.

[18] Whiting, D.; Housley, R.; Ferguson, N.: Counter with CBC-MAC (CCM). RFC 3610, RFC Editor, September 2003.
URL http://www.rfc-editor.org/rfc/rfc3610.txt

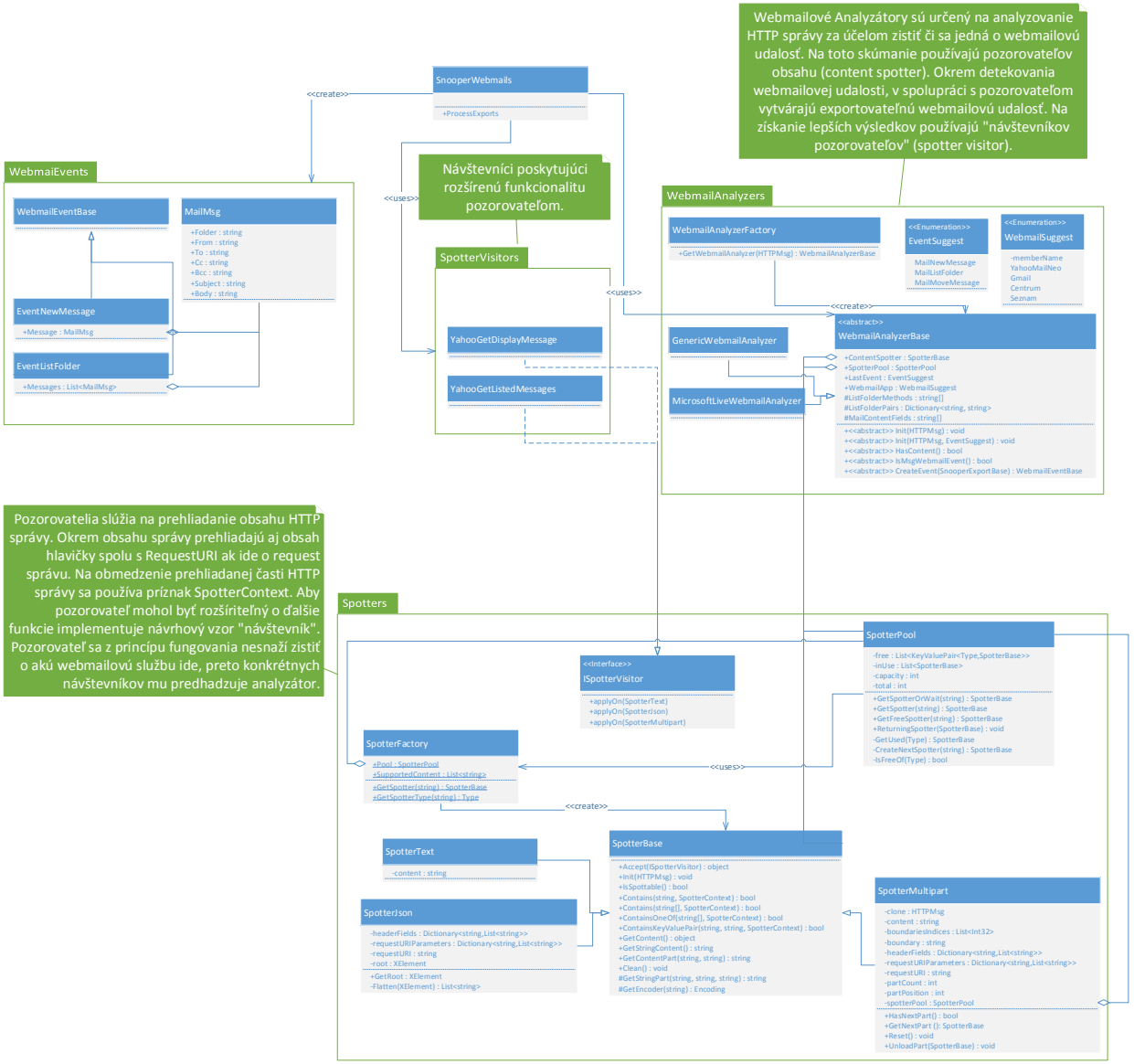
Příloha A

Obsah CD

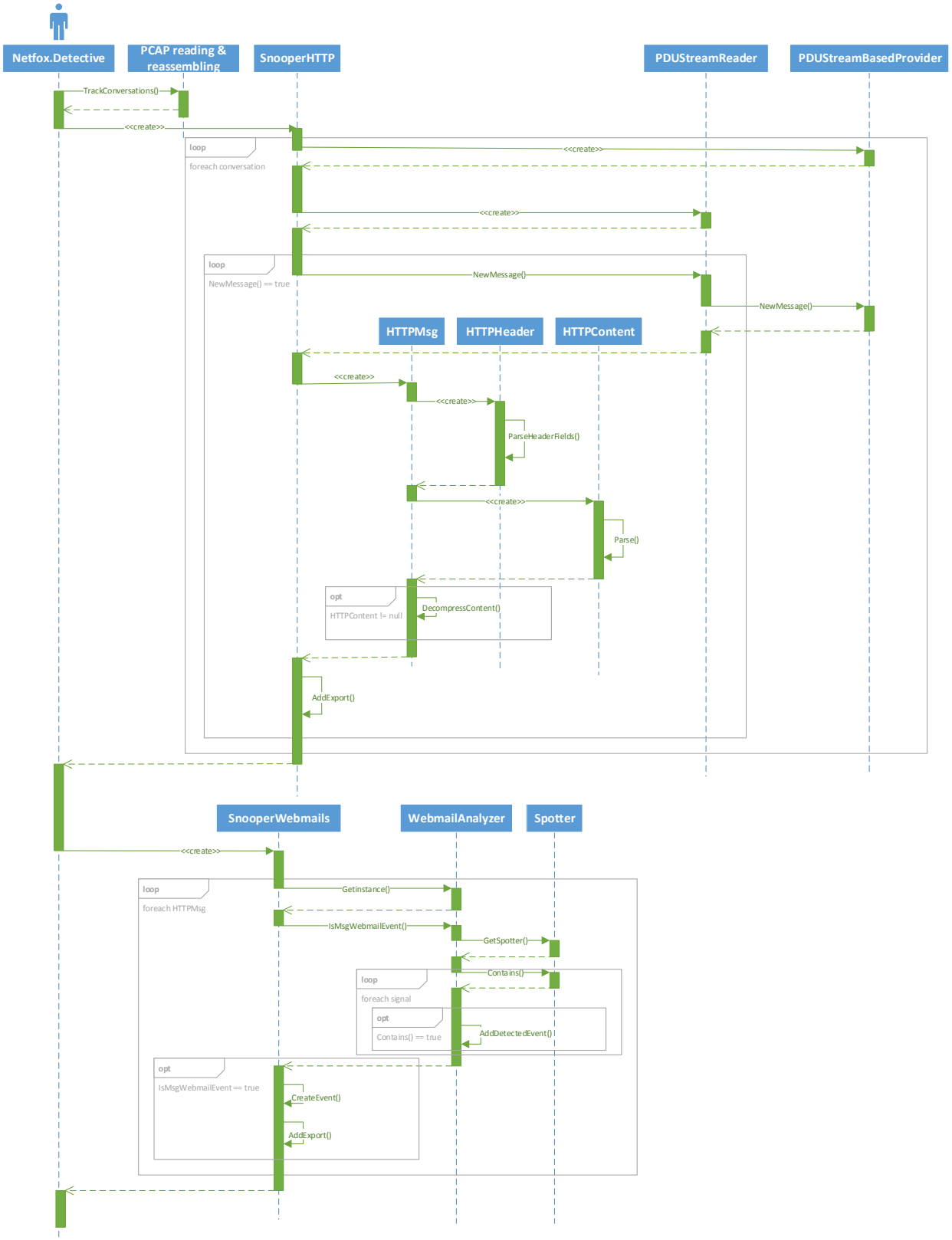
Netfox
+—.nuget
+—Detective
+—Framework
— +—...
— +—Netfox.Framework
— — +—...
— — +—**PDUProviders** - tento balíček obsahuje súčasti modulu PDUDecrypter
— +—**SnooperHTTP**
— — +—Interfaces
— — +—Models
— — +—Properties
— — +—View
— — +—ViewModel
— — +—ViewModels
— +—**SnooperWebmails**
— — +—Interfaces
— — +—Models
— — — +—Analyzers
— — — +—Spotters
— — — +—SpotterVisitors
— — — +—WebmailEvents
— — +—Properties
— — +—ViewModels
— — +—Views
+—...
TestingData
+—**webmail** - sada testovacích dát
AnalyzeData - dáta, ktoré slúžili na analýzu webmailových aplikácií
DP-text - zdrojové súbory textu diplomovej práce

Příloha B

UML Diagramy



Obrázok B.1: Diagram tried modulu SnooperWebmails.



Obrázok B.2: Sekvenčný diagram Netfoxu v spolupráci so SnooperHTTP a SnooperWebmails.