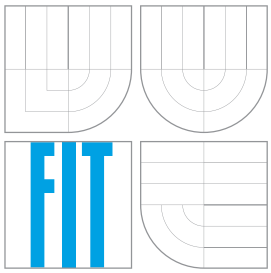
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



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

REKONSTRUKCE ZACHYCENÉ KOMUNIKACE NA PLATFORMĚ IOS

RECONSTRUCTION OF CAPTURED COMMUNICATION ON IOS PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VILIAM LETAVAY

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN PLUSKAL

BRNO 2016

Zadání bakalářské práce/18557/2015/xletav00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Letavay Viliam**
Obor: Informační technologie
Téma: **Rekonstrukce zachycené komunikace na platformě iOS**
Reconstruction of Captured Communication on iOS Platform
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se s mobilní platformou iOS a identifikujte zájmové aplikace a způsoby komunikace z hlediska dig. forenzní analýzy.
2. Proveďte hloubkovou analýzu struktury a komunikačních protokolů vybraných aplikací. Identifikujte události, které jsou zajímavé z pohledu Forenzní analýzy.
3. Rozšiřte aplikaci Netfox Detective o podporu pro rekonstrukci a vizualizaci vybraných aplikací.
4. Porovnejte výsledky rekonstrukce získané z Vaší implementace s ruční analýzou.
5. Implementované řešení řádně otestujte pomocí Unit testů.

Literatura:

- Callegati, F., Cerroni, W. & Ramilli, M., Man-in-the-middle attack to the HTTPS protocol. IEEE Security and Privacy, 7(1), s. 78-81. 2009.
- Mulazzani, M., Huber, M. & Weippl, E., Social Network Forensics: Tapping the Data Pool of Social Networks. In Eighth Annual IFIP WG 11.9 International Conference on Digital Forensics. 2012.

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: **Pluskal Jan, 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 dnešnej dobe nám naše mobilné zariadenia spolu so sociálnymi sieťami umožňujú byť kedykoľvek v spojení s našimi priateľmi a rodinou. Zároveň avšak môžu byť využité aj v spojení s kriminálnou aktivitou. Táto práca sa preto zaoberá rekonštrukciou komunikácie iOS aplikácií Google Hangouts, Twitter a Facebook Messenger pre potreby forenznej analýzy, pričom najväčší dôraz sa kládol na schopnosť obnovy ich chatovej komunikácie. Ďalej taktiež popisuje implementáciu rozšírení nástroja Netfox Detective umožňujúcich rekonštrukciu tejto komunikácie.

Abstract

These days our mobile devices along with social networks are giving us an opportunity to be in touch with our friends and family at any time. Howerver they could be also used for criminal activity. This thesis therefore deals with reconstruction of communication of the iOS applications Google Hangouts, Twitter and Facebook Messenger for use of forensics analysis with the main focus on an abillity to recover their chat communication. Later it also deals with the implementation of Netfox Detective tool extensions, allowing the reconstruction of this communication.

Kľúčové slová

Google Hangouts, Twitter, Facebook Messenger, iOS, dešifrovanie SSL, SPDY, MQTT, Netfox Detective

Keywords

Google Hangouts, Twitter, Facebook Messenger, iOS, SSL decryption, SPDY, MQTT, Netfox Detective

Citácia

LETAVAY, Viliam. *Rekonstrukce zachycené komunikace na platformě iOS*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Pluskal Jan.

Rekonstrukce zachycené komunikace na platformě iOS

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Jana Pluskala. Uviedol som všetky literárne pramene a publikácie z ktorých som čerpal.

.....
Viliam Letavay
19. mája 2016

Poďakovanie

Rád by som poďakoval vedúcemu bakalárskej práce, pánovi Ing. Janovi Pluskalovi, za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Obsah

Kapitola 1

Úvod

Mobilné zariadenia sa v posledných rokoch stali súčasťou našich životov. Rozšírenie dostupnosti mobilného internetového pripojenia nám zároveň pomocou nich umožnilo nepretržite a prakticky kdekoľvek využívať služby, ktoré nám internet ponúka. Jednou z týchto služieb sú rôzne sociálne siete, vďaka ktorým môžme byť neustále v kontakte s našimi priateľmi a rodinou. Avšak popri bežnej komunikácii je ich možné využiť aj na výmenu informácii spojených s kriminálnou činnosťou. Existuje preto požiadavka, aby bolo možné takúto komunikáciu v rámci zákonných odposluchov zrekonštruovať.

Cieľom tejto práce je analýza komunikácie vybraných iOS aplikácií z pohľadu forenznej analýzy a jej následná rekonštrukcia v nástroji Netfox Detective. Práca je rozdelená na päť kapitol, pričom prvá je tento úvod.

Druhá kapitola sa zaoberá stručným popisom platformy iOS a výberom aplikácii, ktorých komunikácia by mohla obsahovať zaujímavé informácie z pohľadu forenznej analýzy. Medzi takéto informácie patrí napríklad chatová komunikácia. Vybrané aplikácie preto predstavovali Google Hangouts, Twitter a Facebook Messenger.

Tretia kapitola analyzuje komunikáciu vybraných aplikácií. Pre každú z nich popisuje akým spôsobom bolo možné dešifrovať ich komunikáciu zabezpečenú protokolom SSL. Následne popisuje aké aplikačné protokoly využívajú na prenos informácií, po čom sa venuje analýze konkrétnych zaujímavých správ, ktoré sú nimi zasielané. Popisuje taktiež protokoly SPDY využitý aplikáciou Twitter a MQTT využitý aplikáciou Facebook Messenger.

Štvrtá kapitola popisuje nástroj Netfox Detective a implementáciu jeho rozšírení umožňujúcich rekonštrukciu komunikácie vybraných aplikácií na základe poznatkov z tretej kapitoly. Po predstavení tohto nástroja, popíše nutné úpravy ktoré v ňom museli prebehnúť, aby bol schopný dešifrovať zachytenú komunikáciu vybraných iOS aplikácii. Následne popíše implementáciu tzv. snooperov ktoré umožňujú rekonštrukciu tejto komunikácie. Posledná časť tejto kapitoly sa venuje testovaniu týchto snooperov.

Piata kapitola zhrnie dosiahnuté výsledky a popíše možné pokračovanie práce v budúcnosti.

Kapitola 2

Popis platformy iOS

iOS je mobilný operačný systém vyvíjaný spoločnosťou Apple určený výhradne pre jej mobilné zariadenia (v súčasnej dobe iPhone, iPod touch a iPad). Pre verejnosť bol prvýkrát dostupný v roku 2007[1], odvtedy jeho vývoj neustále pokračuje. Dnes ide po systéme Android o druhý najrozšírenejší mobilný operačný systém[2].

2.1 Aplikácie zaujímavé z pohľadu forenznej analýzy

Medzi ciele sieťovej forenznej analýzy patrí rekonštrukcia zachytených dát. Táto práca je zameraná na dáta pochádzajúce z mobilných aplikácií a preto prvým krokom je výber aplikácií potencionálne nesúcich informácie, ktoré by mohli byť využité pri dokazovaní kriminálnej činnosti. Hlavnými kandidátmi sú preto mobilní klienti rôznych sociálnych sietí umožňujúci komunikáciu v reálnom čase. Tento výber je zároveň podporený súčasným trendom, kedy užívatelia daných služieb na prístup k nim využívajú čoraz častejšie mobilné zariadenia s ich mobilnými verziami, miesto pôvodných, desktopových[3]. To z pohľadu mobilnej komunikácie zvyšuje šancu na ich využitie pri kriminálnej činnosti.

Z pohľadu forenznej analýzy je zaujímavá taká komunikácia, ktorá nesie informácie o tom, ako a kedy boli dané aplikácie využívane. Pri vybraných aplikáciach, teda pri mobilných klientoch rôznych sociálnych sietí preto pôjde hlavne o sledovanie chatovej komunikácie. Ďalšími užitočnými informáciami, ktoré môžu byť aplikáciami zasielané je informovanie o ich stave, čo umožňuje daným službám oznamovať ktorí ich užívatelia sú online. Z týchto informácií by malo byť následne možné povedať kedy mal užívateľ spustenú danú aplikáciu aj bez toho, aby musel pomocou nej s niekým aktívne komunikovať.

Google Hangouts

Je komunikačná platforma vyvíjaná spoločnosťou Google, určená pre sociálnu sieť Google+. Umožňuje zasielanie správ, zdieľanie fotiek a uskutočňovanie hlasových a video hovorov. Z rekonštrukcie komunikácie by malo byť možné dozvedieť sa s kým a o čom komunikoval

[1] "History of iPhone: Apple reinvents the phone — iMore", 2015, http://imore.com/history-iphone-original.

[2] "Smartphone OS Market Share, 2015 Q2 — IDC", 2015, http://idc.com/prodserv/smartphone-os-market-share.jsp.

[3] "More Than Half A Billion People Access Facebook Solely From Mobile — TechCrunch", 2015, http://techcrunch.com/2015/01/28/facebook-mobile-only-2/.

daný užívateľ. Aplikácia tiež umožňuje sledovanie stavu užívateľov (či sú v danom momente pripojení), preto by malo byť možné extrahovať aj tieto informácie.

Twitter

Ide o mobilného klienta pre sociálnu sieť Twitter. Poskytuje rovnakú funkcionalitu ako webová verzia služby, ako napríklad sledovať tzv. *tweety* ostatných užívateľov, vytvárať nové alebo komunikovať pomocou súkromných správ. Rekonštrukcia komunikácie by mala odhaliť informácie ktoré nie sú verejne dostupné, ako napríklad obsah súkromných správ alebo aký obsah užívateľ na tejto sociálnej sieti vyhľadáva.

Facebook Messenger

Poskytuje možnosť komunikácie s užívateľmi v sociálnej sieti Facebook. Hlavnou funkciou aplikácie je komunikovanie s ostatnými užívateľmi pomocou chatových správ, zdieľanie fotiek a vytváranie hlasových hovorov. Rovnako ako pri aplikácií Google Hangouts, rekonštrukcia komunikácie by mala odhaliť obsahy a adresátov zasielaných správ spolu s informáciami o aktivite samotnej aplikácie.

Kapitola 3

Analýza aplikačných protokolov

Rekonštrukcia vybraných informácií zo zachytenej komunikácie vyžaduje najprv porozumieť štruktúre a významu jej obsahu. Preto sa táto kapitola bude venovať analýzou komunikačných protokolov využitých vybranými aplikáciami.

Dnešné komunikačné aplikácie však pre zvýšenie bezpečnosti svojich užívateľov dbajú na to, aby nimi prenášané informácie boli určitou formou zašifrované a teda nečitateľné pre potencionálnych útočníkov, ktorí sa k nim dostanú. Po predbežnej analýze komunikácie daných aplikácií nástrojom Wireshark[1] bolo zjavné, že všetky tri vybrané aplikácie využívajú pri svojej komunikácii protokol SSL/TLS. Z tohto dôvodu bude nasledujúca kapitola venovaná protokolu SSL/TLS, možnostiam jeho dešifrovania a následne analýze daných aplikačných protokolov vybraných aplikácií.

Pri analýze bol použitý iPhone 4 s operačným systémom iOS 7.1.2. Ten bol tzv. jailbreaknutý, čo umožnilo hlbší prístup do operačného systému a schopnosť inštalovať a spúšťať aplikácie nepochádzajúce z oficiálneho Apple App Storu. Vďaka tomu bolo možné využiť nástroje, ktoré boli použité pri analýze samotných aplikácií a ich komunikácie.

3.1 Protokol SSL/TLS a možnosti jeho dešifrovania

Protokol Secure Socket Layer (SSL) a jeho nástupca Transport Layer Security (TLS)[2] sú kryptografické protokoly umožňujúce bezpečnú komunikáciu medzi dvoma zariadeniami. V súčasnej dobe je dešifrovanie SSL komunikácie možné jedine pomocou znalosti privátneho kľúča použitého pri komunikácii[4]. Pokiaľ tento kľúč nie je k dispozícii, môžme uskutočniť tzv. Man In The Middle (MITM) útok s využitím proxy SSL servera, pri ktorom je dátový tok medzi klientom a serverom presmerovaný cez prostredníka, ktorý zachytí novo nadväzujúce sa SSL spojenie, tým prevezme rolu servera s využitím vlastného privátneho kľúča (pričom klientovi podvrhne svoj falošný certifikát) a nakoniec vytvorí nové spojenie s pôvodným serverom.

Keďže sme nemali prístup k privátnym kľúčom vybraných služieb, využili sme možnosť s proxy SSL serverom[3]. Jedným z nástrojov, ktoré toto umožňujú, je SSLsplit[4]. Po spustení a presmerovaní dátového toku, je tento nástroj schopný pomocou svojho privátneho kľúča a falošného certifikátu nadviazať jedno SSL spojenie s klientom a druhé s pôvodným SSL

[1] Wireshark, http://wireshark.org.

[2] Názvom SSL bude ďalej označovaný aj protokol TLS. V kontexte tejto práce niesu medzi nimi rozdiely.

[3] Pri rekonštrukcií reálnych dát sa predpokladá znalosť privátnych kľúčov. Spôsob ich získania nieje predmetom tejto práce.

[4] Daniel Roethlisberger, "SSLsplit", 2015, http://roe.ch/SSLsplit.

serverom. Následne môže byť komunikácia medzi klientom a proxy SSL serverom zachytená a dešifrovaná pomocou už známeho privátneho kľúča. Na analýzu tejto komunikácie bol ďalej použitý Wireshark, ktorý ju po dodaní privátneho kľúča dokáže dešifrovať[5].

Avšak jednou z vlastností protokolu SSL je schopnosť autentizácie servera voči klientovi, prípadne klienta voči serveru, pomocou digitálnych podpisov certifikátov[5]. Bez znalosti originálneho privátneho kľúča nebude SSL proxy server schopný predložiť klientovi validný SSL certifikát, ktorému by klient dôveroval. To za normálnych okolností spôsobí ukončenie spojenia zo strany klienta (v prípade desktopových internetových prehliadačov je užívateľ informovaný o neplatnom certifikáte a je mu ponúknutá možnosť opustiť požadovanú stránku alebo pokračovať v potencionálne nebezpečnom spojení). Vybrané iOS aplikácie pri obdržaní nedôveryhodného certifikátu ukončia SSL spojenie, načo sa ho opätovne snažia znova nadviazať. Týmto sa ocitnú v cykle, v ktorom nie sú použiteľné.

Jedno z možných riešení je pridať daný falošný certifikát do zoznamu dôveryhodných koreňových certifikátov[6]. Avšak po pridaní a otestovaní neprejavila ani jedna z vybraných aplikácií zmenu v správaní. Z toho sa dalo vyvodiť, že využívajú tzv. certificate pinning[7]. Pri jeho využití aplikácia obchádza tzv. reťaz dôvery tak, že sama obsahuje lokálnu kópiu certifikátu použitého serverom, ktorej bude dôverovať. Namiesto certifikátu je taktiež možné uchovávať len jeho verejný kľúč alebo jeho hash.

3.2 Analýza aplikácie Google Hangouts

Na obídenie certificate pinningu sa dá využiť niektorý z dostupných nástrojov na deaktivovanie kontroly SSL certifikátov. Tie fungujú upravením správania systémových funkcií určených na ich overovanie tak, aby každý z nich prehlásili za dôveryhodný. Príklady týchto nástrojov sú iOS SSL Kill Switch[8] alebo TrustMe[9]. Pri analýze bol použitý nástroj TrustMe, ktorý takto upravuje správanie systémovej funkcie *SecTrustEvaluate*[10]. Po jeho aktivácii spolu s SSLsplitom, aplikácia úspešne nadviazala SSL spojenie a bola použiteľná, pričom jej komunikácia mohla byť dešifrovaná.

3.2.1 HTTP API

Po následnom sledovaní dešifrovanej komunikácie bolo vidno, ako aplikácia primárne komunikuje so servermi *googleapis.l.google.com*, *plus.google.com* a *googlehosted.l.googleusercontent.com* pomocou protokolu HTTP(S) na TCP porte 443. Zo serverov *plus.google.com* a *googlehosted.l.googleusercontent.com* boli sťahované profilové obrázky užívateľov. Na druhú stranu zasielanie informácií zo zariadenia, ako sú napríklad zmena stavu aplikácie alebo odoslanie chatovej správy, prebiehalo pomocou HTTP POST

[5] "SSL — The Wireshark Wiki", 2015, https://wiki.wireshark.org/SSL.

[6] Ïnstalling the root CA on iOS — IBM Knowledge Center", 2015, https://www-01.ibm.com/support/knowledgecenter/SSHSCD_7.0.0/com.ibm.worklight.installconfig.doc/admin/t_installing_root_CA_iOS.html.

[7] "Certificate and Public Key Pinning — OWASP", 2016, https://owasp.org/index.php/Certificate_and_Public_Key_Pinning.

[8] iSEC Partners, ïOS SSL Kill Switch", 2012, https://github.com/iSECPartners/ios-ssl-kill-switch.

[9] Intrepidus Group, "TrustMe", 2013, https://github.com/intrepidusgroup/trustme.

[10] "Certificate, Key, and Trust Services Reference — iOS Developer Library", 2016, https://developer.apple.com/library/prerelease/ios/documentation/Security/Reference/certifkeytrustservices/index.html.

požiadaviek pre HTTP API (Appplication Programming Interface) na servery *googleapis.l.google.com*. URL týchto požiadaviek určovalo druh zasielaných informácií.

3.2.2 Google Protocol Buffers

Podla hodnoty *Content-Type* v hlavičke daných HTTP správ, ktorá bola nastavená na *application-x-protobuff* a ich následného binárneho obsahu, sa dalo vyvodiť použitie technológie Google Protocol Buffers[11]. Tá slúži na serializáciu štrukturovaných dát binárnej podoby. Štruktúra dát je definovaná v samostatnom súbore, ktorý je použitý pri serializácii a deserializácii[12]. Pred preložením aplikácie je na základe tohto súboru vygenerovaný zdrojový súbor príslušného programovacieho jazyka, ktorý obsahuje rozhranie pre prácu s týmito objektami. Daná definícia štruktúry týchto objektov je nutná pre ich deserializovanie, bez nej by nebolo možné (spoľahlivo) dostať pôvodné štrukturované dáta. Protokol použitý aplikáciou Hangouts je však proprietárny, jeho definícia nie je voľne dostupná. Našťastie autor projektu hangups[13] bol schopný jeho reverznou analýzou obnoviť túto definíciu. Tá bola následne využitá pri spätnom deserializovaní zachytených binárnych dát do formy objektov.

3.2.3 Zasielané správy

V tomto momente už mohla prebehnúť analýza samotného komunikačného protokolu. Bolo vybraných niekoľko zaujímavých udalostí z pohľadu forenznej analýzy.

Zaslanie chatovej správy

Pri zaslaní chatovej správy je odoslaná POST požiadavka na URL *googleapis.l.google.com/chat/v1ios/conversations/sendchatmessage?alt=proto* obsahujúca serializovaný objekt triedy *SendChatMessageRequest*[14]. Na obrázku 3.1 je možné vidieť príklad deserializovaného obsahu tohto objektu v nástroji ProtoBufEditor[15].

Obsah samotných chatových správ je uložený v položkách *text* objektov triedy *Segment*. Pole týchto objektov je následne uložené v položke *segments* položky *message_content*.

Položka *event_request_header* obsahuje objekt triedy *EventRequestHeader*, ktorý v položke *conversation_id* nesie identifikátor konverzácie do ktorej bola zaslaná daná chatová správa. Spôsob ktorým by bolo možné identifikovať konkrétnych účastníkov danej konverzácie sa nepodarilo odhaliť.

V každom zo zasielaných objektov sa taktiež nachádza objekt triedy *RequestHeader* uložený v položke *request_header*, ktorý obsahuje informácie o zasielanej požiadavke. Ten obsahuje položku *client_identifier* nesúcu identifikátor užívateľa a položku *client_version* s objektom triedy *ClientVersion* ktorý obsahuje informácie o zariadení, ako napríklad verzia operačného systému a hardwaru.

Informovanie o rozpísaní chatovej správy

Pri písaní chatových správ sa taktiež zasielajú informácie o ich rozpísaní. Tie sú zasielané POST požiadavkou na *googleapis.l.google.com/chat/v1ios/conversations/settyping?alt=proto*

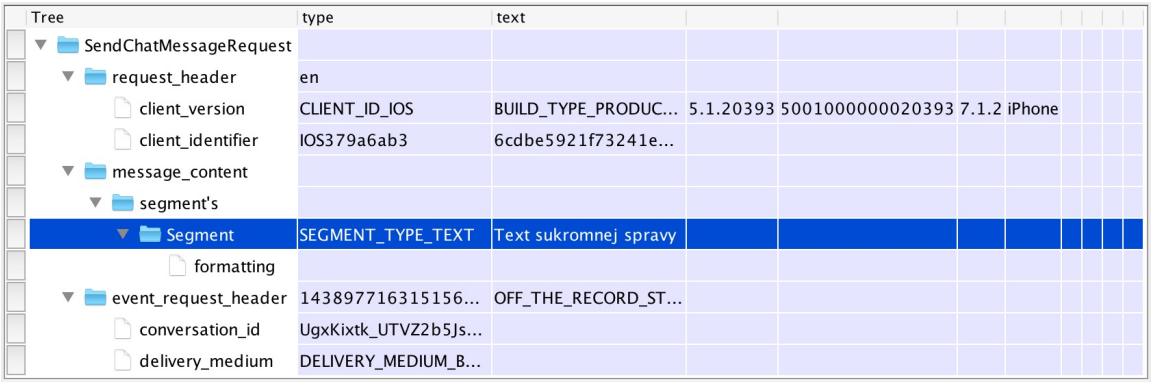
[11] Google, "Protocol Buffers", 2016, https://developers.google.com/protocol-buffers/.

[12] "Language Guide — Protocol Buffers — Google Developers", 2016, https://developers.google.com/protocol-buffers/docs/proto.

[13] Tom Dryer, "hangups", 2016, https://github.com/tdryer/hangups.

[14] Názvy tried a ich dielčich položiek sú prevzaté z projektu hangups.

[15] Bruce Martin, "ProtoBufEditor", 2014, http://sourceforge.net/projects/protobufeditor



Obr. 3.1: Vizualizácia deserializovaného objektu triedy SendChatMessageRequest

s objektom triedy *SetTypingRequest*, ktorého položka *type* informuje o stave rozpísania. Tá môže nadobúdať nasledovné hodnoty:

- `TYPING_TYPE_UNKNOWN` - neznámy stav
- `TYPING_TYPE_STARTED` - rozpísanie chatovej správy
- `TYPING_TYPE_PAUSED` - prerušenie písania so zadaným obsahom
- `TYPING_TYPE_STOPPED` - prerušenie písania bez žiadneho zadaného obsahu

Informovanie o stave aplikácie

Pri spustení, minimalizovaní alebo maximalizovaní aplikácie je odoslaná POST požiadavka na URL *googleapis.l.google.com/chat/v1ios/clients/setactiveclient?alt=proto* obsahujúca objekt triedy *SetActiveClientRequest*. Jeho položka *is_active* je tvorená boolean hodnotou, ktorá informuje o stave aplikácie. Na základe toho je možné zistiť, kedy ju mal užívateľ spustenú.

3.3 Analýza aplikácie Twitter

Rovnako ako aplikácia Hangouts, aj Twitter využíva certificate pinning. No po zopakovaní postupu z minulého prípadu, teda deaktivovaní kontroly SSL certifikátov nástrojom TrustMe, aplikácia nenadviazala úspešné spojenie. To sa dalo vysvetliť využitím vlastného, neštandardného spôsobu overovania certifikátov.

3.3.1 Deaktivovanie kontroly SSL certifikátov

Z tohoto dôvodu bolo nutné zanalyzovať samotný kód aplikácie, lokalizovať v ňom časť, ktorá je zodpovedná za overovanie certifikátov a následne ju upraviť tak, aby prijala aj podvrhnutý.

Na analýzu spustiteľných súborov je možné využiť tzv. disasemblery, nástroje ktoré umožňujú previesť strojový kód daného programu do jazyka symbolických adries (assembleru), vhodne ho vizualizovať a umožniť jeho editáciu. Medzi najznámejšie patrí nástroj IDA Pro od spoločnosti Hex-Rays[16]. Avšak kvôli jeho vysokej cene bola využitá jeho

[16]Hex-Rays, ĬDA Pro", 2015, https://hex-rays.com/products/ida.

alternatíva Hopper disasembler[17]. Ten je navyše primárne zameraný na analýzu aplikácií napísaných v Objective-C (primárnom programovacom jazyku systémov Mac OS X a iOS)[1]. To, v kombinácii s faktom, že tento jazyk využíva neskorú väzbu[18], značne uľahčuje samotnú analýzu aplikácie, pretože sú k dispozícií informácie o názvoch použitých tried a metód. Vďaka tomu je možné lepšie a rýchlejšie odhadnúť ich význam a účel.

Dešifrovanie aplikácie

Všetky aplikácie v systéme iOS, presnejšie ich spustiteľné súbory, sú uložené v zašifrovanej podobe a preto aby bola možná ich analýza je nutné aby boli dešifrované. Na to bolo možné využiť fakt, ze pred samotným spustením vykonávania kódu aplikácie procesorom, musí byť tento kód najprv dešifrovaný samotným systémom (zavádzačom aplikácií). Dešifrovaný kód aplikácie uložený v pamäti RAM sa môže následne začať vykonávať.

Je preto možné aplikáciu spustiť, nechať si systémom dešifrovať jej kód a nahrať ho do pamäte RAM. Následne sa stačí na ňu pripojiť pomocou debuggeru a skopírovať úsek jej adresného priestoru obsahujúci dešifrovaný kód a ten vložiť na miesto zašifrovaného kódu v aplikácii[19]. Na automatizovanie tohto postupu je taktiež možné využiť nástroj idb[20]

Analýza kódu aplikácie

Následnou analýzou aplikácie v disasembleri Hopper bolo zistené, že trieda *TFNTwitterTLSTrustEvaluator* je zodpovedná za verifikáciu predložených SSL certifikátov. Samotná verifikácia prebieha pomocou jej metódy *evaluateServerTrust:forHost:*, ktorá v prípade, že sa pripája na server *api.twitter.com*, nepriamo zavolá metódu *_isPinnedSPKI:* a tá porovná SHA1 hash[21] verejného kľúča predloženého certifikátu s hashom `1a21b4952b6293ce18b365ec9c0e934cb381e6d4`. Ak sa zhodujú, certifikát je akceptovaný.

Úprava aplikácie

Ďalej stačilo už len upraviť správanie tejto metódy tak, aby jej výsledok bol zakaždým pozitívny. To sa dalo docieliť napríklad upravením samotného kódu aplikácie, čo by ale znamenalo závislosť na danej verzií aplikácie a s jej updatom by sa táto zmena stratila. Ďalšou alternatívou bolo využitie nástroja Logos[22] na zavedenie tzv. hookov do aplikácie. Pomocou nich je možné jednoducho nahradiť implementáciu konkrétnych metód alebo funkcií bez nutnosti úpravy samotných aplikácií. Túto techniku využíva taktiež už spomenutý nástroj TrustMe. Výpis 3.1 zobrazuje hook ktorý upraví metódu *_isPinnedSPKI:* tak, aby sa vždy vyhodnotila pozitívne. Po jeho aktivácii aplikácia prijala podvrhnutý certifikát a bola použiteľná. Tak bolo možné prejsť na ďalší krok, analýzu samotnej komunikácie.

Výpis 3.1: Hook nahradzujúci metódu *_isPinnedSPKI*

```
%hook TFNTwitterTLSTrustEvaluator
    + (BOOL)_isPinnedSPKI:(id)arg1 {
```

[17] Cryptic Apps, "Hopper", 2015, http://hopperapp.com.

[18] Taktiež nazývaná dynamickou väzbou. Pri invokovaní danéj metódy objektu, je jej implementácia zvolená až pri behu, podľa jej názvu.

[19] "Decrypting iOS Apps — Information Intoxication", 2013, http://infointox.net/?p=114.

[20] Daniel A. Mayer, ïdb", 2015, https://github.com/dmayer/idb.

[21] D. Eastlake, P. Jones, ÜS Secure Hash Algorithm 1 (SHA1)", 2001, https://tools.ietf.org/html/rfc3174.

[22] "Logos", 2015, http://iphonedevwiki.net/index.php/Logos.

```
        return YES;
    }
%end
```

3.3.2 HTTP API

Sledovanie dešifrovanej komunikácie nástrojom Wireshark odhalilo, že aplikácia komunikuje so serverom *api.twitter.com* pomocou protokolov HTTP a na TCP porte 443. HTTP protokolom sú prenášané primárne nastavenia aplikácie a profilové obrázky užívateľov. SPDY je využívané pri ostatných činnostiach, ako vytvorenie nového *tweetu*, vyhľadávanie užívateľov a prezeranie ich verejných profilov, alebo zasielanie súkromných správ.

3.3.3 Protokol SPDY

Protokol SPDY je sieťový protokol umožňujúci prenos webového obsahu. Hlavnou dôvodom jeho vývoja bolo zníženie latencie a zvýšenie bezpečnosti dátových prenosov. Jeho cieľom nebolo kompletne nahradiť protokol HTTP, ale len upraviť spôsob, akým sú pomocou neho prenášané dáta. Oproti protokolu HTTP poskytuje tieto vylepšenia[3]:

- Udržovanie len jedného TCP spojenia medzi klientom a serverom, v ktorom je možné súčasne vybavovať viacero požiadaviek naraz.
- Prioritizácia požiadaviek.
- Kompresia hlavičiek.
- Asynchrónne zasielanie dát zo servera.

Komunikácia vo vytvorenom TCP spojení sa skladá z tzv. streamov, ktoré sa ďalej skladajú z tzv. rámcov[3]. Tie majú binárnu štruktúru (na rozdiel od textových HTTP správ). Rámce sa delia na dátové alebo kontrolné. Dátové rámce nesú samotný obsah SPDY správ, čím sú ekvivalentom tela HTTP správ. Na druhej strane kontrolné rámce slúžia na riadenie samotného spojenia a streamov, pričom ich je viacero druhov. Najdôležitejšie kontrolné rámce, určené na riadenie streamov sú nasledovné:

- SYN_STREAM - vytvorenie nového streamu. Predstavuje ekvivalent hlavičky HTTP požiadavky.
- SYN_REPLY - odpoveď na vytvorenie nového streamu. Predstavuje ekvivalent hlavičky HTTP odpovede.
- SYN_RST - uzatvorenie streamu.

Obdoba výmeny HTTP požiadavky a odpovede preto pri SPDY prebieha pomocou streamu, zaslaním SYN_STREAM a prípadného dátového rámca, načo je prijatý SYN_REPLY s prípadným dátovým rámcom. Tieto rámce obsahujú číselný identifikátor príslušného streamu, aby bolo možné rozoznať, ku ktorému streamu patria. Ďalej taktiež obsahujú príznak FIN, indikujúci, že odosielateľ zaslal posledný rámec daného streamu.

3.3.4 Zasielané správy

Popis API využívaného na komunikáciu so službou, je zdokumentovaný samotným Twitterom[23]. Ako sa však ukázalo, nie všetky API metódy využívané aplikáciou sa zhodovali s tými v danej dokumentácii. Popis vybraných správ zasielaných protokolom SPDY je uvedený nižšie

Zaslanie chatovej správy

Pri zaslaní chatovej správy je vygenerovaná POST požiadavka na URL *api.twitter.com/1.1/dm/new.json*. Jej obsah je uložený v kódovaní *application/x-www-form-urlencoded*. Príklad obsahu takejto požiadavky je možné vidieť vo výpise 3.2. Najzaujímavejšie parametre tvoria *converation_id*, ktorý je zložený z identifikátoru adresáta a odosielateľa správy a *text* s obsahom správy.

Výpis 3.2: Príklad obsahu správy zaslanej pri odoslaní chatovej spravy

```
cards_platform=iPhone-13
conversation_id=323270758-4589554409
dm_users=1
include_cards=1
request_id=47D34EBC-45AB-44F8-9E5C-6C05615A5488
text=Obsah sukromnej spravy
```

Vytvorenie *tweetu*

Vytvorenie nového *tweetu* prebieha POST požiadavkou na *api.twitter.com/1.1/statuses/update.json*. Jej obsah je rovnako ako v prípade odoslania chatovej správy, uložený v kódovaní *application/x-www-form-urlencoded*. Príklad obsahu takejto požiadavky je možné vidieť vo výpise 3.3. Text nového *tweetu* je uložený v parametri *status*.

Výpis 3.3: Príklad obsahu správy zaslanej pri pri vytvorení *tweetu*

```
cards_platform=iPhone-13
contributor_details=1
include_cards=1
include_entities=1
include_media_features=true
include_my_retweet=1
include_user_entities=true
status=Novy tweet
```

Vyhľadávanie užívateľov

Vyhľadávanie užívateľov prebieha v dvoch fázach. V prvej užívateľ zadáva do vyhľadávacieho poľa výraz na vyhľadanie, pričom sú mu už počas písania zobrazované predbežné výsledky a v druhej, kedy dopísal daný výraz a potvrdil vyhľadávanie, načo sa mu zobrazí konečný výsledok vyhľadávania.

Predbežné vyhľadávanie prebieha zaslaním GET požiadavky na *api.twitter.com/1.1/search/typeahead.json* s parametrom *q* obsahujúcim rozpísaný výraz na

[23] "REST APIs — Twitter Developers", 2015, https://dev.twitter.com/rest/public.

vyhľadanie. Nasledujúce konečné vyhľadanie prebieha zaslaním GET požiadavky na *api.twitter.com/1.1/search/universal.json* s parametrom *q* obsahujúcim kompletný výraz na vyhľadanie.

Zistenie vzťahu medzi užívateľmi

Počas vyhľadávania sa taktiež zasielajú GET požiadavky na *api.twitter.com/1.1/friendships/lookup.json*, pomocou ktorých aplikácia zisťuje vzťah prihláseného užívateľa k ďalším užívateľom (či si odoberajú navzájom príspevky). Parameter požiadavky *user_id* obsahuje zoznam identifikátorov užívateľov oddelených pomocou čiarky.

Pri zobrazení profilu užívateľa sú získané detailnejšie informácie o vzťahu k nemu z pohľadu prihláseného užívateľa pomocou GET požiadavky na *api.twitter.com/1.1/friendships/show.json*. Parametre požiadavky *source_id* a *target_id* obsahujú identifikátory užívateľov medzi ktorými sa má zistiť vzťah.

Zobrazenie časovej osi

Získanie zoznamu najnovších *tweetov* z časovej osi prihláseného užívateľa prebieha zaslaním GET požiadavky na *api.twitter.com/1.1/timeline/home.json*. Pri zobrazení časovej osi ostatných užívateľov je zoznam ich najnovších *tweetov* získaný pomocou GET požiadavky na *api.twitter.com/1.1/timeline/user.json*, pričom parameter požiadavky *id* obsahuje identifikátor užívateľa ktorého, časová os sa má získať.

3.4 Analýza aplikácie Facebook Messenger

Rovnako ako v prípade aplikácie Twitter, ani Messenger neprijal podvrhnutý SSL certifikát, aj napriek použitiu nástroja TrustMe. Tým pádom bola takisto nutná analýza samotnej aplikácie.

3.4.1 Deaktivovanie kontroly SSL certifikátov

Postup bol rovnaký, ako prípade aplikácie Twitter. Aplikácia bola najprv dešifrovaná a následne zanalyzovaná pomocou disassembleru. Presný spôsob overovania certifikátov sa nepodarilo identifikovať, bola však nájdená trieda *FBLigerConfig* s inicializačnou metódou obsahujúcou parameter *sslPinningEnabled*. Pri sledovaní vykonávania tejto metódy za behu, daný parameter bol zakaždým nastavený na `YES` (`true` v jazyku C). Vytvorenie hooku zobrazeného vo výpise 3.4, ktorý pred volaním tejto metódy pozmenil daný parameter na hodnotu `NO` (`false` v jazyku C), malo za následok akceptovanie podvrhnutého certifikátu a úspešné nadviazanie komunikácie.

Výpis 3.4: Hook pozmeňujúci hodnotu parametra *sslPinningEnabled*

```
%hook FBLigerConfig
    - (id)initConfigWithLigerEnabled:(BOOL)arg1 allRequestsEnabled:(BOOL)arg2 \
        cloudFrontEnabled:(BOOL)arg3 dnsCacheEnabled:(BOOL)arg4 \
        requestProcessLogEnabled:(BOOL)arg5 useStaleAnswers:(BOOL)arg6 \
        cAresEnabled:(BOOL)arg7 dnsRequestsOutstanding:(int)arg8 \
        dnsFallbackAnswers:(id)arg9 persistentSSLCacheEnabled:(BOOL)arg10 \
        persistentSSLCacheFilename:(id)arg11 persistentSSLCacheCapacity:(int)arg12 \
        persistentSSLCacheSyncInterval:(int)arg13 crossDomainSSLCacheEnabled:(BOOL)arg14 \
        httpSessionManagerMaxIdleHTTPSessions:(int)arg15 \
        httpSessionManagerMaxIdleSPDYSessions:(int)arg16 \
```

```
        httpSessionManagerBackupTimeout:(int)arg17 \
        httpSessionManagerReuseUnfilledSessionsEnabled:(BOOL)arg18 \
        httpSessionManagerOriginLimitEnabled:(BOOL)arg19 \
        httpSessionManagerOriginMaxRetries:(int)arg20 hpackEnabled:(BOOL)arg21 \
        sslPinningEnabled:(BOOL)arg22 schedulerEnabled:(BOOL)arg23 \
        schedulerLowPriUpperBound:(int)arg24 schedulerNormalPriUpperBound:(int)arg25 \
        schedulerHighPriUpperBound:(int)arg26 tracerouteEnabled:(BOOL)arg27 \
        heV4PrefEnabled:(BOOL)arg28 tcpPingEnabled:(BOOL)arg29 tcpPingRetries:(int)arg30 \
        tcpPingTimeout:(int)arg31 tcpPingCancelOnReply:(BOOL)arg32 \
        ligerNetworkStatusMonitor:(id)arg33 requestObservers:(id)arg34 {
            // Pozmenenie parametra sslPinningEnabled
            arg22 = NO;
            // Spustenie povodnej metody
            return %orig;
    }
%end
```

3.4.2 Protokol MQTT

Následné sledovanie dešifrovanej komunikácie vo Wiresharku spolu s ďalšou analýzou aplikácie odhalilo, že na výmenu dát je využitý protokol MQTT. Pomocou neho aplikácia komunikuje cez TCP port 443 s jedným zo serverov Facebooku s doménovým menom *edge-mqtt-shv-xx-yyy.facebook.com*, kde *xx* a *yyy* predstavujú označenie konkrétneho serveru. Príklady týchto domén sú *edge-mqtt-shv-01-vie1.facebook.com* alebo *edge-star-shv-01-amt2.facebook.com*. Výber jedného z viacerých takýchto serverov umožňuje službe lepšie rozložiť záťaž.

Message Queue Telemetry Transport (MQTT) je tzv. publish/subscribe protokol, umožňujúci klientom prihlasovanie k odberu správ daného druhu tzv. topicu, spolu s ich vytváraním[2]. Je zameraný na nízku réžiu a veľkosť správ. Obsahuje radu príkazov ktoré si môžu klient a server zaslať. Medzi tie najdôležitejšie patria:

- CONNECT - zasielaný klientom po vytvorení TCP spojenia. Môže obsahovať autentizačné údaje.
- SUBSCRIBE - pomocou nich sa klient prihlasuje k odberu správ z jedného alebo viacerých topicov.
- PUBLISH - prenášajú správy vytvorené klientom alebo serverom v danom topicu. Spolu s obsahom samotnej správy, tieto pakety taktiež obsahujú aj názov topicu do ktorého je daná správa zaslaná.
- PINGREQ a PINGRESP - príkazy určené na sledovanie živosti spojenia.

Využitie protokolu MQTT aplikáciou Messenger

Správy vymieňané medzi aplikáciou a službou Facebook sú prenášané pomocou PUBLISH príkazov, pričom každý typ správy má pridelený svoj názov topicu. Popis vybraných správ je uvedený nižšie. Ich obsah je skomprimovaný pomocou algoritmu *DEFLATE*[24] knižnicou

[24] P. Deutsch, "RFC 1951 DEFLATE Compressed Data Format Specification version 1.3", IETF, 1996, http://rfc-editor.org/info/rfc6101.

zlib. Po jeho následnej dekompresii, môže v závislosti na topicu pozostávať z JSON[25] reťazca alebo z objektu serializovaného technológiou Apache Thrift[26].

3.4.3 Apache Thrift a obnova definície objektov

Apache Thrift, je podobne ako Google Protocol Buffers, technológia umožňujúca serializáciu a deserializáciu objektov. Zo zdrojového súboru definujúceho štruktúru Thrift objektov[27] sú vygenerované zdrojové súbory vybraného programovacieho jazyka, obsahujúce triedy reprezentujúce dané Thrift objekty. Pomocou nich je následne možné následne tieto objekty serializovať a deserializovať

Aj v tomto prípade platí, že je nutné poznať samotnú definíciu štruktúry serializovaných objektov, aby bola možná ich deserializácia. Tá však pre aplikáciu Messenger nebola nikde dostupná a preto bola nutná jej obnova zo samotnej aplikácie. Obnova preto prebiehala pomocou extrakcie informácií o použitých triedach v aplikácii vo formáte, v akom by boli zapísané v objektovom C pomocou nástroja *class-dump*[28] a výberom tých tried, ktoré boli zdedené z triedy *TBase*. Tie predstavovali Thrift objekty. Príklad zrekonštruovaných informácií o takejto triede je možné vidieť vo výpise 3.5. Ďalej boli podľa ich názvov a premenných vytipované tie, ktoré by mohli byť používané pri vybranej komunikácii (ako napríklad pripojenie alebo zaslanie chatovej správy). Tieto predpoklady sa následne overili pomocou spôsobu ich využitia v aplikácii pomocou disassembleru.

Posledným krokom bol spätný prepis informácií o vybraných triedach do jazyka definujúceho štruktúry Thrift objektov. V prípadoch, kedy daná trieda obsahovala hodnotu s dátovými typmi *NSMutableArray* alebo *NSMutableDictionary* (reprezentujúce dátový typ zoznam a slovník), bolo nutné zistiť dátové typy ich dielčich hodnôt s využitím disassembleru. Príklad výsledného Thrift objektu je možné vidieť vo výpise 3.6. Obnovená definícia štruktúr Thrift objektov, ktoré boli využívané pri vybranej komunikácii aplikácie sa nachádza na priloženom DVD.

Výpis 3.5: Príklad výstupu z nástroja *class-dump*

```
@interface FBMQTTConnectMessage : NSObject <TBase, NSCoding>
{
    NSString *__clientIdentifier;
    NSString *__willTopic;
    NSString *__willMessage;
    FBMQTTClientInfo *__clientInfo;
    NSString *__password;
    NSMutableArray *__getDiffsRequests;
    ...
}
```

Výpis 3.6: Príklad obnovenej štruktúry Thrift objektu

```
struct FBMQTTConnectMessage {
    1: string ClientIdentifier
    2: string WillTopic
    3: string WillMessage
    4: FBMQTTClientInfo ClientInfo
```

[25] T. Bray, "RFC 7159 The JavaScript Object Notation (JSON) Data Interchange Format", IETF, 2014, https://tools.ietf.org/html/rfc7159.

[26] Apache, "Thrift", 2015", http://thrift.apache.org/.

[27] "Thrift interface description language", Apache, 2016, https://thrift.apache.org/docs/idl.

[28] Steve Nygard, "class-dump", 2016, http://stevenygard.com/projects/class-dump/.

```
    5: string Password
    6: list<string> GetDiffsRequests
}
```

3.4.4 Zasielané a prijímané správy

Nasledujúce správy predstavujú potencionálny zdroj užitočných informácií pri forenznej analýze. Všetky z nich, okrem správy zaslanej pri pripojení, boli obsahom MQTT PUBLISH príkazov. Zároveň pre všetky z nich platí, ako už bolo spomenuté, že v danom MQTT príkaze je ich obsah skomprimovaný algoritmom DEFLATE.

Pripojenie

Po nadviazaní spojenia klient odosiela MQTT CONNECT príkaz, ktorého obsah pozostáva zo serializovaného *FBMQTTConnectMessage* objektu. Ten okrem iného ďalej obsahuje *ClientInfo* objekt, obsahujúci hodnotu *UserId*, pomocou ktorej je možné identifikovať daného užívateľa aplikácie.

Zasielanie chatovej správy

Na zasielanie chatových správ sú využité správy v topicu */t_sm*, ktoré obsahujú serializovaný *FBMQTTSendMessageRequest* objekt. Jeho hodnoty *To* a *Body* reprezentujú identifikátor príjemcu a obsah správy. Ďalšou zaujímavou hodnotou môže byť *FBMQTTLocationAttachment*, ktorá je využitá pri zaslaní polohy.

Prijatie chatovej správy

Správy v topicu */t_ms* sú využité na príjem synchronizačných údajov v chate, ako napríklad príjem novej chatovej správy, informácií o prečítaní chatovej správy, alebo o pridaní alebo odstránení užívateľa do daného chatu (konverzácie). Tieto správy sú serializované pomocou objektu *MNMessagesSyncClientPayload*, ktorý obsahuje zoznam *MNMessagesSyncDeltaWrapper* objektov.

MNMessagesSyncDeltaWrapper, ako už názov napovedá, zaobaľuje v sebe ďalšie objekty, ktoré reprezentujú konkrétne synchronizačné údaje. Jedným z nich je napríklad *MNMessagesSyncDeltaNewMessage*, ktorý je využitý na príjem nových chatových správ. Jeho hodnota *Body* reprezentuje obsah prijatej chatovej správy, pričom v jeho objekte *MessagesSyncMessageMetadata* sa nachádza informácia o odosielateľovi.

Informovanie o stave aplikácie

Ďalšie správy zasielané zo zariadenia tvoria informácie o zmene stavu aplikácie. Tie sú zasielané v topicu */foreground_state*. Ich obsah už ale netvorí serializovaný Thrift objekt, ale JSON reťazec, aký je možné vidieť na príklade zachytenej správy vo výpise 3.7. Hodnota *foreground* nesie informáciu o stave aplikácie, či je na popredí alebo je minimalizovaná. Hodnota *keepalive_timeout* zasa určuje interval zasielania MQTT PINGREQ príkazov.

Výpis 3.7: Príklad JSON reťazcu informujúceho o stave aplikácie

```
{
    "foreground" : 1,
    "keepalive_timeout" : 60
```

```
}
```

Informovanie o rozpísaní chatovej správy

Podobným spôsobom prebieha aj zasielanie informácií o rozpísaní chatových správ v topicu */typing*. Príklad zachytenej správy vo výpise 3.8 ukazuje, že tieto správy pozostávajú z identifikátora príjemcu danej správy a z informácie, či má užívateľ, v danom momente rozpísanú chatovú správu.

Výpis 3.8: Príklad JSON reťazcu informujúceho o rozpísaní chatovej správy

```
{
    "to" : "100010419973288",
    "state" : 1
}
```

Kapitola 4

Netfox Detective

Netfox Detective je nástroj umožňujúci forenznú analýzu zachytenej sieťovej komunikácie. Je vyvíjaný na pôde Fakulty informačních technológií Vysokého učení technického v Brně. Konečným cieľom tejto práce bolo rozšírenie tohto nástroja o moduly umožňujúce rekonštrukciu komunikácie vybraných aplikácií. Táto kapitola sa preto bude venovať jeho bližšiemu popisu jeho štruktúry, nutných úprav jeho existujúcich častí a implementácie tzv. snooperov.

4.1 Popis

Netfox Detective je napísaný v jazyku C# s využitím .NET frameworku. Je rozdelený na viacero modulov, pričom každý z nich plní samostatnú úlohu pri analýze komunikácie a následnom prezentovaní jej výsledkov užívateľovi[6]. Analýza komunikácie je riadená modulom *CoreController*. Pri jej zahájení, modul *PmLib* spracuje vstupný PCAP súbor so zachytenou komunikáciou a vyextrahuje z nej rámce. Tie sú následne postupne predané modulom *L3ConversationTracker*, *L4ConversationTracker* a *L7ConversationTracker*, ktoré ich spracujú podľa vrstiev OSI modelu. Výstupom modulu *L7ConversationTracker* sú *L7PDU* objekty, ktoré reprezentujú samotné správy aplikačných protokolov. Modul *PDUStreamReader* následne poskytuje jednotné rozhranie pre ďalšie moduly ako pomocou modulov *PDUStreamBasedProvider* a *PDUDecrypterBase* čítať *L7PDU* objekty. Využitie modulu *PDUDecrypterBase* umožňuje transparentné dešifrovanie *L7PDU* objektov obsahujúce SSL komunikáciu.

Samotná analýza aplikačných protokolov prebieha pomocou tzv. snooperov. Tieto moduly si pomocou *PDUStreamReader* modulu preberajú spracované dáta z *L7PDU* objektov (či už v ich priamej alebo dešifrovanej podobe) a rekonštruujú z nich správy príslušného aplikačného protokolu. Výsledkom ich činnosti sú tzv. export objekty, ktoré reprezentujú zrekonštruované správy. Tie sa môžu prezentovať užívateľovi, alebo sa ďalej dajú využiť ako vstup do ďalšieho snooperu, čo umožňuje ich reťazenie za sebou. Príkladom môže byť *SnooperHangouts*, ktorý spracováva export objekty vytvorené modulom *SnooperHTTP*.

Triedy implementujúce funkcionalitu snooperov sú zdedené z abstraktnej triedy *SnooperBase*. Tie okrem iného obsahujú implementáciu abstraktnej metódy *ProcessConversation*, ktorá je jadrom samotnej rekonštrukcie daného aplikačného protokolu. Bližší popis fungovania jednotlivých snooperov bude vysvetlený v samostatných podkapitolách.

4.2 Úpravy dešifrovacieho modulu

Vybrané aplikácie, respektíve operačný systém iOS, pri komunikácii protokolom SSL využívali niektoré jeho vlastnosti, ktoré *PDUDecrypterBase*, modul dešifrujúci SSL komunikáciu neimplementoval a z toho dôvodu nebol schopný dešifrovať ich zachytenú komunikáciu. Preto pred ďalšou implementáciou snooperov bolo nutné tieto nedostatky odstrániť.

Samostatné zasielanie správ SSL handshaku

Prvým problémom bol priebeh nadväzovania samotného SSL spojenia. Na jeho začiatku vždy prebehne tzv. handshake, pri ktorom sa komunikujúce strany dohodnú na verzíi SSL protokolu, kryptografických protokoloch, vymenia si certifikáty a bezpečne si vymenia vygenerovaný premaster key[5]. SSL handshake je tvorený postupnosťou správ, ktoré si komunikujúce strany zašlú, pričom je možné, aby za sebou idúce správy boli zaslané naraz, v jednom pakete.

Systém iOS však pri nadväzovaní SSL spojenia tieto správy zasielal oddelene, každú v samostatnom pakete. S touto možnosťou *PDUDecrypterBase*, presnejšie jeho stavový automat implementujúci spracovanie SSL správ handshaku, nepočítal a tak nebol schopný spracovať takýto handshake. Riešením bolo upravenie daného stavového automatu tak, aby bol schopný prijať takúto sekvenciu správ.

Opätovné použitie SSL relácií

Ďalšou vlastnosťou protokolu SSL využívanou iOS aplikáciami a neimplementovanou modulom *PDUDecrypterBase* bolo opätovné použitie SSL relácií (ang. session resumption). Tento mechanizmus umožňuje nadviazať na minulú SSL reláciu pomocou opätovného použtia jej master key, kedy nemusí dochádzať k jeho novému generovaniu a výmene (pomocou premaster key), vďaka čomu je nadväzovanie spojenia časovo a výpočtovo jednoduchšie[5].

Pri vytváraní novej relácie server v `Server Hello` správe nastavuje 32 bajtový identifikátor relácie. Klient si ho uloží a spáruje s novým master key (po vygenerovaní). Pri vytvorení ďalšej relácie klient nastaví tento identifikátor v `Client Hello` správe a pokiaľ ho server má v pamäti spolu s príslušným master key, v `Server Hello` správe nastaví rovnaký identifikátor relácie, načo následne zašle `Change Cipher Spec` správu oznamujúcu začatie šifrovanej komunikácie. Klient v tomto prípade nezasiela `Client Key Exchange` správu obsahujúcu nový zašifrovaný premaster key.

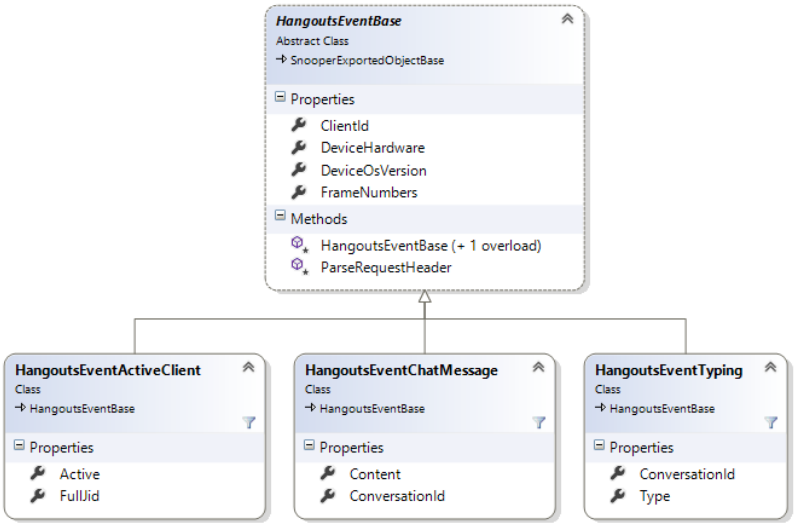
Na dešifrovanie komunikácie bolo však nutné zachytiť výmenu zašifrovaného premaster key (ktorý je pomocou privátneho kľuča dešifrovaný a je z neho odvodený master key) ku ktorej, ako bolo vidieť, pri znovu použitých reláciach nedochádza. Riešením preto bolo ukladať master key vytvorené pri nových reláciách spolu s identifikátormi ich relácií. Pri následnom opätovnom použití SSL relácie je podľa jej identifikátora obnovený master key, ktorý sa ďalej použije na dešifrovanie komunikácie.

4.3 Google Hangouts Snooper

Google Hangouts Snooper je implementovaný triedou *SnooperHangouts*. Ako jeho vstupné dáta slúžia *SnooperExportedDataObjectHTTP* objekty vytvorené modulom *SnooperHTTP*, obsahujúce *HTTPMsg* objekty. Tie predstavujú zrekonštruované HTTP správy, z ktorých sa následne v metóde *ProcessConversation* rekonštruuje vybraná komunikácia aplikácie.

Táto metóda pre každú HTTP požiadavku na server *googleapis.l.google.com* overí, či je jej URI jedno zo sledovaných, ktoré boli popísané v minulej kapitole. Ak áno, metódou *HandleEvent* sa vytvorí príslušný objekt podtriedy *HangoutsEventBase*, ktorý reprezentuje danú udalosť. Tieto objekty predstavujú výsledné export objekty tohto snooperu. Obrázok 4.1 ukazuje UML diagram objektov udalostí.

Konštruktorom týchto objektov je predaný obsah daných HTTP požiadaviek. Ten, ako už bolo popísané, pozostáva zo serializovaných dát technológiou Google Protocol Buffers. Ich prvotnou úlohou je preto tieto dáta vhodne deserializovať a následne z ich deserializovanej formy vybrať zaujímavé informácie. Na to bolo nutné najprv vygenerovať zdrojové súbory s triedami reprezentujúcimi dané objekty protokolu na základe získanej definície s nástrojom *protoc*[1]. Výpis 4.1 zobrazuje spôsob vytvorenia udalosti vygenerovanej pri zaslaní chatovej správy. V prvom kroku je z obsahu HTTP správy deserializovaný *SendChatMessageRequest* objekt, z ktorého je následne zostavený a okopírovaný obsah chatovej správy a identifikátor konverzácie, v ktorej bola zaslaná. Posledným krokom je zavolanie metódy *ParseRequestHeader*, ktorá spracuje *RequestHeader* objekt obsiahnutý v každom zaslanom objekte. Z neho je zkopírovaný identifikátor klienta, verzia použitého hardwaru a operačného systému. Rovnaký spôsob rekonštrukcie sa využíva aj pri zvyšných udalostiach.



Obr. 4.1: UML diagram tried objektov udalostí

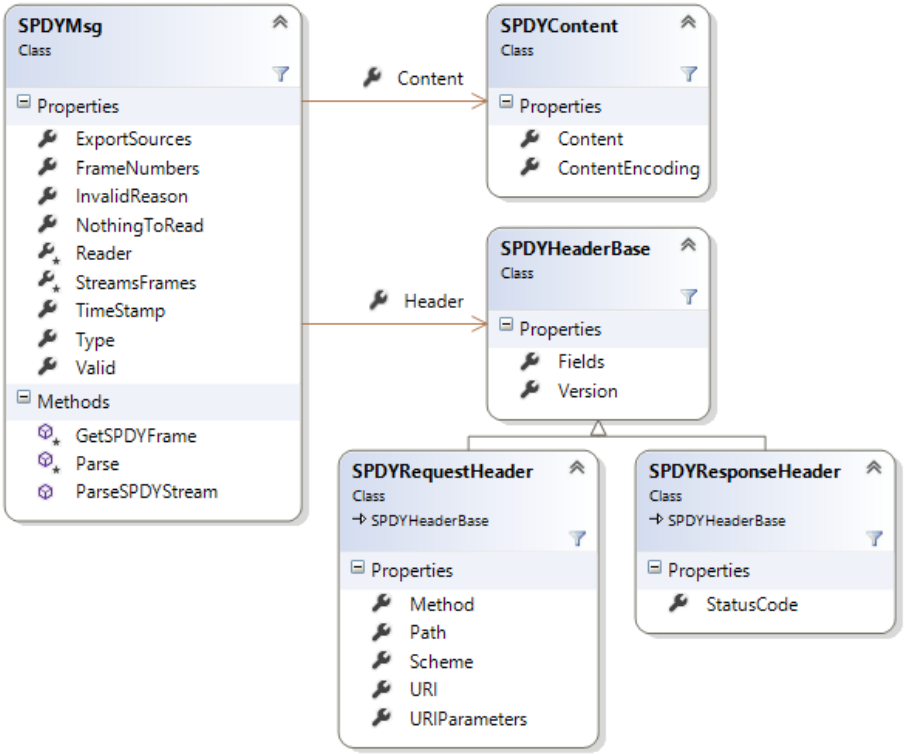
Výpis 4.1:

```
public HangoutsEventChatMessage(SnooperExportBase exportBase, byte[] protobufData)
    : base(exportBase)
{
    var sendChatMessageRequest = SendChatMessageRequest.Parser.ParseFrom(protobufData);
    this.Content = string.Join("",
        sendChatMessageRequest.MessageContent.Segment.Select(segment => segment.Text));
    this.ConversationId = sendChatMessageRequest.EventRequestHeader.ConversationId.Id;
    this.ParseRequestHeader(sendChatMessageRequest.RequestHeader);
}
```

[1]Google, "protobuf", 2016, https://github.com/google/protobuf.

4.4 SPDY Snooper

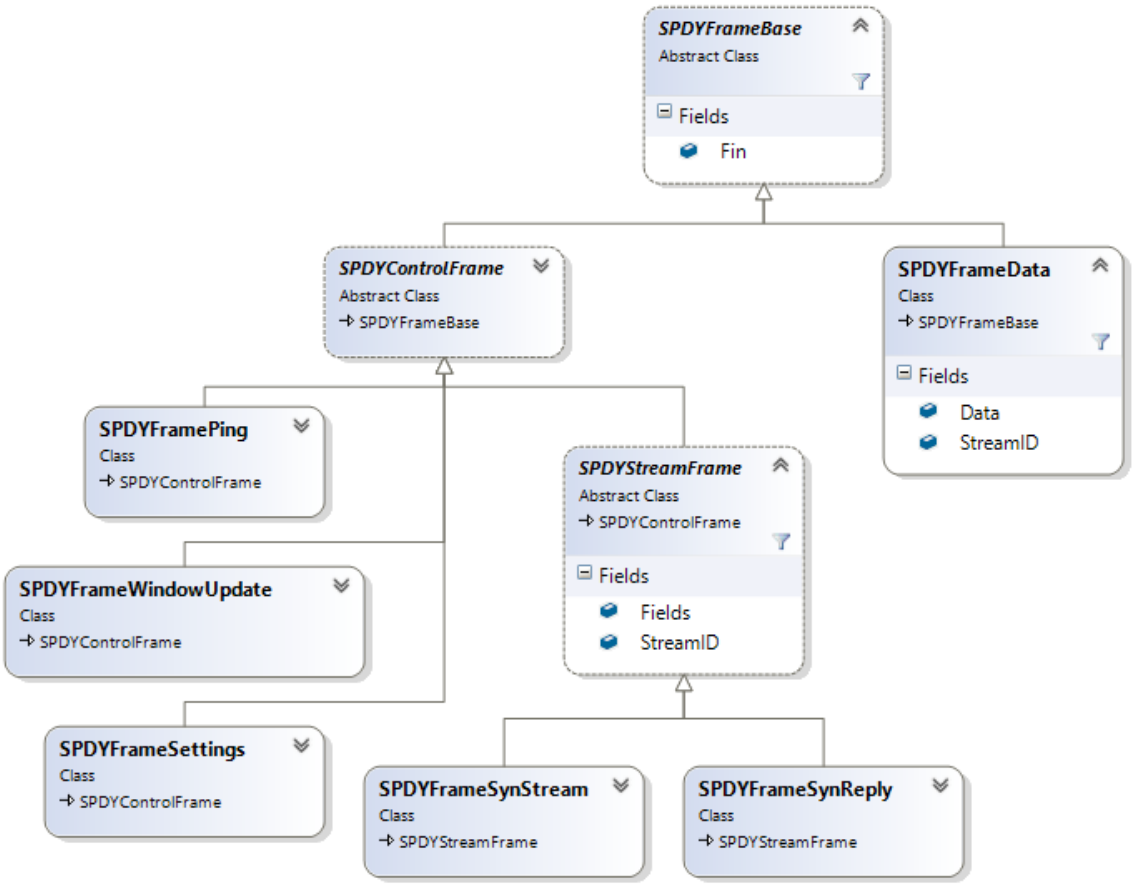
Aplikácia Twitter na svoju komunikáciu využívala protokol SPDY, pre ktorý nebol v danom momente implementovaný samostatný snooper. Preto prvým krokom vývoja Twitter snooperu bolo implementovanie SPDY snooperu. SPDY Snooper je implementovaný triedou *SnooperSPDY*. V jej metóde *ProcessConversation* sú vytvárané *SPDYMsg* objekty, ktorým je predaný *PDUStreamReader* objekt. Týmto spôsobom je samotná rekonštrukcia SPDY komunikácie riadená *SPDYMsg* objektami pri ich vytváraní. Jednými z hodnôt *SPDYMsg* objektov sú *Content* a *Header*, ktoré predstavujú obsah a hlavičku SPDY správ. Diagram týchto objektov je možné vidieť na obrázku 4.2. Pri následnom vytvorení *SPDYMsg* objektu, jeho konštruktor zavolá metódu *Parse*, ktorá prečíta a zrekonštruuje SPDY správu a uloží ju do hodnôt Content a *Header*. Nakoniec je tento objekt uložený v novom *SnooperSPDYExportedObject* objekte, ktorý predstavuje export tohto snooperu.



Obr. 4.2: UML diagram tried zostavujúcich SPDYMsg objekt

Ako bolo vysvetlené v časti popisujúcej SPDY protokol, prenos dát prebieha pomocou tzv. streamov, ktoré sa skladajú tzv. rámcov. Rozpoznávanie týchto rámcov je úlohou metódy *GetSPDYFrame*. Tá z daného *PDUStreamReader* objektu prečíta nový rámec, rozpozná ho a vytvorí preň objekt príslušnej podtriedy *SPDYFrameBase*, ktorým bude následne reprezentovaný. Diagram týchto tried je možné vidieť na obrázku 4.3. Medzi tie najpodstatnejšie objekty týchto patria *SPDYFrameSynStream* vytvorené zo SYN_STREAM rámcov, *SPDYFrameSynReply* vytvorené zo SYN_REPLY rámcov a *SPDYFrameData* vytvorené z dátových rámcov. Konštruktorom týchto a zvyšných *SPDYFrameBase* objektov sú predané prečítané dáta rámcov, z ktorých sa následne inicializujú. Pre *SPDYFrameSynStream* a*SPDYFrameSynReply* objekty, ich spoločný *SPDYStreamFrame* konštruktor uk-

ladá identifikátor streamu spolu so zoznamom HTTP hlavičiek. Tie sú na rozdiel od HTTP uložené v binárnej podobe a následne skomprimované využitím zlib knižnice[2]. Na ich dekomprimovanie bola použitá C# knižnica ZLIB.NET. Pred jej využitím musela prebehnúť jej úprava, pretože obsahovala niekoľko chýb. Všetky chyby sa však nepodarilo opraviť a tak v tomto momente, pri niektorých nezistených situáciach nie je schopná dané hlavičky dekomprimovať. *SPDYFrameData* konštruktor si ukladá identifikátor streamu spolu s telom SPDY správy.



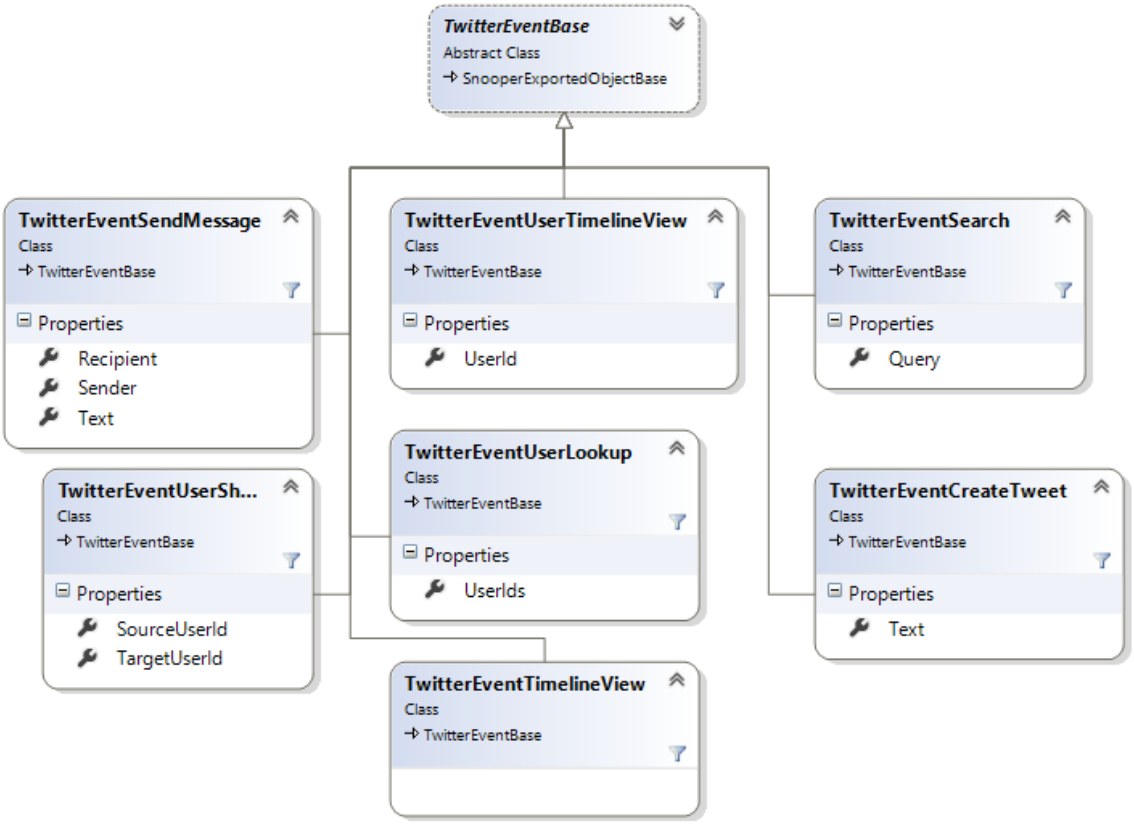
Obr. 4.3: UML diagram popisujúci triedy SPDY rámcov

GetSPDYFrame metóda je následne postupne volaná metódou *Parse*, pričom úspešne rozpoznané objekty rámcov sú ukladané do slovníka mapujúceho identifikátory streamov na zoznam ich rámcov. V momente, keď sa prečíta *SPDYFrameSynStream*, *SPDYFrameSynReply* alebo *SPDYFrameData* objekt s nastaveným FIN príznakom, je zavolaná metóda *ParseSPDYStream*, ktorej je predaný zoznamom rámcov posledného streamu[3]. Tá prejde dané rámce, pričom pre *SPDYFrameSynStream* alebo *SPDYFrameSynReply* objekt vytvorí *SPDYRequestHeader* alebo *SPDYResponseHeader* objekt, ktorý uloží do hodnoty *Header* a pre *SPDYFrameData* vytvorí *SPDYContent* objekt, ktorý je uložený do hodnoty *Content*. Konštruktorom týchto objektov sú predané dané objekty rámcov, z ktorých si následne preberú príslušné hodnoty.

[2] "zlib", 2016, http://zlib.net/.
[3] Streamu do ktorého patril aktuálne prečítaný rámec.

4.5 Twitter Snooper

Po úspešnom implementovaní SPDY snooperu bolo možné prejsť na samotný Twitter snooper. Ten je implementovaný triedou *SnooperTwitter*. Ako jeho vstupné dáta slúžia *SnooperSPDYExportedObject* objekty vytvorené SPDY snooperom. Tie, ako bolo popísané v minulej podkapitole, obsahujú *SPDYMsg* objekty reprezentujúce zrekonštruované SPDY správy. Princíp rekonštrukcie Twitter udalostí je ďalej zhodný s tým, ktorý využíva Google Hangouts snooper. Metóda *ProcessConversation* prechádza všetky dané SPDY správy a hladá v nich tie, ktorých URI sa zhoduje s URI niektorej zo sledovaných udalostí. Pri zhode je metódou *HandleEvent* vytvorený objekt príslušnej podtriedy *TwitterEventBase* s predanou SPDY správou. Z nej si daný konštruktor udalosti uloží príslušné hodnoty správy na základe jej typu (udalosti). Tieto hodnoty môžu byť obsiahnuté v URI alebo tele požiadavky. Následne je tento objekt vyexportovaný. Diagram popisujúci triedy udalosti je možné vidieť na obrázku 4.4.

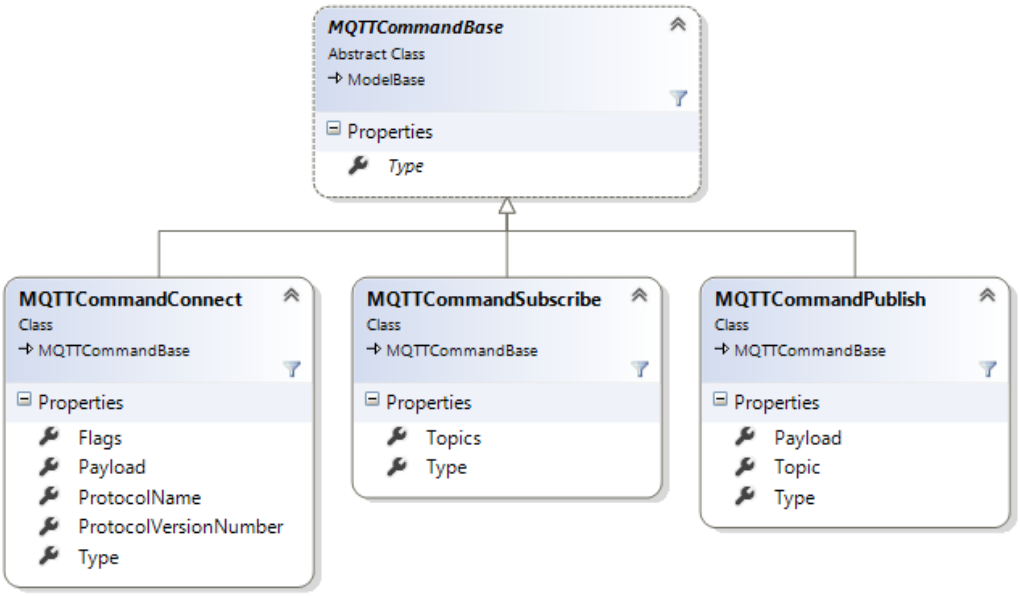


Obr. 4.4: UML diagram popisujúci triedy Twitter udalostí

4.6 MQTT Snooper

Podobne ako v prípade aplikácie Twitter, kedy bolo nutné vytvárať samostatný snooper pre SPDY komunikáciu, aj pri aplikácii Facebook Messenger bolo nutné najprv implementovať snooper schopný rekonštruovať MQTT komunikáciu. MQTT snooper je implementovaný triedou *SnooperMQTT*. V jej metóde *ProcessConversation* sú vytvárané *MQTTMsg* ob-

jekty podobným spôsobom, ako v prípade SPDY snooperu . Konštruktor týchto objektov volá metódu *Parse*, ktorá následne zavolá metódu *GetCommand*, ktorej úlohou je prečítať a zrekonštruovať nový MQTT príkaz. Tie sa skladajú z fixnej hlavičky obsahujúcej hlavne typ príkazu a obsahu samotného príkazu. Metóda *GetCommand* preto prečíta obe časti a následne podľa daného typu vytvorí objekt jednej z podtried *MQTTCommandBase*. Najvýznamnejšie z nich sú *MQTTCommandConnect*, *MQTTCommandSubscribe* a *MQTTCommandPublish*, ktoré je vidno na obrázku 4.5. Konštruktorom týchto objektov je predaný obsah príkazov, ktorý ďalej spracovávajú a ukladajú z neho podstatné informácie. Zrekonštruovaný MQTT príkaz je následne uložený v hodnote *Command* objektu *MQTTMsg*, ktorý nakoniec predstavuje finálny export snooperu.



Obr. 4.5: UML diagram popisujúci triedy vybraných MQTT príkazov

4.7 Facebook Messenger Snooper

Tento snooper je implementovaný triedou *SnooperMessenger*. Vstupné dáta predstavujú *MQTTMsg* objekty vytvorené MQTT snooperom. Tie sa v metóde *ProcessConversation* postupne prechádzajú, pričom sa pracuje s ich *Command* hodnotami obsahujúcimi príslušný MQTT príkaz. Z nich sú vytvárané objekty podtried triedy *MQTTEventBase*, ktoré reprezentujú samotné udalosti rozpoznané týmto snooperom. Diagram týchto tried je možné vidieť na obrázku 4.6.

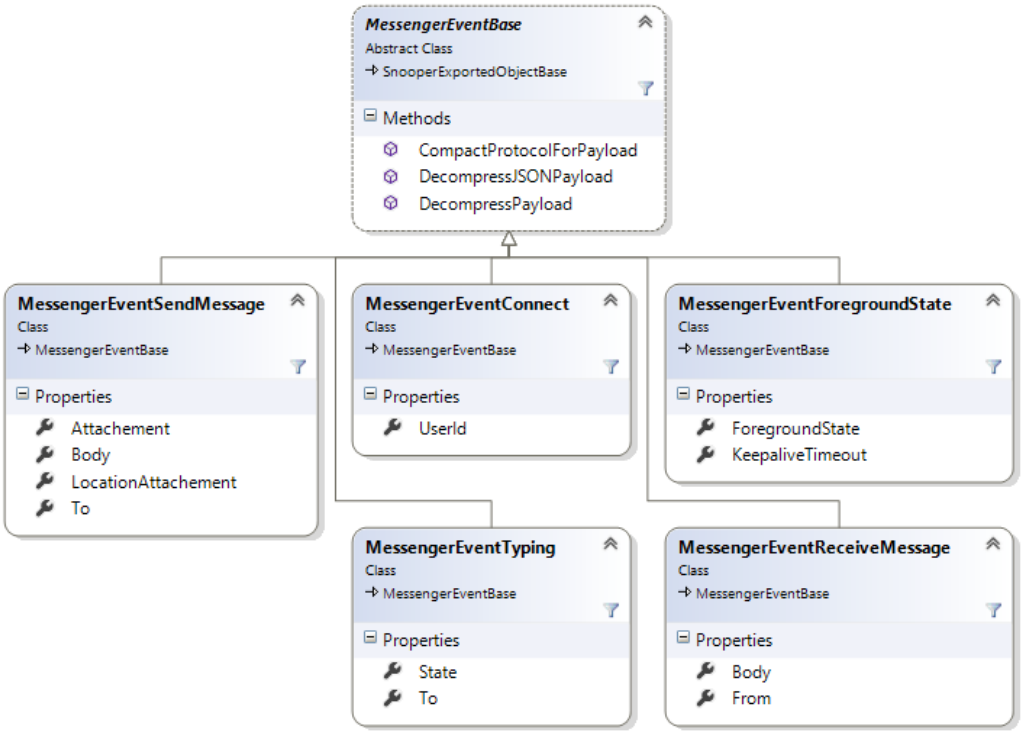
Pre MQTT Publish príkazy s topicom */foreground_state* a */typing* sú metódou *HandleEvent* vytvorené *MessengerEventForegroundState*, prípadne *MessengerEventTyping* objekty, ktorým je predaný obsah daného Publish príkazu. Ten je tvorený skomprimovaným JSON reťazcom. Ich konštruktory ho preto metódou *DecompressJSONPayload* dekomprimujú a s využitím knižnice Json.NET prevedú na C# objekt. Z neho si následne prevezmú príslušné hodnoty.

Pre MQTT Publish príkaz s topicom */t_sm* je metódou *HandleEvent* vytvorený *MessengerEventSendMessage* objekt, ktorému je tiež predaný obsah daného príkazu. Ten po

dekomprimovaní už nie je tvorený JSON reťazcom, ale serializovaným Thritf *SendMessageRequest* objektom. Na jeho spätné deserializovanie bolo nutné najprv vygenerovať príslušné zdrojové súbory týchto objektov z obnovenej štruktúry Thrift štruktúr[4], rovnako ako v prípade Google Hangouts snooperu. Po jeho následnom deserializovaní sú z neho uložené vybrané hodnoty.

V prípade MQTT Publish príkazu s topicom */t_ms* je obsah príkazu priamo dekomprimovaný a deserializovaný do formy *MNMessagesSyncClientPayload* Thrift objektu, ktorý je predaný metóde *HandleSyncMessage*. Tá následne prejde cez všetky objekty typu *MNMessagesSyncDeltaWrapper* v hodnote *Deltas* daného *MNMessagesSyncClientPayload* objektu. Ak daný má daný delta objekt nastavenú *DeltaNewMessage* hodnotu, vytvorí sa nový *MessengerEventReceiveMessage* objekt s predanou touto hodnotou, z ktorej si daný konštruktor prevezme informácie o prijatej správe.

V prípade MQTT Connect príkazu je metódou *HandleEvent* vytvorený *MessengerEventConnect* objekt. Ten daný obsah príkazu takisto dekomprimuje a deserializuje ho do podoby *ConnectMessage* Thrift objektu, z ktorého si prevezme identifikátor užívateľa, ktorý sa v danom momente prihlasoval.



Obr. 4.6: UML diagram popisujúci triedy Facebook Messenger udalostí

4.8 Testovanie

Testovanie snooperov prebiehalo pomocou Unit Testov, ktorých úlohou bolo overiť správnosť rekonštrukcie komunikácie. Vytvorenie testovacej sady prebiehalo zachytením reálnej komunikácie vybraných aplikácií pri ich bežnom využití. S každou aplikáciou bola vykonaná rada

[4]Apache, Äpache Thrift Tutorial", 2015", https://thrift.apache.org/tutorial.

akcií rozpoznateľnou ich snoopermi, ako napríklad zaslanie chatovej správy. Táto testovacia sada je dostupná na DVD.

Každý zo snooperov následne obsahoval testovací prípad, ktorý inštruoval rekonštrukciu príslušného súboru so zachytenou komunikáciou a vytvorenie výsledných export objektov predstavujúcich zrekonštruované udalosti. Tie sa nakoniec porovnali s očakávaným výsledkom.

Kapitola 5

Záver

Táto práca sa zaoberala rekonštrukciou komunikácie iOS aplikácií. Na jej začiatku bolo vysvetlené, aký druh aplikácií je z pohľadu forenznej analýzy zaujímavý a prečo sú to konkrétne mobilní klienti rôznych sociálnych sietí. Boli vybrané aplikácie Google Hangouts, Twitter a Facebook Messenger.

Nasledujúca kapitola sa venovala analýze komunikácie týchto aplikácií. Všetky z nich využívali na zabezpečenie svojej komunikácie protokol SSL a preto sa prvá časť tejto kapitoly venovala spôsobu jej dešifrovania pomocou využitia SSL proxy s podvrhnutím falošného certifikátu. Následne boli pre každú z aplikácií popísané ďalšie potrebné na to, aby prijali daný falošný SSL certifikát a bolo možné dešifrovať ich komunikáciu, po čom nasledovala analýza protokolov, ktoré využívajú na prenos informácií a popis vybraných zasielaných správ. Boli tu taktiež popísané protokoly SPDY využitý aplikáciou Twitter a MQTT využitý aplikáciou Facebook Messenger.

V poslednej kapitole bol popísaný nástroj Netfox Detective a implementácia tzv. snooperov, modulov umožňujúcich rekonštrukciu aplikačných protokolov. Prvá časť po predstavení tohto nástroja bola venovaná popisu úprav, ktoré v ňom museli prebehnúť, aby bol schopný dešifrovať zachytenú iOS komunikáciu. Následne bola popísaná implementácia každého zo snooperov.

Cieľ tejto práce bol dosiahnutý. Implementované snoopery boli schopné zrekonštruovať vybranú zachytenú komunikáciu aplikácií Google Hangouts, Twitter a Facebook Messenger. Ďalším možným pokračovaním tejto práce je analýza komunikácie spolu s implementáciou snooperov ďalších aplikácií, alebo rozšírenie schopnosti rozpoznávať ďalšie udalosti už existujúcich snooperov.

Literatúra

[1] Apple: About Objective-C. 2014.
URL https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

[2] Banks, A.; Gupta, R.: MQTT Version 3.1.1. 2014.
URL http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[3] Belshe, M.; Peon, R.: SPDY Protocol draft-mbelshe-httpbis-spdy-00. 2012.
URL https://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00

[4] Davidoff, S.; Ham, J.: *Network Forensics: Tracking Hackers through Cyberspace*. Prentice Hall, 2012.

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

[6] 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 Defence, 2015, ISBN 978-80-7231-997-8, s. 147–163.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=10863

Prílohy

Zoznam príloh

Príloha A

Obsah DVD

Priložené DVD obsahuje:

- Text bakalárskej práce vo formáte PDF
- Zdrojové súbory bakalárskej práce pre systém LaTeX
- Zdrojové súbory snooperov
- Sada testovacích dát snooperov