

LINE 即時通訊軟體之通訊協定與安全性分析

王傑民¹ 伍立鈞² 李泓曄³ 吳育松⁴

安全與系統實驗室，國立交通大學資訊工程系

chiehmin18@gmail.com¹ lee987667@hotmail.com² g6_7893000@hotmail.com³

hankwu@g2.nctu.edu.tw⁴

摘要

LINE[5]即時聊天與語音通話軟體是目前國人在智慧型裝置上最常使用的即時通訊 app，在台灣已經累積了超過 1700 萬名使用者，但多數人並不清楚 LINE 背後的運作模式以及潛在的資安問題，所以本研究將針對 LINE 的通訊協定運作以及探討 LINE 的訊息傳遞是否安全?是否有被監聽的可能?等問題來進行研究。透過對 LINE Android App 的反向工程、Line 連線通訊行為的觀察我們發覺 LINE 的傳輸協定是以 Apache Thrift[4]這套框架所架構而成，大部分的訊息傳輸都以 TLS/SSL 進行加密，但是在 3G 行動網路下測試時，語音訊息留言卻是採取未加密的明碼格式傳輸，對於使用上有一定的隱私疑慮，所以我們也透過重建 LINE Apache Thrift 定義檔來實作第三方開源 LINE 客戶端應用程式，希望能提供更為安全的 LINE 客戶端平台。

1. 研究簡介

即時聊天與語音通話軟體 LINE 是目前國人在智慧型裝置上最常使用的即時通訊 App，在台灣已經累積了超過 1700 萬名使用者，甚至還與中華電信合作推出了使用 mPro 月租費[30]的用戶可免費無限使用 LINE 傳送接收文字訊息、照片、影片、聲音訊息、語音通話與貼圖，皆不額外收取費用。儼然取代了傳統的通訊方式與習慣，但多數人並不清楚 LINE 背後的運作模式以及潛在的資安問題，

在本研究中我們針對 LINE 的安全性進行探討，包括測試連線過程是否有漏洞以及被竊聽的可能，並同時針對 LINE 的訊息傳遞次序保證進行分析。我們同時也透過對 LINE Android App 的反向工程來檢測 LINE App 本體的內部設計是否有洩露使用者隱私的嫌疑，並藉且進一步探討其是否有被惡意程式攻擊而造成訊息被監聽、私密資料被竊取的風險。最後基於我們對於其通訊協定的掌握，我們也實作出了一個可與 LINE Server 連線並成功進行訊息收送的 Pidgin plugin[19]，藉此來印證我們的觀察跟發現。

2. 研究動機

LINE 是國人最常使用的智慧裝置通訊軟體，所以我們想要去探究 LINE 背後的訊息與語音通訊協定，研究是否可能有資安上的漏洞，以及如何來去反制惡意程式的攻擊。另外，即便 LINE 為免費的通訊應用程式，但是他的通訊協定並不是公開的，所以使用者只能強迫使用官方提供的應用程式，而官方的應用程式在我們初步的觀察下，其事實上有夾帶數個廣告公司提供的追蹤程式，使用者在使用 LINE 的過程中，也有可能因此將個人的隱私資訊提供給廣告公司，對於注重使用者隱私的使用者恐是一個隱憂。此外 LINE 公司並沒有製作 Linux 平台的客端程式，對於 Linux 的使用者十分不友善，所以我們也希望透過研究 LINE protocol 來做為未來由開源社群通力實作第三方 LINE 客端程式的基礎。

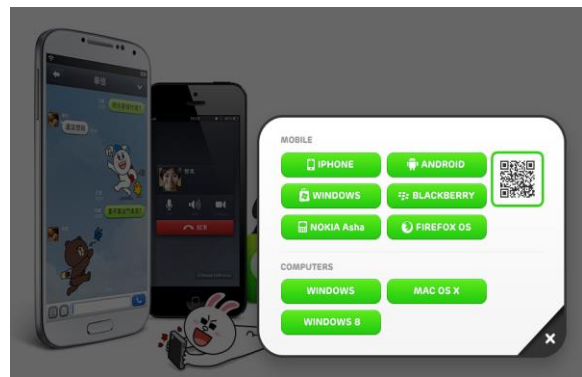


圖 1. LINE 不支援 Linux 平台

3. LINE 訊息傳遞行為以及通訊協定之觀察

LINE 的使用者高達上億，使用者們也相當依賴 LINE 作為日常生活的溝通，所以 LINE 訊息傳遞的可靠性與一致性是一個需要深究重要的課題。我們透過實驗來去觀察 LINE 訊息傳遞的可靠性和一致性，我們想要探討的問題有：(1) LINE 的最大訊息傳遞量、(2) LINE 在快速的訊息傳遞下是否存在訊息漏傳的可能性、(3) LINE 的訊息傳遞在不同的使用者或是 client 上是否具備有相同的次序性(total ordering)[6]。如圖 2 所示，我們觀察到另

一款大家十分常用到的 Facebook Messenger[8]並不具備 total ordering 的特性，所以我們十分好奇 LINE 在這方面是否會提供比 Facebook messenger 更嚴格的訊息遞送次序保證。

LINE 的通訊協定是經由 TLS/SSL protocol[10, 11]進行加密，安全性看似相當的足夠。我們嘗試透過 SSL man-in-the-middle 攻擊來攔截 LINE 傳輸的訊息內容。而在另一方面，我們在研究 LINE protocol 的時候也意外發現，LINE 同時也提供了 HTTP port 未加密連線服務，所以我們也嘗試監聽 LINE 的未加密訊息傳遞，包括透過 Android 基於 Linux 系統的防火牆 iptables[16]設置阻擋 SSL port 443 連線的規則，實測 LINE 是否會在 port 443 受阻擋的情況下改經由 port 80 進行明碼傳輸。接著我們使用封包監聽軟體 tcpdump[2]來監聽 LINE 的通訊連線，並利用 Wireshark[27]進行分析，藉此對 LINE protocol 有一個概觀的認識。

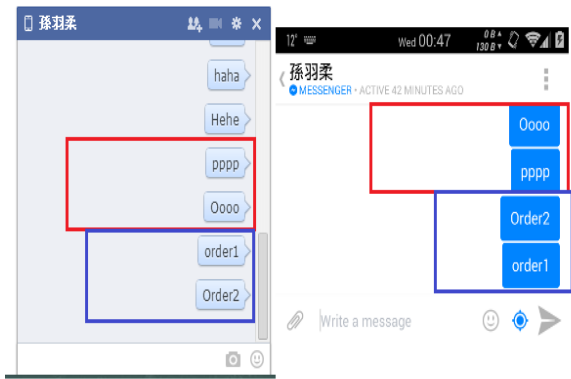


圖 2. Facebook messenger 訊息傳遞並不具備 total ordering 的特性

3.1. LINE 大量訊息傳遞的可靠性與一致性分析

在這個實驗中為了實測極端情況下 LINE 的訊息傳遞能力，我們採用 Android 平台下的 Xposed framework[22] 來做為實驗的輔助工具，Xposed framework 是一套 Android 平台下的 instrumentation 工具，他透過更換 Android 系統下帶起 Android runtime Zygote[20] 的啟動程式 /system/bin/app_process 來達到對 app 內部函式與 class 定義進行 hooking 和 replacement 等做。我們可藉助其在程式執行階段注入我們想要新增的功能的程式碼，或者改變原本程式的功能。

首先如圖 3，我們透過 Xposed framework 對 LINE App 的訊息欄進行 hooking，藉此對訊息欄注入一個有 40,000 個字元的長訊息，並嘗試傳送該訊息來觀測其反應。實驗結果如圖 3 所示，Line App 事實上是有設定一 10,000 字元的訊息字數上限。

```

import static de.robv.android.xposed.XposedBridge.hookAllMethods;

import android.text.SpannableStringBuilder;
import de.robv.android.xposed.IXposedHookLoadPackage;
import de.robv.android.xposed.XC_MethodHook;
import de.robv.android.xposed.XC_MethodReplacement;
import de.robv.android.xposed.callbacks.XC_LoadPackage.LoadPackageParam;

public class Main implements IXposedHookLoadPackage{

    String message;

    @Override
    public void handleLoadPackage(LoadPackageParam lpparam) throws Throwable {

        final String pkg = lpparam.packageName;

        message = "";
        for(int i = 0; i < 10000; i++) {
            message += "hank";
        }

        if(!pkg.equals("jp.naver.line.android")) {
            return;
        }

        final Class<?> editText = findClass("android.widget.EditText", lpparam.classLoader);
        hookAllMethods(editText, "getText", new XC_MethodReplacement() {

            @Override
            protected Object replaceHookedMethod(MethodHookParam param) throws Throwable {

                SpannableStringBuilder ret = new SpannableStringBuilder(message);

                return ret;
            }
        });

        hookAllMethods(editText, "setText", new XC_MethodHook() {

            protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
                param.args[0] = message;
            }
        });
    }
}

```

圖 3. 利用撰寫 Xposed module 對 LINE app 注
入一個長訊息

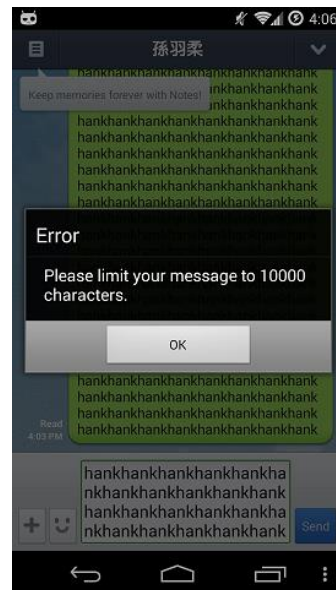


圖 4. LINE app 的訊息傳輸上限為 10000 個字元

接下來我們同樣是利用 Xposed framework 以極高的速度和頻率進行聊天訊息的交換，想要測試 LINE 是否具有掉訊息的隱憂，以及 LINE 的訊息記錄在不同的使用者及客戶端間是否具有 total ordering 的特性。

如圖 5，我們透過撰寫 Xposed module 對 LINE 每秒鐘注入 10 個聊天訊息，持續 10 秒，總共注入 100 個訊息。我們在兩台手機不同帳號下同時觸發該 Xposed module，透過這個方式來去檢驗

第二十四屆全國資訊安全會議(CISC 2014) Cryptology and Information Security Conference 2014

兩邊是否都有確實顯示總共 200 個訊息，且驗證兩邊的訊息傳遞次序是否有保持一致。最後經過此測試方法的檢測之下，我們發現結果如 LINE 的訊息傳遞不但沒有漏接訊息，且其訊息的傳遞是符合 total ordering 的，如圖 6 所示。

```
@Override
public void handleLoadPackage(LoadPackageParam lpparam) throws Throwable {
    final String pkg = lpparam.packageName;
    message = "hank22";
    if(!pkg.equals("jp.naver.line.android")) {
        return;
    }

    final Class<?> editText = findClass("android.widget.EditText", lpparam.classLoader);
    hookAllMethods(editText, "getText", new XC_MethodReplacement() {
        @Override
        protected Object replaceHookedMethod(MethodHookParam param) throws Throwable {
            SpannableStringBuilder ret = new SpannableStringBuilder(message);
            return ret;
        }
    });

    hookAllMethods(editText, "setText", new XC_MethodHook() {
        protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
            param.args[0] = message;
        }
    });

    final Class<?> button = findClass("android.view.View", lpparam.classLoader);
    hookAllMethods(button, "onClick", new XC_MethodHook() {
        @Override
        protected void afterHookedMethod(MethodHookParam param) throws Throwable {
            XposedBridge.log("clicked");
            if(first) {
                first = false;
                Button btn = (Button) param.thisObject;
                for(int i = 0; i < 10; i++) {
                    btn.performClick();
                }
            }
        }
    });
}
```

圖 5 利用撰寫 Xposed module 進行高速、高頻率的訊息傳遞



圖 6 LINE 訊息遞送經過高強度的測試的確具有 total ordering 的特性

3.2. SSL Man-in-the-middle attack 攔截封包訊息實驗

LINE 使用 TLS/SSL 方式加密其應用層封包。對於 TLS/SSL 加密通道內容的竊取，最常見

的攻擊手法為 man-in-middle attack[29]，其原理是在於當攻擊者擁有伺服器端程式的私鑰和憑證，則攻擊者可以架設 proxy 偽裝成伺服器端以欺騙用戶端程式將資料傳送給他，而攻擊者使用伺服器端的私鑰解讀資料後只要再將資料重新加密並傳給真正的伺服器就可以維持兩端點的正常溝通，這樣就可以在不被兩端點發現的情況下盜取的資料。

Man-in-the-middle attack 也被用做網路協定的反向工程。此情況下用戶端多半是可被控制的，攻擊者很有機會在無需伺服器的私鑰、憑證展開攻擊。其背後的關鍵在於讓用戶端程式信任攻擊者 proxy 所產生的公鑰和憑證，並透過其對 proxy 進行連線。

```
0x26aa A gd2.line.naver.jp
response 0x26aa A 203.104.131.5 A 119.235.235.91
0x4665 A tmedia.p2sp.baidu.com
response 0x4665 CNAME tmedia.p2sp.n.shifen.com A
0x63bd A s.p2sp.baidu.com
response 0x63bd CNAME s.p2sp.n.shifen.com A 123.1
0xcff8 A google.com
response 0xcff8 A 140.113.14.59 A 140.113.14.57 A
0xbc35 A openapis.jboard.naver.jp
response 0xbc35 A 111.91.133.91
```

圖 7 查找 LINE 使用的 domain name

首先我們將 LINE 封包導向自己架設的 proxy，我們藉由 Wireshark 找出 LINE 所查找的 domain name，如圖 7，並且藉由修改系統的 hosts 檔中 domain 和 IP 位址的對應，來讓 LINE 傳送資料至我們指定的 IP 位址。

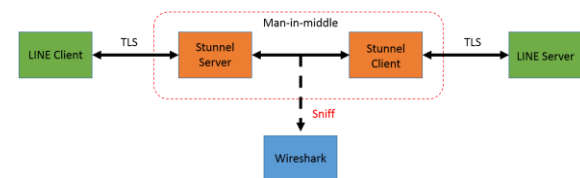


圖 8 Man-in-middle 架構

緊接著，我們使用 stunnel[28]這套軟體來作為 proxy server，stunnel 有 client mode 和 server mode 兩種模式，我們讓 stunnel server 偽裝成 LINE server、stunnel client 偽裝成 LINE client，如圖 8。stunnel 會與 LINE server、LINE client 建立 TLS/SSL tunnel，但他們之間的連線我們設為不加密，所以可以直接透過 Wireshark 來聽取流經的封包。

```
# 192.168.1.102
[stunnel-client]
client = yes
accept = 192.168.1.102:8888
connect = 203.104.131.5:443
```

```
# 192.168.1.3
[stunnel-server]
client = no
accept = 192.168.1.3:443
connect = 192.168.1.102:8888
```


TIMEOUTclose = 0

表格 1 stunnel 設定檔

在 Linux 系統中，若封包的來源、目的只在單一台機器內部傳送，作業系統會在 IP 層就將封包往回傳，這時我們就必須要去監聽的就不是一般的網卡(例如: eth0)，而是改成監聽 lo(lookback)這個裝置，否則 Wireshark 這類 sniffer 軟體就無法聽到該封包，而當初我們為了簡化設置伺服器的步驟，我們選擇將 stunnel 兩端拉開到兩台虛擬機器上。

而為了欺騙用戶端程式我們是合法的伺服器，我們用 OpenSSL[26]做了一個假的根憑證，並將其加入到作業系統的信任憑證列表中，並用這個根憑證將我們 stunnel server 的私鑰做簽名。

我們首先以 https 加密的網頁來做測試，我們使用 python 寫了一個簡單的、支援 https 加密的網頁伺服器，並用另一台電腦的瀏覽器連上這個伺服器，並用 Wireshark 做訊息的擷取，結果是成功的，整個網頁資料的傳送都能透過 Wireshark 觀察到，並且瀏覽器並沒有發出非法憑證警告。

下一步我們就將其套用至 LINE 上，但我們發現同樣的架構下，當 LINE 要做登入動作時，它會立即回報網路錯誤訊息，如圖 9，並中斷登入。為了找出錯誤；我們在各個節點之間都使用 Wireshark 做觀察，發現當 stunnel server 和 LINE client 做完公鑰交換後，LINE client 就會自動將連線斷開，我們猜測 LINE client 有自己的根憑證列表，Firefox、Chrome 中也可以看見相同的做法，由應用程式自己維護根憑證列表可以避免因為其他應用程式匯入惡意憑證到作業系統中而造成的危險。



圖 9 LINE 無法通過 stunnel

3.3. 利用防火牆 iptables 設置阻斷 port 443 連線

在之後的章節 LINE 相容客戶端實作中，我

們對 LINE 訊息處理伺服器利用 Linux 環境下的 nmap 工具進行通訊埠的掃描，圖 10 對 LINE 訊息伺服器進行通訊埠掃描 LINE 結果顯示 LINE 也開啟了 http 通訊埠，並且經過實測後意外發現 LINE 其實也支援經由 http 協定的明碼傳輸，我們立刻聯想到或許在不支援 https 的環境下，LINE 可能會經由明碼進行傳輸，我們在圖 11 利用 iptables 阻擋 Android 系統上 https 連線，檢測 LINE 是否會改使用明碼 http 傳輸，經過此設定後我們發現，LINE 仍可正常登入，但是無法傳送任何文字訊息或語音通話。

```
[fatminmin@fatminmin-desktop]:~$ dig gd2.line.naver.jp
; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Ubuntu-1:9.9.3.dfsq.P2-4ubuntu
;; global options: +cmd
;; Got answer:
;; -->HEADER<-- opcode: QUERY, status: NOERROR, id: 37587
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL:
0
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:;, udp: 512
;; QUESTION SECTION:
;gd2.line.naver.jp.                IN      A
;; ANSWER SECTION:
gd2.line.naver.jp.                2778    IN      A      119.235.235.91
gd2.line.naver.jp.                2778    IN      A      203.104.131.5
;; Query time: 8 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Wed Feb 19 20:38:35 CST 2014
;; MSG SIZE rcvd: 78

[fatminmin@fatminmin-desktop]:~$ nmap 203.104.131.5
Starting Nmap 6.40 ( http://nmap.org ) at 2014-02-19 20:39 CST
Nmap scan report for 203.104.131.5
Host is up (0.042s latency).
Not shown: 964 filtered ports, 32 closed ports
PORT      STATE SERVICE
80/tcp    open  http
443/tcp   open  https
5000/tcp  open  upnp
9418/tcp  open  git
Nmap done: 1 IP address (1 host up) scanned in 4.02 seconds
```

圖 10 對 LINE 訊息伺服器進行通訊埠掃描

```
[fatminmin@fatminmin-desktop]:~$ adb shell
shell@d802:/ $ su
root@d802:/ # iptables -A OUTPUT -p tcp --dport 443 -j DROP
root@d802:/ #
```

圖 11 利用 iptables 阻擋 Android 系統上 https 連線

3.4. 以 tcpdump 觀測 LINE 未加密連線

在 3.3 節中我們利用 iptables 阻斷 port 443 連線試圖讓 LINE 全部改採 port 80 未加密的連線，雖然嘗試最後是失敗的，但 LINE 的確使有透過 port 80 未加密的連線傳定某些類型的資料，而這些資料是可以透過 tcpdump[2](表格 2)來進行觀測的，並進一步透過 Wireshark 做進一步詳細的分析。

tcpdump -vv -s 0 -w /sdcard/LINE.pcap

表格 2 tcpdump 觀測 LINE 連線指令

透過 tcpdump 封包的擷取我們發現，LINE 的動態時報功能如圖 12，其中對於動態時報裡面的圖片傳輸是經由 port 80 以未加密的形式進行傳送的，所以有心人只要透過公開的無線網路就可以輕易的知道你目前動態時報裡的所有照片，圖 13 為

第二十四屆全國資訊安全會議(CISC 2014) Cryptology and Information Security Conference 2014
我們所攔截到的 timeline 裡的圖片。



圖 12 LINE 的動態時報

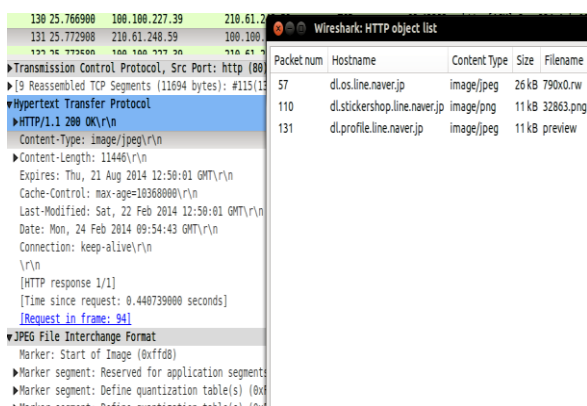


圖 13 LINE 動態時報好友或自己所張貼的圖片
皆為明碼傳輸

3.5. 以 tcpdump 觀測 LINE 3G 連線

接著我們嘗試將連結網路方式由 Wi-Fi 改成使用 3G 行動網路連結，檢測訊息的傳遞是否會因此有所改變，我們意外的發現，在使用 Wi-Fi 連線的情況下，語音留言訊息是以加密的形式傳遞，但若是 3G 行動網路的情況下，該訊息留言竟然是以 HTTP 明碼傳輸，並且能夠輕易地還原成原來的音檔，如圖 13 所示，LINE 在使用 3G 行動網路的情況下語音留言訊息是未加密的。或許在 3G 網路下使用未加密連線能夠降低伺服器的負擔(3G 連線於底層已有提供加密)，但是用戶的隱私卻也因此沒有完全地被保護。倘若 3G 網路到 LINE 伺服器間的網路連線被有心人士所掌控，使用者的通訊是能夠輕易地被竊取的。

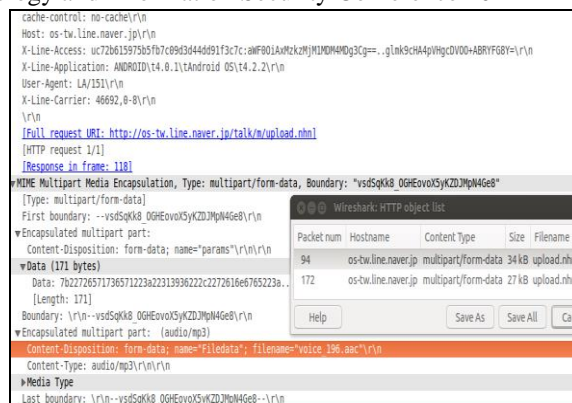


圖 14 LINE 在 3G 行動網路模式下的語音留言
訊息是未加密的

4. LINE Android App 反向工程

Android 平台上的 App 運行於 Dalvik virtual machine[3]之上，一般 App 是使用 Java 語言編寫，並編譯成 Dalvik executable format(dex)的 byte code 形式。由於 Java 語言本身有 reflection 的功能，所以使得 Android 平台上的反向工程相當的容易，大多數的開發者因此都會使用 ProGuard[1]工具進行程式碼的混淆工作，甚至將程式的重要功能以 native code 的形式撰寫來提升反向工程的難度。

LINE Android App 的程式中也同樣利用上述的兩個方式對於程式碼進行保護，但我們仍然能從中分析出 LINE 程式的大致架構圖以及功能區塊，我們利用了不少工具來進行反向工程，包括了(1) apktool[24]將 app 反組譯成 smali 組語格式以及 xml 檔案的還原、(2) dex2jar[25]將 Dalvik executable format 重新還原成 Java 的 jar 格式、(3) jd-gui[12]對 jar 內包含的 class 檔案反編譯成原始的 Java 原始碼(雖然反編譯出來的程式碼不一定正確，但可以給予我們粗略的程式大綱)、(4) IDA [9]是一款支援多種指令集的反組譯工具，我們利用他來將 LINE App 中 ARM 格式的 native code 函式庫進行反組譯，並透過 IDA 的程式流程分析模式來觀測 LINE 的程式流程。

4.1. LINE Android App 程式架構分析

在 LINE Android app 的最外層是由 Java 語言所撰寫而成，負責處理使用者與介面的互動和一些 Android 系統的基本功能如相機、GPS 定位、網路狀態檢查等，在這個部分 LINE 的開發商有效的將程式碼利用 ProGuard 混淆的很徹底，但有趣的地方是 LINE 也引入了不少第三方函式庫，這些函式庫的引入方式並不像一般的 Android 程式撰寫方式是將函式庫的 jar 與 app 一起建置，而是將這些函式庫預先編譯成 dex 格式放置在 asset 資料夾下，再透過特殊的方式進行呼叫，而這些第三方函式庫

第二十四屆全國資訊安全會議(CISC 2014) Cryptology and Information Security Conference 2014 完全沒有經過混淆處理，所以惡意軟體可能較容易經由分析這部分程式碼找出潛在漏洞進行攻擊，這些函式庫包含以下幾項：

- dex-alm1-1.0.0.jar: 與日本電信公司 KDDI Corporation[13]服務連結的函式庫，驗證使用者是否為該公司的用戶，提供特定的服務。
- dex-mangoplayer-1.0.0.jar: 由韓國軟體公司 DreamToBe, Inc[7]所開發的 Android 影音撥放程式，作為 LINE 的影像、聲音撥放器。
- dex-location-1.0.0.jar: 包含多個 Google Mobile Server 的函式庫，包括 Google Cloud Messaging(GCM)、Tracking 等函式庫。
- dex-snsauth-1.0.0.jar: LINE 與社群網路連結的函式庫，支援的社群網路包括 Facebook、微博、Feinno(北京新傳媒信科技有限公司)、人人網等社群網路。
- dex-traces-1.1.0.jar: 多個廣告公司的追蹤器，包括了 InMobi、IGAWorks、Mobileapptacker 等，用以分析使用者習慣以及位置等資訊。
- dex-zxing-1.0.0.jar: Zxing(Zebra Crossing)[17]為 Android 平台上知名的 Open Source QR code 產生器及讀取器，該函式庫以 Apache License 2.0 釋出，所以使用者可自由使用且不須公開原始碼。
- dex-ampkit-1.0.1.jar: LINE 用來和 native code library 連結的介面，LINE 將所有訊息傳輸及語音通話的功能寫成 native code 保護起來，防止反向工程，而 dex-ampkit 即是 LINE 用來和該功能溝通的介面函式庫。

而最底端的 libamp 函式庫則是 LINE 的主要核心功能，包含了文字聊天、語音通話與貼圖購買皆全部必須透過 native code library 來進行溝通，讓反向工程的難度提高，保護 LINE 的核心功能。LINE 的程式架構經整理如圖 15 LINE 程式架構圖所示。

此外我們也嘗試找出在 3.2 中 SSL man-in-the-middle attack 失敗的原因。在 Android 的反向工程中，我們發現到 LINE Android app 在進行 SSL 連線時會先將 LINE Android app 中的 /res/raw/cacerts.bks 的認證檔加入其 TrustedStore 中，所以若是能夠將我們使用的 stunnel server 憑證加入其中，我們的 SSL Man-in-the-middle attack 攔截封包訊息實驗應該就能成功了。



圖 15 LINE 程式架構圖

4.2. LINE Android App 使用者隱私疑慮

在 4.1 節 Line Android App 程式架構分析中提到 LINE 的函式庫中包含了 dex-traces-1.1.0.jar 此一函式庫，在這函式庫中引入了多個廣告公司的追蹤器，可用於分析 LINE 使用者的使用情況，但並不是每個使用者都能夠接受自己的使用習慣被收集，除此之外，LINE 本身所要求的使用者權限也相當的多如表格 3 所示，包刮了蒐集使用者的地理位置 approximate location(network-based)、precise location(GPS and network-based)，還有其他 app 的使用資訊 retrieve running apps 等，這些對於一個即時聊天軟體並非一個絕對必要的選項，LINE 中也沒有選項可以關閉 LINE 對於使用者地理位置的獲取、或是對於廣告公司追蹤器的使用，若這些資訊也透過追蹤器傳回廣告公司，對於使用者的隱私保護有相當程度的隱憂，也不是使用者所樂見的。這些隱私問題的解決辦法僅能透過市面上的 App 權限管理程式如 Xprivacy[15]、LBE privacy manager[21]或者 Android 尚未提供 App Ops[14]來限制個別 App 對於不同權限的存取，例如禁止 LINE 具有存取使用者地理位置的權限等手段。但

這些權限管理程式大多需要 root 權限或者是使用修改過的 Android 系統，對於一般使用者難度較高且不友善，所以這更讓我們提高了對第三方 Open Source 不含追蹤器的 LINE app 的需求，所以我們會在 LINE 相容客戶端實作的章節中，實作出一個第三方 LINE client。

Permissions

Development tools

test access to protected storage

Your personal information

read your contacts, read call log

Services that cost you money

directly call phone numbers

Your location

approximate (network-based) location, precise

(GPS) location, access extra location provider

commands

Your messages

receive text messages (SMS)

Network communication

full network access, view network connections,

view Wi-Fi connections, receive data from Internet,

Google Play billing service, pair with Bluetooth

devices

Your accounts

find accounts on the device

Storage

modify or delete the contents of your USB storage

Phone calls

read phone status and identity

Hardware controls

take pictures and videos, control vibration, record

audio, change your audio settings

System tools

run at startup, close other apps, prevent phone from

sleeping, send sticky broadcast, change network

connectivity, connect and disconnect from Wi-Fi,

disable your screen lock, retrieve running apps,

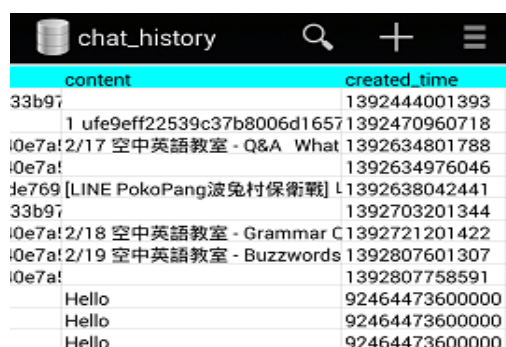
modify system settings, install shortcuts

表格 3 LINE 的權限宣告

4.3 Android 上可能存在的惡意程式攻擊手法

除了前面幾個章節的實驗和所發現的未加密連線可能遭到監聽，我們還發現了一些可能的入侵手法，透過 SQLite Editor[23]這個 Android 上的資

料庫編輯工具可以發現，LINE app 的聊天訊息、歷史紀錄以及程式設定等，都是以明碼的格式儲存於資料庫中，如圖 16 LINE 的資料庫內容是以明碼儲存所示，在具有 root 權限的 Android 裝置中，惡意程式即可透過 root 權限直接讀取 LINE 資料庫的內容。即便在不具有 root 權限的裝置上，仍然可以透過 Android 提供給開發者的工具 adb tool 執行 adb backup 來將 app 的隱私資料取出，所以可以透過撰寫桌面個人電腦上的木馬程式，在使用者將 USB 傳輸線連結電腦的時候執行 adb 命令來將該資料庫檔案取出，回傳給駭客，洩漏使用者的隱私。



content	created_time
33b97	1392444001393
1 ufe9eff22539c37b8006d1657	1392470960718
!0e7a!2/17 空中英語教室 - Q&A What	1392634801788
!0e7a!	1392634976046
!e769 [LINE PokoPang波兔村保衛戰] L	1392638042441
33b97	1392703201344
!0e7a!2/18 空中英語教室 - Grammar C	1392721201422
!0e7a!2/19 空中英語教室 - Buzzwords	1392807601307
!0e7a!	1392807758591
Hello	92464473600000
Hello	92464473600000
Hello	92464473600000

圖 16 LINE 的資料庫內容是以明碼儲存

5. LINE 相容客戶端實作

在對 LINE 進行反向工程的研究中，我們發現 LINE 服務的底層通訊協定是透過 Apache Thrift[4]來完成的。Apache Thrift 是由 Facebook 所開發並提交給 Apache 基金會成為開源項目，Apache Thrift 的目的是希望解決跨平台、跨程式語言間訊息傳遞與溝通的困難，使用者能夠透過 Apache Thrift 的協助之下，快速建立服務伺服器與客戶端，而不需要煩惱傳輸過程中的通訊協定以及底層繁瑣的實作。Apache Thrift 的操作模式只要由使用者撰寫介面函示檔，Apache Thrift 便能如圖 17 Apache Thrift 架構所描繪的自動產生 Remote Procedure Call(RPC)的中間傳輸程式碼，透過 Apache Thrift 的協助客戶端只需要呼叫定義好的函式，便能與服務伺服器端做連結，所以使用者就可以專心編寫程式的主要功能而不需要煩惱底層的訊息傳輸。目前市面上有許多知名的軟體公司便是採用 Apache Thrift 來架設自己的系統服務，包括了 Facebook、Twitter、Quora、Evernote 等大公司，當然也包含我們的研究主題 LINE。

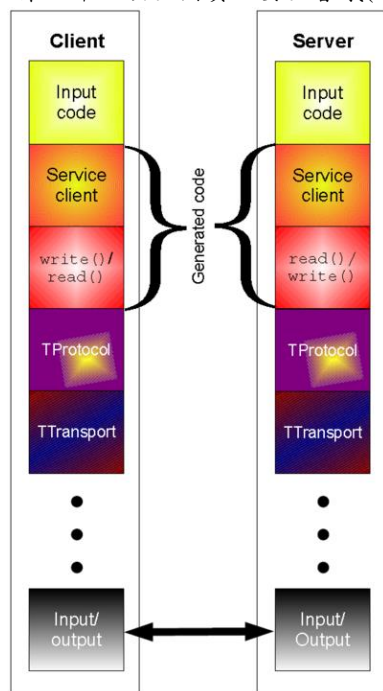


圖 17 Apache Thrift 架構

由於LINE protocol是經由Apache Thrift定義產生的，所以我們只要知道如同表格 4 Apache Thrift 範例介面定義檔的LINE的介面定義，便可以透過Apache Thrift compiler重新編譯、產生LINE的底層通訊協定。

```
typedef i64 long
typedef i32 int
service ArithmeticService { // defines simple
    arithmetic service
    long add(1:int num1, 2:int num2),
    long multiply(1:int num1, 2:int num2),
}
```

表格 4 Apache Thrift 範例介面定義檔

5.1. LINE Apache Thrift Interface Definition

儘管LINE將所有的通訊協定部分以native code撰寫直接編譯成machine code，但是透過反向工程，我們仍然可以通過LINE內部Java程式碼呼叫native的介面函式來去回述，了解LINE底層通訊的基本介面定義與資料型態，重新寫成一個Apache Thrift的介面定義檔，即可在Apache Thrift compiler的協助之下，產生客戶端的通訊協定，便可以透過這個方法和LINE服務端的API進行溝通且順利傳送訊息。

圖 18 透過反向工程了解LINE底層通訊函式定義是對LINE進行反向工程後萃取出來的一部分程式碼，我們首先追蹤與底層native code溝通的介面函式，從該介面函式反追回上層即可發現，LINE透過範例中的a函式首先傳入了底層native函式的名稱，接下來將各項參數放入

localdez物件中，即可呼叫底層native函式，處理底層通訊協定傳輸的部分。透過這種追蹤方式我們就可以完全的掌握LINE底層所有重要函式，以及相關參數和回傳值，並如同表格 5 經由LINE反向工程所重建的部分Apache Thrift介面定義檔，重新產生Apache Thrift介面定義檔中，並經由Apache Thrift compiler編譯該定義檔，產生客戶端的連結函式定義，如此一來便可與LINE service API進行溝通。

```
public final List a(String paramString1, String paramString2, long paramLong1, long paramLong2)
{
    dez localdez = new dez();
    localdez.a(paramString1);
    localdez.b(paramString2);
    localdez.a(paramLong1);
    localdez.b(paramLong2);
    a("getMessagesBySequenceNumber", localdez);
    dfb localdfb = new dfb();
    a(localdfb, "getMessagesBySequenceNumber");
    if (localdfb.a())
        return localdfb.a;
    if (localdfb.b != null)
        throw localdfb.b;
    throw new dur(5, "getMessagesBySequenceNumber failed: unknown result");
}

public final void a()
{
    a("unregisterUserAndDevice", new dnl());
}

public final void a(int paramInt, csk paramcsk)
{
    dnt localdnt = new dnt();
    localdnt.a(paramInt);
    localdnt.a(paramcsk);
    a("updateGroup", localdnt);
}

public final void a(int paramInt, csu paramcsu, String paramString, boolean paramBoolean)
{
    dmj localdmj = new dmj();
    localdmj.a(paramInt);
    localdmj.a(paramcsu);
    localdmj.a(paramString);
    localdmj.a(paramBoolean);
    a("setNotificationsEnabled", localdmj);
}

public final void a(int paramInt, csx paramcsx)
{
    dmb localdmb = new dmb();
    localdmb.a(paramInt);
    localdmb.a(paramcsx);
    a("sendMessage", localdmb);
}
```

圖 18 透過反向工程了解LINE底層通訊函式定義

```
struct UserAuthStatus {
    1: bool phoneNumberRegistered;
    2: list<SnsIdType> registeredSnsIdTypes;
}

service LineTalk {
    void acceptGroupInvitation(
        1: i32 reqSeq,
        2: string groupId) throws (1:
TalkException e);

    void acceptProximityMatches(
        2: string sessionId,
        3: list<string> ids) throws (1:
TalkException e);

    list<string> acquireCallRoute(
        2: string to) throws (1: TalkException e);
}
```


5.2. LINE Pidgin plugin 實作

Pidgin 是一款跨平台的即時聊天與語音通話軟體，經由 GNU General Public License 發布，他能夠讓使用者透過這款軟體同時登入多個不同即時聊天帳號 ex: Facebook messenger、IRQ、ICQ、以前的 MSN、Yahoo messenger 等，使用者也可以透過第三方插件來擴充 Pidgin 的功能，而我們的目標便是透過撰寫 Pidgin 插件來達到與 LINE 溝通、傳送即時訊息的能力。

Pidgin 的底層是 libpurple[18]，libpurple 提供了 Pidgin 的所有核心功能，其實 Pidgin 僅僅是 libpurple 的前端介面，所以我們真正要實作的是如圖 19 Pidgin 程式架構的核心 libpurple 的 protocol plugin 功能。

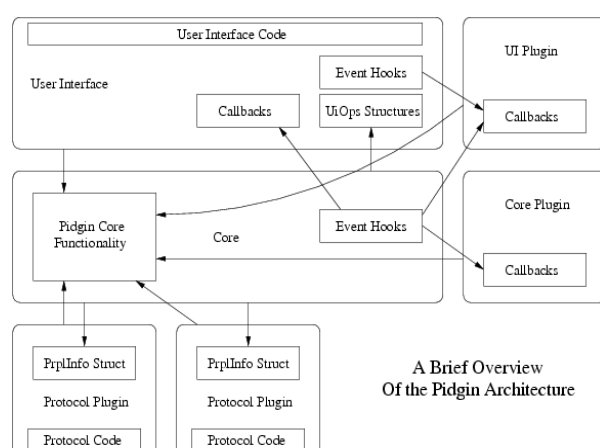


圖 19 Pidgin 程式架構

由於前一個章節已經將底層傳輸協定的程式碼透過 Apache Thrift 所產生出來了，所以我們所需要做的僅僅是與 Plugin 做一個整合，並且注意連線上的細節，便可將 LINE protocol 完整的重現。

經過一連串的測試後我們將撰寫好的 Pidgin LINE plugin 安裝在 Pidgin 上，首先如圖 20 在 Pidgin 下登入 LINE 帳號，並嘗試傳送訊息給自己，有在使用 LINE 的使用者可以發現，LINE 在正常情況下是不允許自己發送訊息給自己的，所以我們嘗試透過我們的 plugin 看看能不能突破這個限制，實測結果如圖 21 所示。目前的雛形並無法新增自己成為好友(會跳出系統警示)，但對於與其他使用者聊天(圖 23)、接受訊息等功能則可運作正常(圖 24)，由此可印證我們對於 LINE 的通訊協定之掌握。

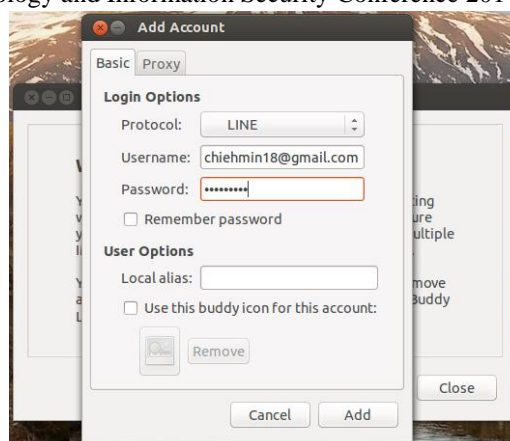


圖 20 在 Pidgin 下登入 LINE 帳號

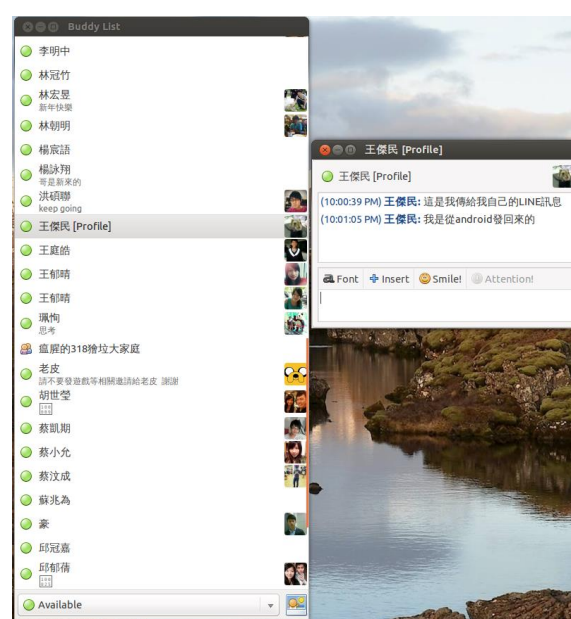


圖 21 利用我們所撰寫的 plugin 發送訊息給自己

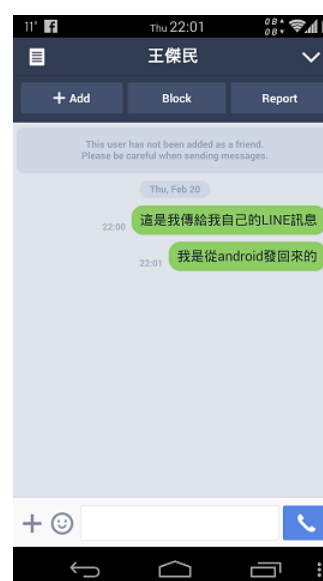


圖 22 在 LINE Android app 上收到相同的訊息

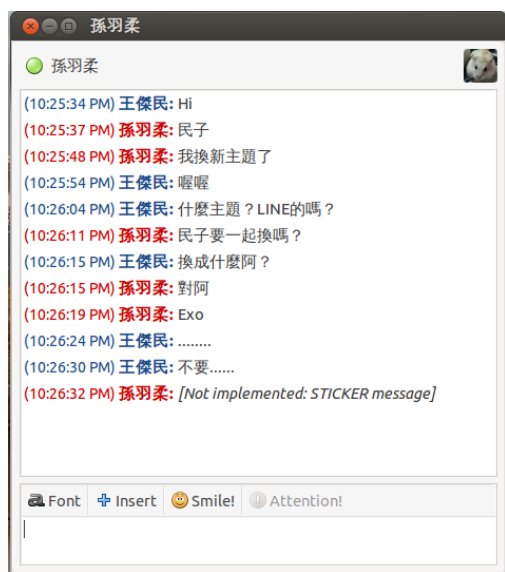


圖 23 利用 Pidgin 與其他 LINE 使用者聊天



圖 24 其他 LINE 使用者也能夠正確接收訊息

6. 結論

總體而言，LINE 在通訊層面的設計跟實作上算是相當嚴謹的。其大部分的訊息皆有加密。對於憑證也有充分地進行驗證來抵抗 SSL man-in-the-middle 攻擊。目前若要竊取 LINE 訊息比較可行的做法是透過竄改 LINE 應用程式本體(如做一個惡意程式版的 LINE App 等)。但 LINE 對於動態時報的功能，其中圖片的傳輸經過我們的實測仍是以未加密的方式進行傳輸，且在行動網路

下語音訊息的傳遞也是沒有加密的，這是我們所發現的 LINE 的兩大隱私漏洞。此外 LINE 所包含的廣告公司追蹤程式若被啟動將有可能造成使用者隱私的侵犯並且佔用使用者不必要的網路流量。

LINE 並沒有提供個人電腦上 Linux 環境下的客戶端程式，對於 Linux 使用者有點不便。在此需求下催生了我們自製第三方 Open Source LINE App 以及跨平台應用程式的想法，首要課題便是研究 LINE protocol。LINE protocol 為閉源，我們透過反向工程得知其傳輸協定是藉由 Apache Thrift，然後藉由找出 LINE 經由 Thrift 所定義的函式宣告，我們可以再次利用 Thrift compiler 重新產生出 client 端的程式碼以便我們的第三方應用程式連結 LINE 官方 API，最後透過實作開源即時聊天軟體 Pidgin 平台上的插件，我們成功的實作出一第三方的開源 LINE 聊天軟體作為本研究的論證。

希望透過本研究能讓國人對於國內最知名的通訊軟體 LINE 的內部通訊協定以及程式架構能有更進一步的認識，並且希望基於本研究於未來能有具備更完善隱私保護的第三方開源 LINE 應用程式可供大眾使用。

參考文獻

- [1] Android Developers. "ProGuard," <http://developer.android.com/tools/help/proguard.html>.
- [2] Android Open Source Project. "Using tcpdump on Android," <http://www.kandroid.org/online-pdk/guide/tcpdump.html>.
- [3] Android Open Source Project. "Dalvik Technical Information," <http://source.android.com/devices/tech/dalvik/>.
- [4] Apache Software Foundation. "Apache Thrift," <http://Thrift.apache.org/>.
- [5] Line Corporation. "Line Introduction," <http://line.me/zh-hant/>.
- [6] Xavier Défago, André Schiper, and Péter Urbán, "Total order broadcast and multicast algorithms: Taxonomy and survey," *ACM Computing Surveys (CSUR)*, vol. 36, no. 4, pp. 372-421, 2004.
- [7] DreamToBe Inc. <http://www.dreamtobe.net/>.
- [8] Facebook. "Facebook messenger," <http://www.facebook.com/mobile/messenger>.
- [9] Hex-Rays. "IDA," <https://www.hex-rays.com/products/ida/>.
- [10] IETF. "The Secure Sockets Layer (SSL) Protocol," <http://tools.ietf.org/html/rfc6101>.
- [11] IETF. "The Transport Layer Security (TLS) Protocol," <http://tools.ietf.org/html/rfc5246>.
- [12] Java Decompiler. "JD-GUI,"

- <http://jd.benow.ca/>.
- [13] KDDI Corporation. <http://www.kddi.com/>.
- [14] Joe Levi. "What is App Ops, and why did Google remove it from Android?," <http://pocketnow.com/2013/12/17/app-ops>.
- [15] M66B. "Xprivacy," <https://github.com/M66B/XPrivacy>.
- [16] Netfilter Core Team. "What is iptables?," <http://www.netfilter.org/projects/iptables/index.html>.
- [17] Sean Owen. "ZXing ("Zebra Crossing")," <https://github.com/zxing/zxing>.
- [18] Pidgin. "What is libpurple?," <http://developer.pidgin.im/wiki/WhatIsLibpurple>.
- [19] Pidgin Developers. "Pidgin plugin," <http://developer.pidgin.im/wiki/ThirdPartyPlugins>.
- [20] Jorge Suarez Rivaya. "Zygote," <http://anatomyofandroid.com/2013/10/15/zygote/>.
- [21] Matthew Rogers. "LBE Privacy Guard," <http://lifehacker.com/5807797/lbe-privacy-guard-monitors-and-controls-what-permissions-your-android-apps-have>.
- [22] Rovo89. "How Xposed works," <http://github.com/rovo89/XposedBridge/wiki/Development-tutorial#wiki-how-xposed-works>.
- [23] Speed Software. "SQLite Editor," https://play.google.com/store/apps/details?id=com.speedsoftware.sqleditor&hl=zh_TW.
- [24] The Apktool Project. "Apktool," <https://code.google.com/p/android-apktool/>.
- [25] The dex2jar Project. "dex2jar," <https://code.google.com/p/dex2jar/>.
- [26] The OpenSSL Project. "OpenSSL," <https://www.openssl.org/>.
- [27] The Wireshark Team. "Wireshark," <http://www.wireshark.org/>.
- [28] Michal Trojnara. "stunnel," <https://www.stunnel.org/index.html>.
- [29] Wikipidia. "Man-in-the-middle attack," http://en.wikipedia.org/wiki/Man-in-the-middle_attack.
- [30] 中華電信. "mPro 全系列客戶可免費無限使用 LINE 傳送接收文字訊息、照片、影片、聲音訊息、語音通話與貼圖," http://mpro.emome.net/OTT/OTT_LINE.html.