THREAT RESEARCH

The Latest Android Overlay Malware Spreading via SMS Phishing in Europe

WU ZHOU, LINHAI SONG, JENS MONRAD, JUNYUAN ZENG, JIMMY SU

JUN 28, 2016 | 14 MINS READ

#MALWARE

Introduction

In April 2016, while investigating a Smishing campaign dubbed RuMMS that involved the targeting of Android users in Russia, we also noticed three similar Smishing campaigns reportedly spreading in Denmark (February 2016), in Italy (February 2016), and in both Denmark and Italy (April 2016).

Unlike the RuMMS campaign, these three campaigns in Europe used view overlay techniques (the same technique we described being used by SlemBunk malware) to present nearly identical credential input UIs as seen in benign apps, subsequently tricking unwary users into providing their banking credentials.

Figure 1 shows the process of how these overlay malware spread via Smishing and infect Android users.

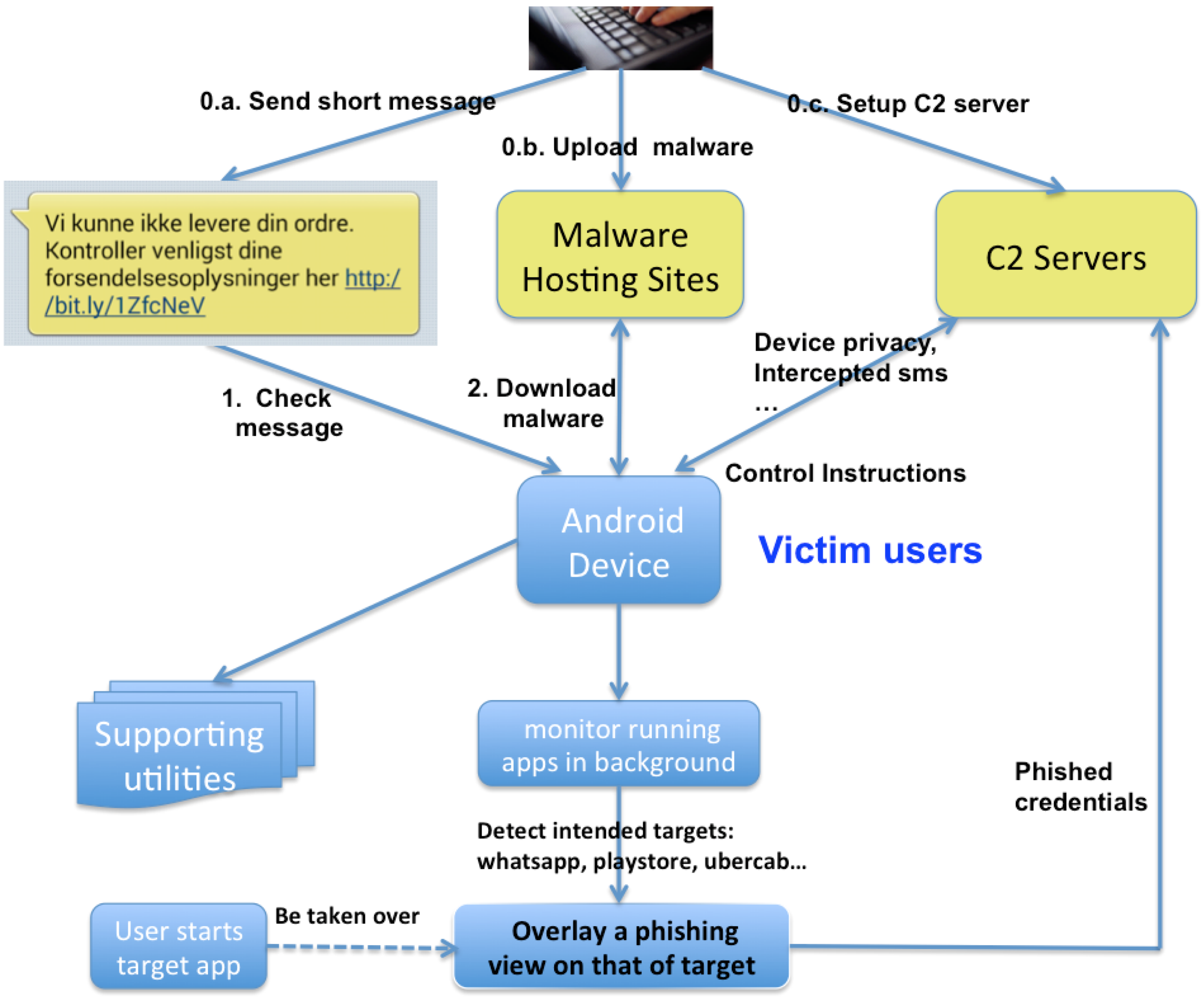

0.a. Send short message
0.b. Upload malware
0.c. Setup C2 server
Vi kunne ikke levere din ordre. Kontroller venligst dine forsendelsesoplysninger her http://bit.ly/1ZfcNeV
Malware Hosting Sites
C2 Servers
1. Check message
2. Download malware
Device privacy, Intercepted sms …
Control Instructions
Android Device
Victim users
Supporting utilities
monitor running apps in background
Phished credentials
Detect intended targets: whatsapp, playstore, ubercab...
User starts target app
Be taken over
Overlay a phishing view on that of target

Figure 1: Overview

Threat actors typically first setup the command and control (C2) servers and malware hosting sites, then put the malware apps on the hosting sites and send victims SMS messages with an embedded link that leads to the malware app. After landing on the user's device, the malware launches a process to monitor which app is running in the foreground on the compromised device. When the user launches a benign app into the foreground that the malware is programmed to target (such as a banking app), the malware overlays a phishing view on top of the benign app. The unwary user, assuming that they are using the benign app, will enter the required account credentials, which are then sent to remote C2 servers controlled by threat actors.

Through our close monitoring of overlay malware spreading via Smishing messages, we recently observed that these types of attacks did not stop despite publicity from security researchers. Instead, our systematic study revealed some interesting and simultaneously worrying findings:

- From February 2016 to June 2016, we observed 55 malicious binaries used in a series of communication protocol. Besides the three publicly disclosed campaigns in **Denmark** and **Italy**, we observed the same threats targeting **Germany** in March 2016 and **Austria** from April 2016 to May 2016. In June 2016, we still see new samples emerging and being used to target users in Denmark; a few other European countries could be impacted as well.

- The key functions of these samples have been the same; however, over time, we noticed that the samples keep evolving in a few different directions. For example, later campaigns usually **targeted more benign apps** than earlier campaigns, focusing on messaging apps, for example, as opposed to banking apps. Also, the malicious apps used in later campaigns are often harder to analyze because **obfuscation techniques** were adopted to evade detection. In addition, some new functionality was added; in particular, we noticed that more recent samples leveraged reflection to bypass the SMS writing restriction enforced by the App Ops service (introduced in Android 4.3). All of this suggests that threat actors are **actively improving their code**.

- Unlike the RuMMS campaign, which mainly used shared hosting services to distribute the malware, the Europe Smishing campaigns show **more diversity** in the associated infrastructure, including the use of self-registered domains, compromised websites, and URL shortening services. Since February 2016, we observed that 27 **Bit.ly** links have been used. In June 2016, we noticed that another three URL shorteners, including **tr.im**, **jar.mar** and **is.gd,** were adopted in the latest campaign. This suggests that threat actors are trying to **diversify the URL shorteners to avoid detection.**

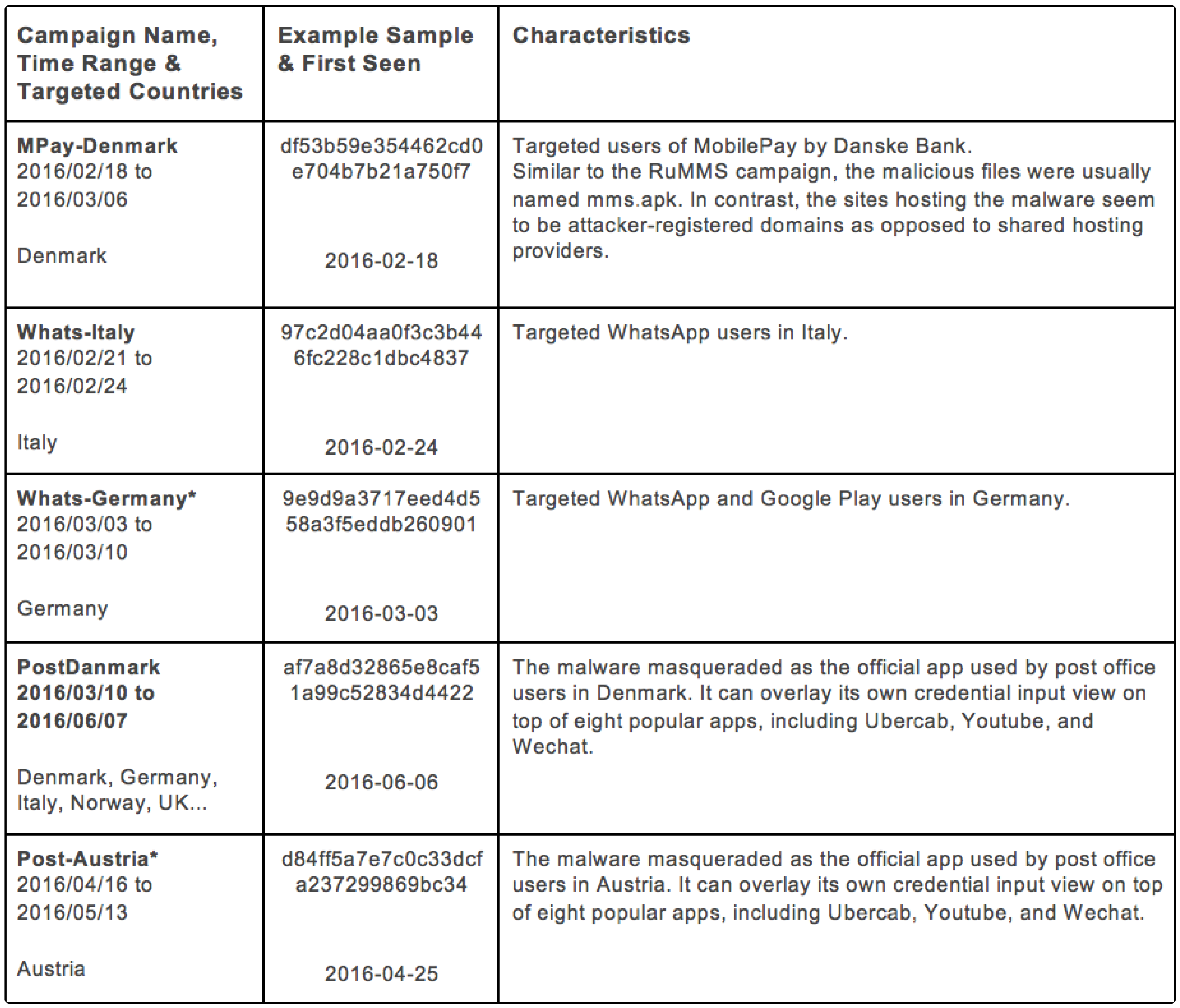
- In total, we identified **12 C2 servers** hosted in **five different countries** that were involved in these campaigns. Among them, the IP address 85.93.5.109 has been used by 24 malicious apps in two campaigns and 85.93.5.139 has been used by eight malicious apps. We also observed that **four C2 servers are within the same 85.93.5.0/24 network segment**. All this suggests that the threat actors have **control over considerable network resources**.

- URL shortening services usually provide **link analytics services**, which enables us to collect data on how many users (from which countries) clicked particular short links and when it happened. Using these services, we found there have been at least **161,349** clicks on the 30 short links redirecting to the overlay malware, each of which can lead to the infection of one Android device. The date information indicated that **most of the clicks occurred in the first few days after the links were created**.

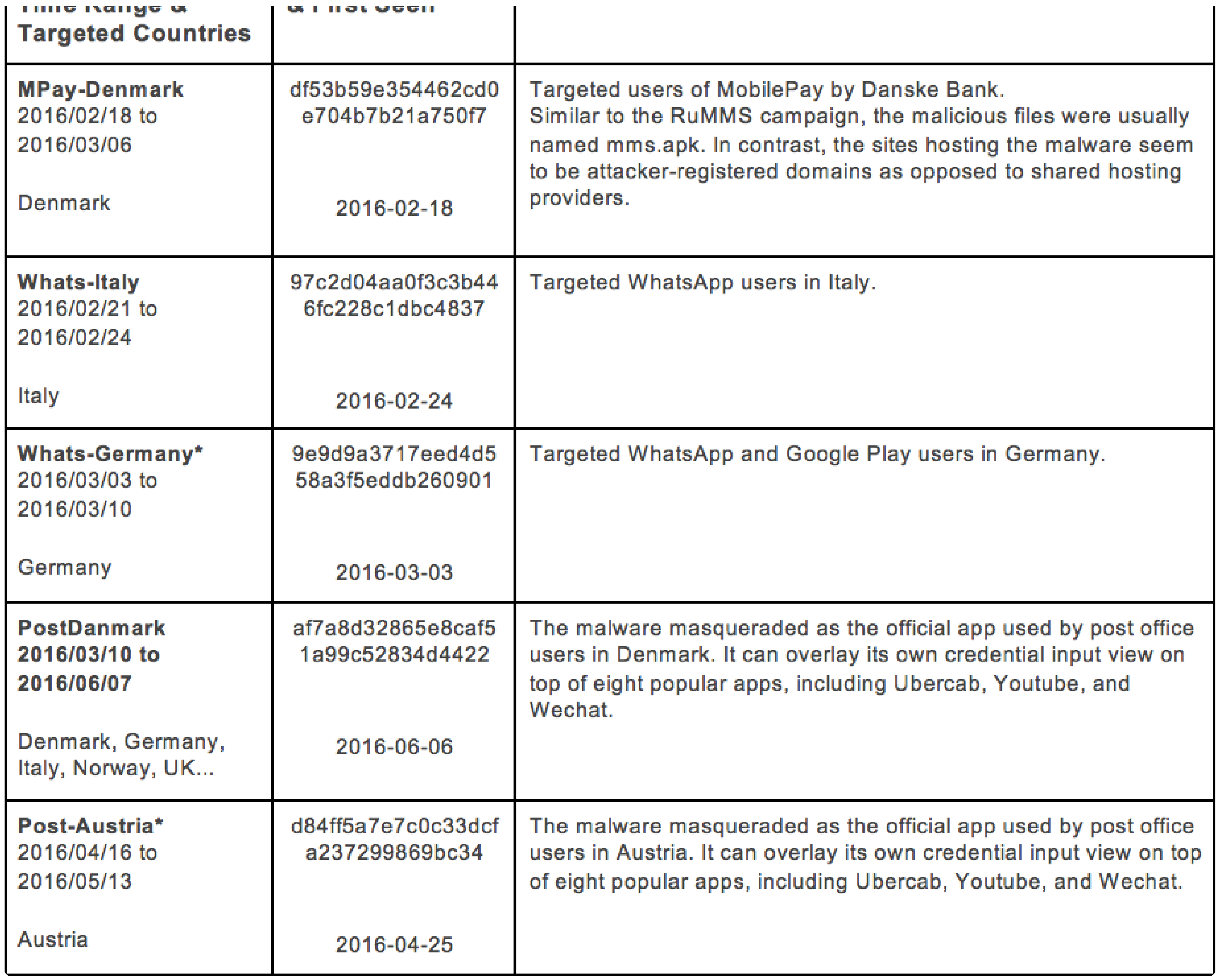
Five Europe Smishing Campaigns

From February 2016 to April 2016, security researchers reported on three campaigns involving example SMS message in the latest campaign is shown in Figure 1. The message roughly translates to, “We could not deliver your order. Please check your shipping information here hxxp://bit[.]ly/1ZfcNeV”. Users in Denmark and Italy were reported to be the primary targets of these three campaigns.

Our recent investigation revealed that these activities keep developing, with other European countries, including **Germany** and **Austria**, being impacted as well. We group these activities into five campaigns, as shown in Table 1.



Campaign Name, Time Range & Targeted Countries	Example Sample & First Seen	Characteristics
MPay-Denmark 2016/02/18 to 2016/03/06 Denmark	df53b59e354462cd0e704b7b21a750f7 2016-02-18	Targeted users of MobilePay by Danske Bank. Similar to the RuMMS campaign, the malicious files were usually named mms.apk. In contrast, the sites hosting the malware seem to be attacker-registered domains as opposed to shared hosting providers.
Whats-Italy 2016/02/21 to 2016/02/24 Italy	97c2d04aa0f3c3b446fc228c1dbc4837 2016-02-24	Targeted WhatsApp users in Italy.
Whats-Germany* 2016/03/03 to 2016/03/10 Germany	9e9d9a3717eed4d558a3f5eddb260901 2016-03-03	Targeted WhatsApp and Google Play users in Germany.
PostDanmark 2016/03/10 to 2016/06/07 Denmark, Germany, Italy, Norway, UK...	af7a8d32865e8caf51a99c52834d4422 2016-06-06	The malware masqueraded as the official app used by post office users in Denmark. It can overlay its own credential input view on top of eight popular apps, including Ubercab, Youtube, and Wechat.
Post-Austria* 2016/04/16 to 2016/05/13 Austria	d84ff5a7e7c0c33dcfa237299869bc34 2016-04-25	The malware masqueraded as the official app used by post office users in Austria. It can overlay its own credential input view on top of eight popular apps, including Ubercab, Youtube, and Wechat.



Time Range & Targeted Countries	& First Seen	
MPay-Denmark 2016/02/18 to 2016/03/06 Denmark	df53b59e354462cd0 e704b7b21a750f7 2016-02-18	Targeted users of MobilePay by Danske Bank. Similar to the RuMMS campaign, the malicious files were usually named mms.apk. In contrast, the sites hosting the malware seem to be attacker-registered domains as opposed to shared hosting providers.
Whats-Italy 2016/02/21 to 2016/02/24 Italy	97c2d04aa0f3c3b44 6fc228c1dbc4837 2016-02-24	Targeted WhatsApp users in Italy.
Whats-Germany* 2016/03/03 to 2016/03/10 Germany	9e9d9a3717eed4d5 58a3f5eddb260901 2016-03-03	Targeted WhatsApp and Google Play users in Germany.
PostDanmark 2016/03/10 to 2016/06/07 Denmark, Germany, Italy, Norway, UK...	af7a8d32865e8caf5 1a99c52834d4422 2016-06-06	The malware masqueraded as the official app used by post office users in Denmark. It can overlay its own credential input view on top of eight popular apps, including Ubercab, Youtube, and Wechat.
Post-Austria* 2016/04/16 to 2016/05/13 Austria	d84ff5a7e7c0c33dcf a237299869bc34 2016-04-25	The malware masqueraded as the official app used by post office users in Austria. It can overlay its own credential input view on top of eight popular apps, including Ubercab, Youtube, and Wechat.

Table 1: Overview of the five Europe Smishing campaigns ordered in the beginning dates

(*: First publicized by FireEye researchers)

Shortened links were commonly used in the five campaigns. In total, we identified 30 short links. Some URL shorteners provide analytics, through which anyone can see how many people clicked the link and the countries those clicks came from. For example, Figure 2 shows that there were 135 clicks from Germany on one of the **Whats-Germany** samples, and 1,633 clicks from Austria on one of the **Post-Austria** samples.

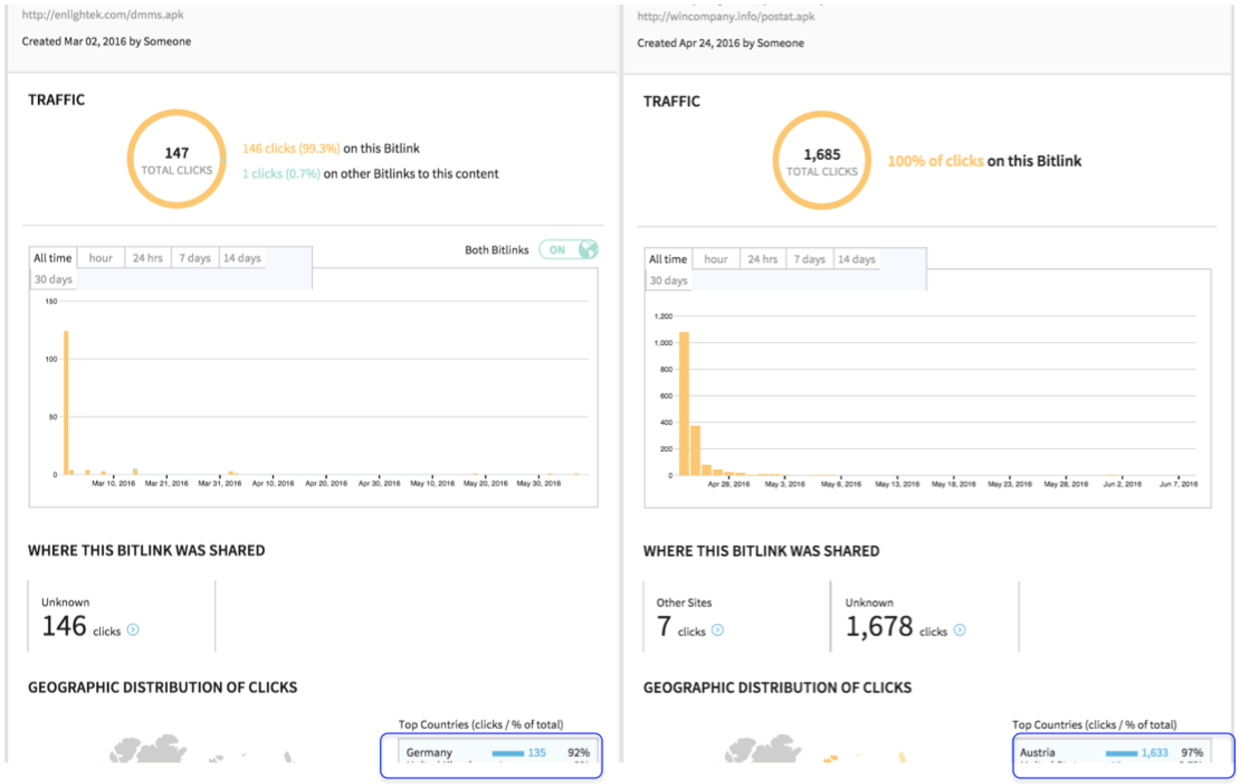

http://enlightek.com/dmms.apk
Created Mar 02, 2016 by Someone
TRAFFIC
147 TOTAL CLICKS
146 clicks (99.3%) on this Bitlink
1 clicks (0.7%) on other Bitlinks to this content
All time | hour | 24 hrs | 7 days | 14 days | 30 days
Both Bitlinks ON
WHERE THIS BITLINK WAS SHARED
Unknown 146 clicks
GEOGRAPHIC DISTRIBUTION OF CLICKS
Top Countries (clicks / % of total)
Germany 135 92%
http://wincompany.info/postat.apk
Created Apr 24, 2016 by Someone
TRAFFIC
1,685 TOTAL CLICKS
100% of clicks on this Bitlink
All time | hour | 24 hrs | 7 days | 14 days | 30 days
WHERE THIS BITLINK WAS SHARED
Other Sites 7 clicks
Unknown 1,678 clicks
GEOGRAPHIC DISTRIBUTION OF CLICKS
Top Countries (clicks / % of total)
Austria 1,633 97%

Figure 2: Analytics pages on one Whats-Germany sample and one post-Austria sample

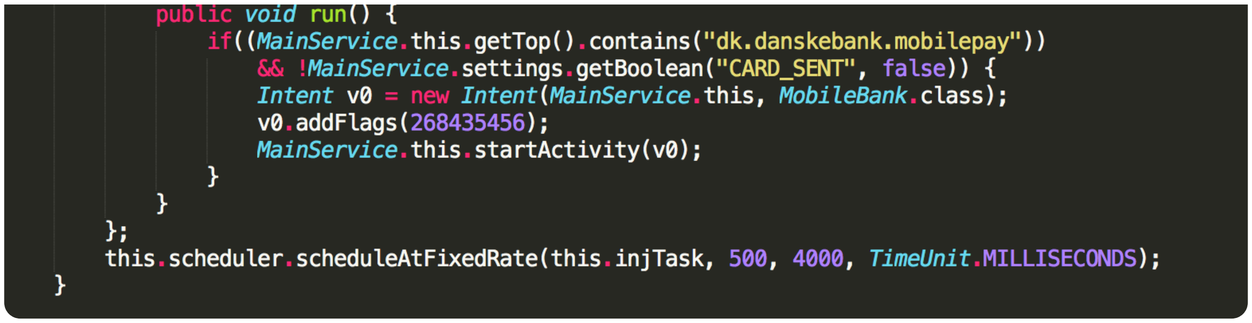
Code Evolution

In the aforementioned Smishing campaigns, we observed that the malware code has been evolving over time. The malware author(s) seems to be working diligently to improve the code by adding new target apps, obfuscating the code to evade detection, and trying to bypass App Ops restrictions.

Adding New Target Apps

All five campaigns attempt to steal credentials from various targeted apps. When the malicious app is started, a background service is triggered to periodically monitor the apps running in the foreground. When the service detects that the foreground app is one of its targeted apps, it overlays a carefully designed phishing view on top of the target app.

Analysis of the malware code shows that this task is executed by a method in the main service, named *initInjTask* in most cases. Figure 3 shows the code of *initInjTask* in one of the earliest samples of the **MPay-Denmark** campaign, in which only a localized app named MobilePay was targeted.

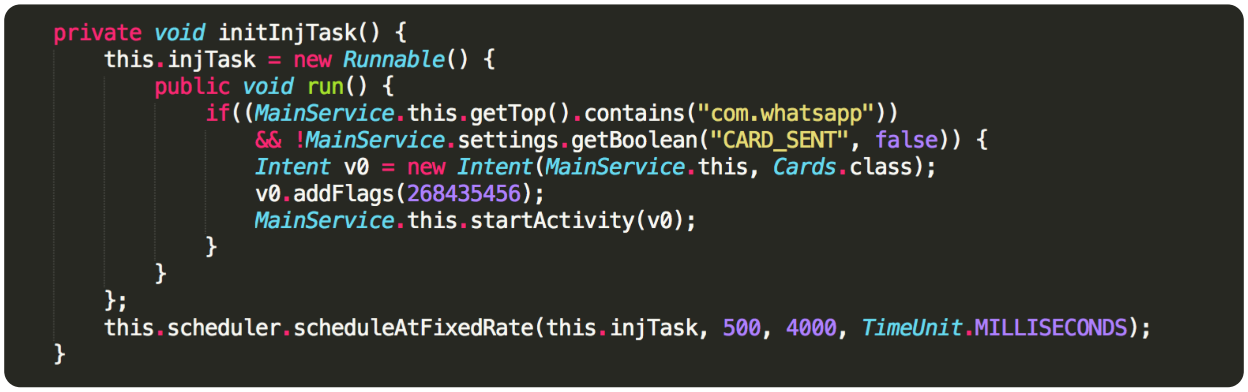


```
            public void run() {
                if((MainService.this.getTop().contains("dk.danskebank.mobilepay"))
                    && !MainService.settings.getBoolean("CARD_SENT", false)) {
                    Intent v0 = new Intent(MainService.this, MobileBank.class);
                    v0.addFlags(268435456);
                    MainService.this.startActivity(v0);
                }
            }
        };
        this.scheduler.scheduleAtFixedRate(this.injTask, 500, 4000, TimeUnit.MILLISECONDS);
    }
```

Figure 3. MobileBank class to be started to overlay app named "dk.danskebank.mobilepay"

(code extracted from app with a MD5 of 49dac3b35afb2e8d3605c72d0d83f631)

Figure 4 shows the code of *initInjTask* in one **Whats-Italy** sample, in which the target was changed to a more widely used app: WhatsApp Messenger.

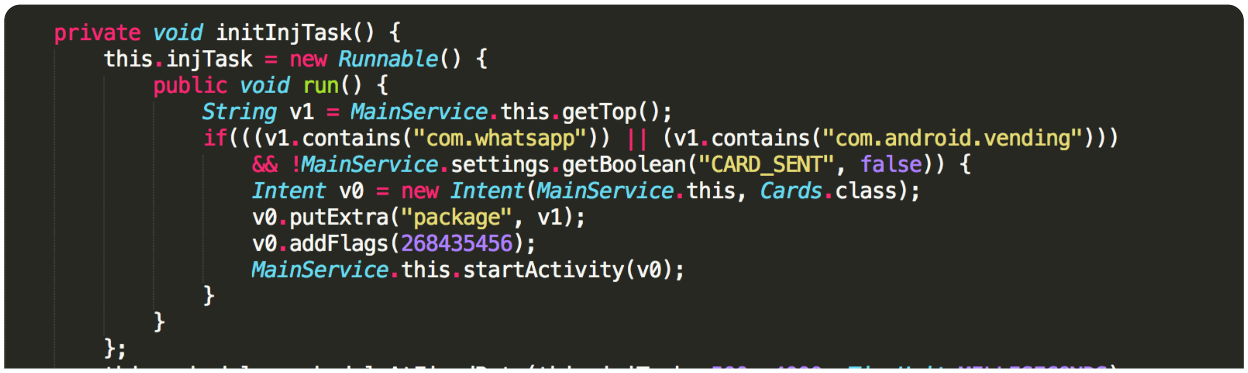


```
    private void initInjTask() {
        this.injTask = new Runnable() {
            public void run() {
                if((MainService.this.getTop().contains("com.whatsapp"))
                    && !MainService.settings.getBoolean("CARD_SENT", false)) {
                    Intent v0 = new Intent(MainService.this, Cards.class);
                    v0.addFlags(268435456);
                    MainService.this.startActivity(v0);
                }
            }
        };
        this.scheduler.scheduleAtFixedRate(this.injTask, 500, 4000, TimeUnit.MILLISECONDS);
    }
```

Figure 4: Cards class to be started to overlay app named "com.whatsapp"

(code extracted from app with a MD5 of 97c2d04aa0f3c3b446fc228c1dbc4837)

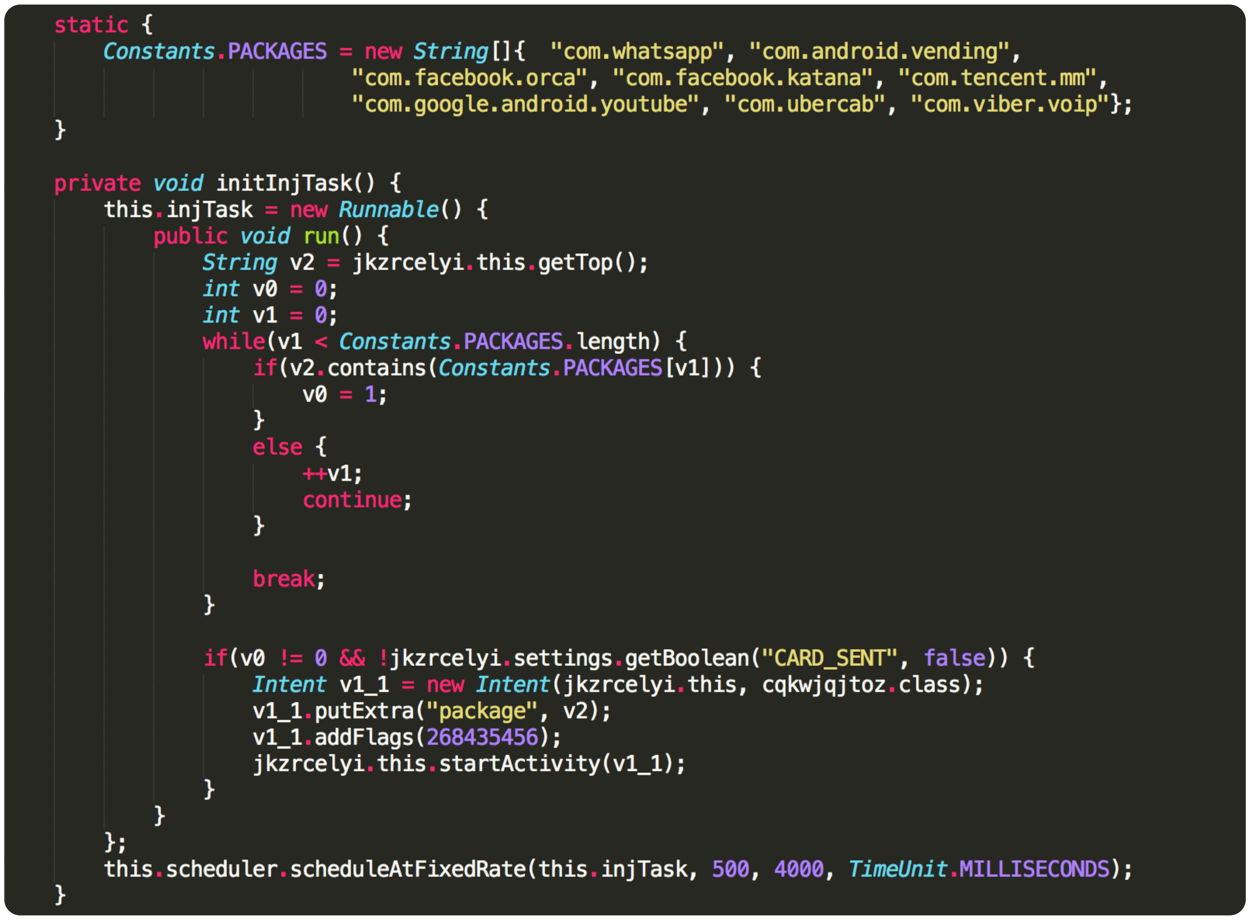
Figure 5 shows the code of *initInjTask* in one **Whats-Germany** sample, in which two apps – WhatsApp and the Google Play Store – were targeted.



```
    private void initInjTask() {
        this.injTask = new Runnable() {
            public void run() {
                String v1 = MainService.this.getTop();
                if(((v1.contains("com.whatsapp")) || (v1.contains("com.android.vending")))
                    && !MainService.settings.getBoolean("CARD_SENT", false)) {
                    Intent v0 = new Intent(MainService.this, Cards.class);
                    v0.putExtra("package", v1);
                    v0.addFlags(268435456);
                    MainService.this.startActivity(v0);
                }
            }
        };
```

(code extracted from app with a MD5 of 9e9d9a3717eed4d558a3f5eddb260901)

Figure 6 shows the code of *initInjTask* in one **Post-Austria** sample (in this case, the malicious app was obfuscated; the code was extracted from the dropped jar file). In total, eight worldwide popular apps – including Uber and Tencent's WeChat – were on its radar.



```
static {
    Constants.PACKAGES = new String[]{  "com.whatsapp", "com.android.vending",
                        "com.facebook.orca", "com.facebook.katana", "com.tencent.mm",
                        "com.google.android.youtube", "com.ubercab", "com.viber.voip"};
}

private void initInjTask() {
    this.injTask = new Runnable() {
        public void run() {
            String v2 = jkzrcelyi.this.getTop();
            int v0 = 0;
            int v1 = 0;
            while(v1 < Constants.PACKAGES.length) {
                if(v2.contains(Constants.PACKAGES[v1])) {
                    v0 = 1;
                }
                else {
                    ++v1;
                    continue;
                }

                break;
            }

            if(v0 != 0 && !jkzrcelyi.settings.getBoolean("CARD_SENT", false)) {
                Intent v1_1 = new Intent(jkzrcelyi.this, cqkwjqjtoz.class);
                v1_1.putExtra("package", v2);
                v1_1.addFlags(268435456);
                jkzrcelyi.this.startActivity(v1_1);
            }
        }
    };
    this.scheduler.scheduleAtFixedRate(this.injTask, 500, 4000, TimeUnit.MILLISECONDS);
}
```

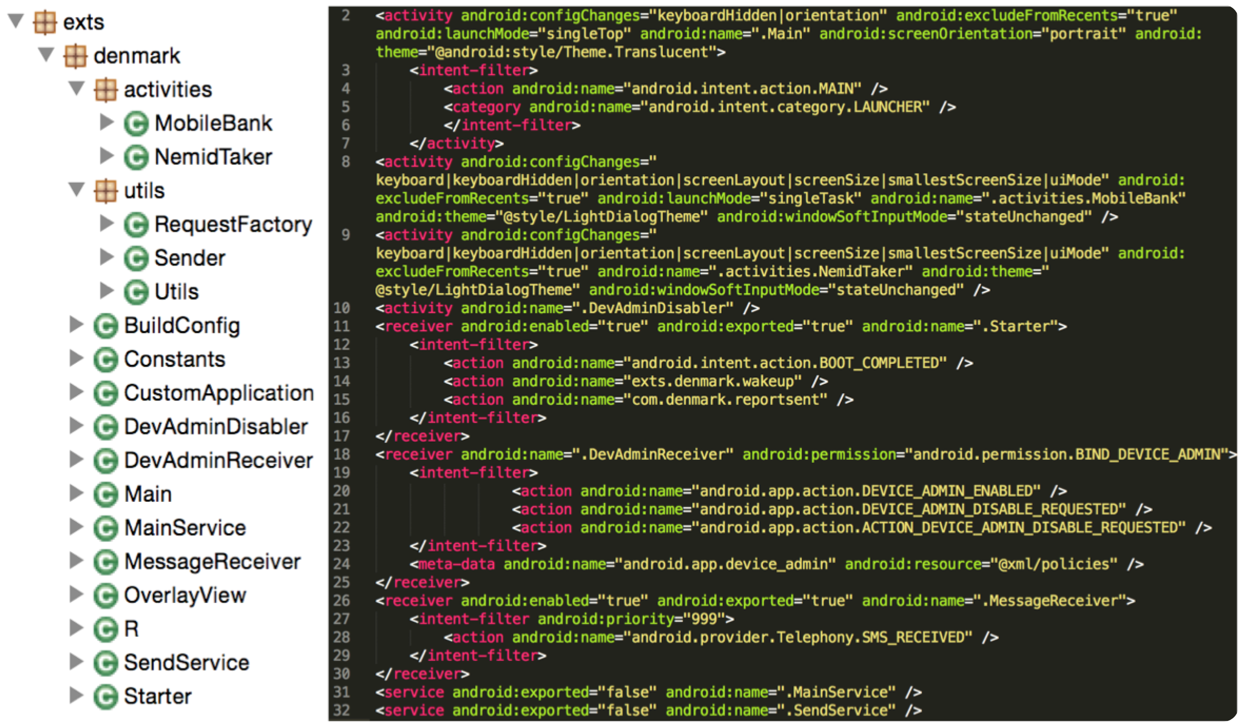
Figure 6: cqkwjqjtoz class to be started to overlay apps 8 popular apps

(code extracted from app with a MD5 of d70296d3dc4937dedd44f93bb3b74034)

The code examples demonstrate changes in the malware over time. Early samples targeted single apps (a localized banking app and WhatsApp) while later samples included a broader range of apps, suggesting that the threat actors continue to both improve their malware and broaden their targeting, presumably for greater financial gain.

Code Obfuscation

In earlier campaigns, including MPay-Denmark, Whats-Italy and Whats-Germany, most of the pieces of information, we see that three receivers are registered for various purposes: to handle incoming SMS messages; to request device-admin privileges; and to start the app at booting time and handle two application-specific events. There are also two services designed to be running in the background and four activities meant to interact with users. With this basic information at hand, adept malware analysts can readily figure out the role played by each part of the code and further understand how these pieces work together to achieve the malware's goal.



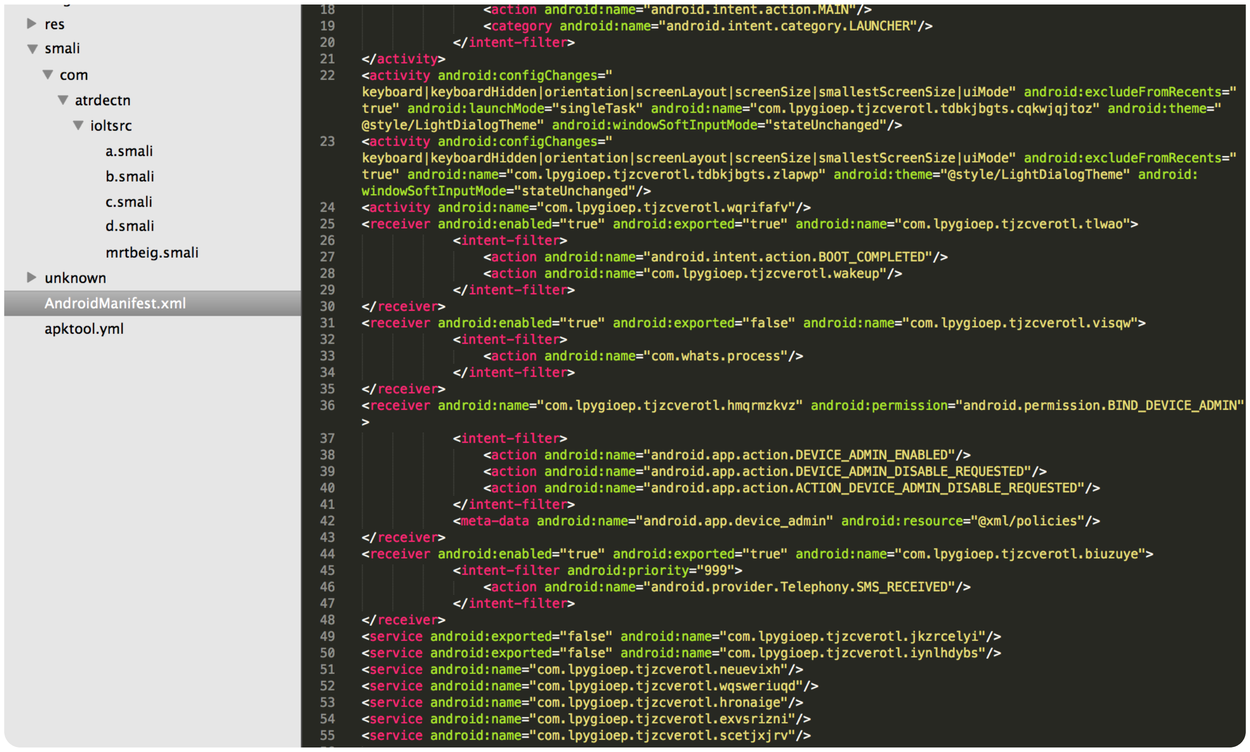
```
exts
  denmark
    activities
      MobileBank
      NemidTaker
    utils
      RequestFactory
      Sender
      Utils
    BuildConfig
    Constants
    CustomApplication
    DevAdminDisabler
    DevAdminReceiver
    Main
    MainService
    MessageReceiver
    OverlayView
    R
    SendService
    Starter
```

```xml
<activity android:configChanges="keyboardHidden|orientation" android:excludeFromRecents="true"
android:launchMode="singleTop" android:name=".Main" android:screenOrientation="portrait" android:
theme="@android:style/Theme.Translucent">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
<activity android:configChanges="
keyboard|keyboardHidden|orientation|screenLayout|screenSize|smallestScreenSize|uiMode" android:
excludeFromRecents="true" android:launchMode="singleTask" android:name=".activities.MobileBank"
android:theme="@style/LightDialogTheme" android:windowSoftInputMode="stateUnchanged" />
<activity android:configChanges="
keyboard|keyboardHidden|orientation|screenLayout|screenSize|smallestScreenSize|uiMode" android:
excludeFromRecents="true" android:name=".activities.NemidTaker" android:theme="
@style/LightDialogTheme" android:windowSoftInputMode="stateUnchanged" />
<activity android:name=".DevAdminDisabler" />
<receiver android:enabled="true" android:exported="true" android:name=".Starter">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
        <action android:name="exts.denmark.wakeup" />
        <action android:name="com.denmark.reportsent" />
    </intent-filter>
</receiver>
<receiver android:name=".DevAdminReceiver" android:permission="android.permission.BIND_DEVICE_ADMIN">
    <intent-filter>
                <action android:name="android.app.action.DEVICE_ADMIN_ENABLED" />
                <action android:name="android.app.action.DEVICE_ADMIN_DISABLE_REQUESTED" />
                <action android:name="android.app.action.ACTION_DEVICE_ADMIN_DISABLE_REQUESTED" />
    </intent-filter>
    <meta-data android:name="android.app.device_admin" android:resource="@xml/policies" />
</receiver>
<receiver android:enabled="true" android:exported="true" android:name=".MessageReceiver">
    <intent-filter android:priority="999">
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
<service android:exported="false" android:name=".MainService" />
<service android:exported="false" android:name=".SendService" />
```

Figure 7: Code structure and manifest file of earlier un-obfuscated code

Since April 2016, we observed that all the samples in our dataset had adopted obfuscation techniques. With the obfuscation, the manifest file became harder to read and the code structure looked totally different.

Figure 8 presents one sample in the **PostDanmark** campaign. The code structure on the left shows that there are five classes named "*a*", "*b*", "*c*", "*d*" and "*mrtbeig*" with a same package name of "*com.atrdectn.ioitsrc*". At the right side, the manifest file shows there are four receivers, seven services and four activities declared, with a different package name of "*com.lpygioep.tjzcverotl*". So where is the code of these declared classes? What are the purposes of these classes named at left? Here the code is much more complex to analyze.



```
▶ res
▼ smali
  ▼ com
    ▼ atrdectn
      ▼ ioltsrc
          a.smali
          b.smali
          c.smali
          d.smali
          mrtbeig.smali
▶ unknown
  AndroidManifest.xml
  apktool.yml
```

```xml
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
</activity>
<activity android:configChanges="
keyboard|keyboardHidden|orientation|screenLayout|screenSize|smallestScreenSize|uiMode" android:excludeFromRecents="
true" android:launchMode="singleTask" android:name="com.lpygioep.tjzcverotl.tdbkjbgts.cqkwjqjtoz" android:theme="
@style/LightDialogTheme" android:windowSoftInputMode="stateUnchanged"/>
<activity android:configChanges="
keyboard|keyboardHidden|orientation|screenLayout|screenSize|smallestScreenSize|uiMode" android:excludeFromRecents="
true" android:name="com.lpygioep.tjzcverotl.tdbkjbgts.zlapwp" android:theme="@style/LightDialogTheme" android:
windowSoftInputMode="stateUnchanged"/>
<activity android:name="com.lpygioep.tjzcverotl.wqrifafv"/>
<receiver android:enabled="true" android:exported="true" android:name="com.lpygioep.tjzcverotl.tlwao">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"/>
                <action android:name="com.lpygioep.tjzcverotl.wakeup"/>
            </intent-filter>
</receiver>
<receiver android:enabled="true" android:exported="false" android:name="com.lpygioep.tjzcverotl.visqw">
            <intent-filter>
                <action android:name="com.whats.process"/>
            </intent-filter>
</receiver>
<receiver android:name="com.lpygioep.tjzcverotl.hmqrmzkvz" android:permission="android.permission.BIND_DEVICE_ADMIN"
>
            <intent-filter>
                <action android:name="android.app.action.DEVICE_ADMIN_ENABLED"/>
                <action android:name="android.app.action.DEVICE_ADMIN_DISABLE_REQUESTED"/>
                <action android:name="android.app.action.ACTION_DEVICE_ADMIN_DISABLE_REQUESTED"/>
            </intent-filter>
            <meta-data android:name="android.app.device_admin" android:resource="@xml/policies"/>
</receiver>
<receiver android:enabled="true" android:exported="true" android:name="com.lpygioep.tjzcverotl.biuzuye">
            <intent-filter android:priority="999">
                <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
            </intent-filter>
</receiver>
<service android:exported="false" android:name="com.lpygioep.tjzcverotl.jkzrcelyi"/>
<service android:exported="false" android:name="com.lpygioep.tjzcverotl.iynlhdybs"/>
<service android:name="com.lpygioep.tjzcverotl.neuevixh"/>
<service android:name="com.lpygioep.tjzcverotl.wqsweriuqd"/>
<service android:name="com.lpygioep.tjzcverotl.hronaige"/>
<service android:name="com.lpygioep.tjzcverotl.exvsrizni"/>
<service android:name="com.lpygioep.tjzcverotl.scetjxjrv"/>
```

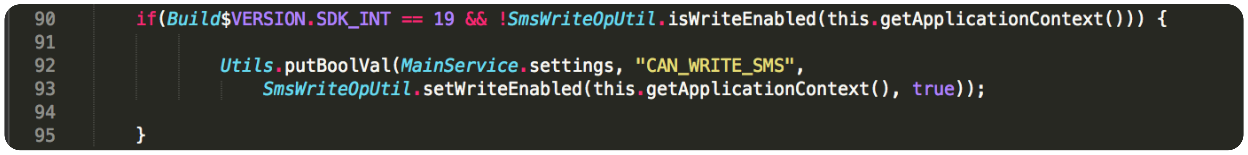
Figure 8: Code structure and manifest file of later obfuscated code

Deeper investigation showed that these classes defined on the left side compose the real payload and overlay the phishing view on top of the eight popular benign apps. Their code is in fact hidden in the asset file named ***mptxip.dat***, which was encoded in a special manner beforehand.

The classes at the left side are actually unpacking code to decode the asset file, to load the real payload at runtime, and leverage reflection to execute the malicious code in the payload. This process is usually much more complex, and involves a round of static analysis first to understand what is in the code, then dynamic analysis to recover the real payload, and then both analyses to understand the real payload. Antivirus vendors often have difficulty identifying such threats. As of June 8, 2016, only 6 out of 54 anti-virus tools labeled these samples as malicious.

Bypassing App Ops Restriction

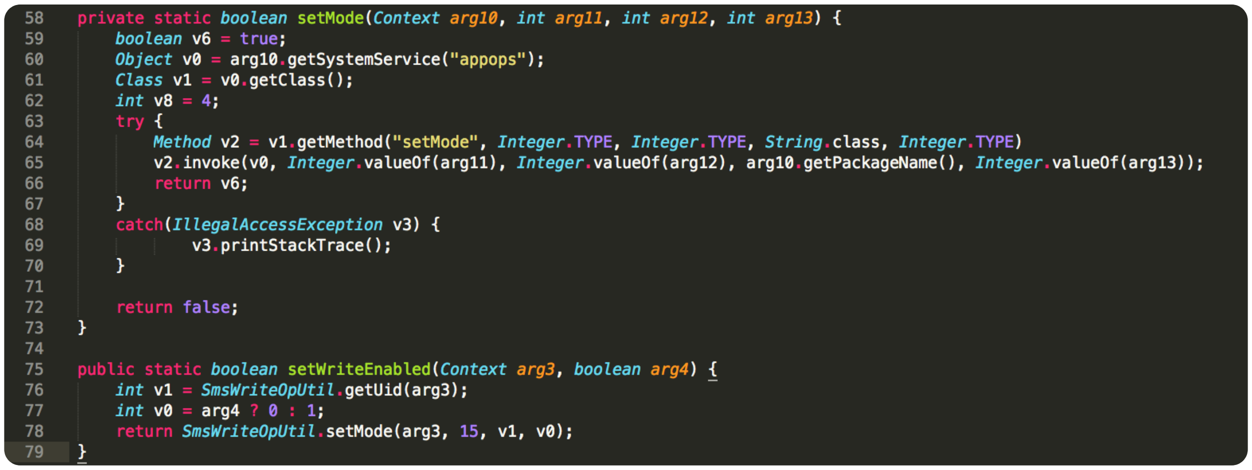
Android uses app permissions to restrict the set of sensitive actions a particular app can take. With earlier versions of the Android operating system, when an app is installed, the user is prompted to agree to the permissions the app requests. If the user declines, then the app isn't installed – it is an all or nothing situation. App Ops is a service framework introduced in Android 4.3 that allows the permissions of individual apps to be changed at runtime. With App Ops, users can disallow some permission requests at runtime. Interestingly, we observed that, starting from the **Whats-Italy** campaign, the overlay malware began to adopt some code to bypass these start time. It checks whether the build version of the device is 19 (Android 4.4) and whether the WRITE_SMS ops are disabled. If both conditions are true, the malware will call the method ***setWriteEnabled*** of class ***SmsWriteOpUtil*** (at line 93) to re-enable the permission of writing SMS.



```
90      if(Build$VERSION.SDK_INT == 19 && !SmsWriteOpUtil.isWriteEnabled(this.getApplicationContext())) {
91
92              Utils.putBoolVal(MainService.settings, "CAN_WRITE_SMS",
93                  SmsWriteOpUtil.setWriteEnabled(this.getApplicationContext(), true));
94
95      }
```

Figure 9: Code to check and re-enable the permission of writing SMS

Figure 10 shows the major code of ***SmsWriteOpUtil*** to re-enable the SMS writing permission. At line 60, a handle to the system service App Ops is fetched. At line 61, reflection is used to get access to the particular class. At line 64 and 65, the reflection methods ***getMethod*** and ***invoke*** are used to call a method named ***setMode***. These API methods are usually designed for use by other framework code or pre-installed apps. However, in this case threat actors use reflection to bypass the App Ops restriction.



```
58  private static boolean setMode(Context arg10, int arg11, int arg12, int arg13) {
59      boolean v6 = true;
60      Object v0 = arg10.getSystemService("appops");
61      Class v1 = v0.getClass();
62      int v8 = 4;
63      try {
64          Method v2 = v1.getMethod("setMode", Integer.TYPE, Integer.TYPE, String.class, Integer.TYPE)
65          v2.invoke(v0, Integer.valueOf(arg11), Integer.valueOf(arg12), arg10.getPackageName(), Integer.valueOf(arg13));
66          return v6;
67      }
68      catch(IllegalAccessException v3) {
69              v3.printStackTrace();
70      }
71
72      return false;
73  }
74
75  public static boolean setWriteEnabled(Context arg3, boolean arg4) {
76      int v1 = SmsWriteOpUtil.getUid(arg3);
77      int v0 = arg4 ? 0 : 1;
78      return SmsWriteOpUtil.setMode(arg3, 15, v1, v0);
79  }
```

Figure 10: Code using reflection to call App Ops service and to enable writing SMS

Hosting Sites

To execute Smishing campaigns, threat actors first have to determine where to host their malware. Shared hosting services were used heavily in the RuMMS campaign, but the threat actors in these five campaigns varied it up a bit by using self-registered domains, URL shorteners, and compromised websites.

Self-Registered Domains

In our investigation, we noticed that some of the URL domains were registered a few days the Smishing campaigns.

To lure victim users to clicks these links, the domain names were often carefully crafted for a particular campaign. For example, in the earlier MPay-Denmark campaign, threat actors used the Danish postal service provider as a theme and the Smishing messages came as: "You received an MMS from XXX. Follow hxxp://mms4you[.]us/mms.apk to view the message." Thus, many of the domains included the words "mms" and/or "you", such as ***mmsforyou.pw***, ***mmsservice.pw*** and ***mmstildig.net*** ("til dig" is "for you" in Danish).

In the later **PostDanmark** campaign, the Smishing messages came as: "Your package is available for pick up. Follow hxxp://postdanmark[.]org/post.apk to see all the information on your package:" Thus, many URL domains had the words "post" and/or "danmark" present, such as ***postdanmark.net***, ***postdanmark.online***, ***postdanmark.menu*** and ***postdanmarks.com***. Note that the official website for Post Danmark is "www.postdanmark.dk", so all these phishing URLs were actually mimicking the official website for Post Danmark.

Shortened URLs

A small screen size makes shortened URLs perfect for mobile devices. Threat actors seem to understand this, and will leverage it for their own gain. While monitoring these five Smishing campaigns in Europe, we observed shortened URLs being used frequently. In total, we observed four different URL shorteners were used at least once, including **bit.ly**, **tr.im**, **is.gd** and **jar.ma**.

Of the four, **bit.ly** has been the most commonly used URL shortener. In total, we identified **27** bit.ly links were used from February 2016 to June 2016. The other three URL shorteners were not observed until June 2016, and only one was used for each service. Diversifying URL shorteners suggests that the threat actors are trying to avoid detection.

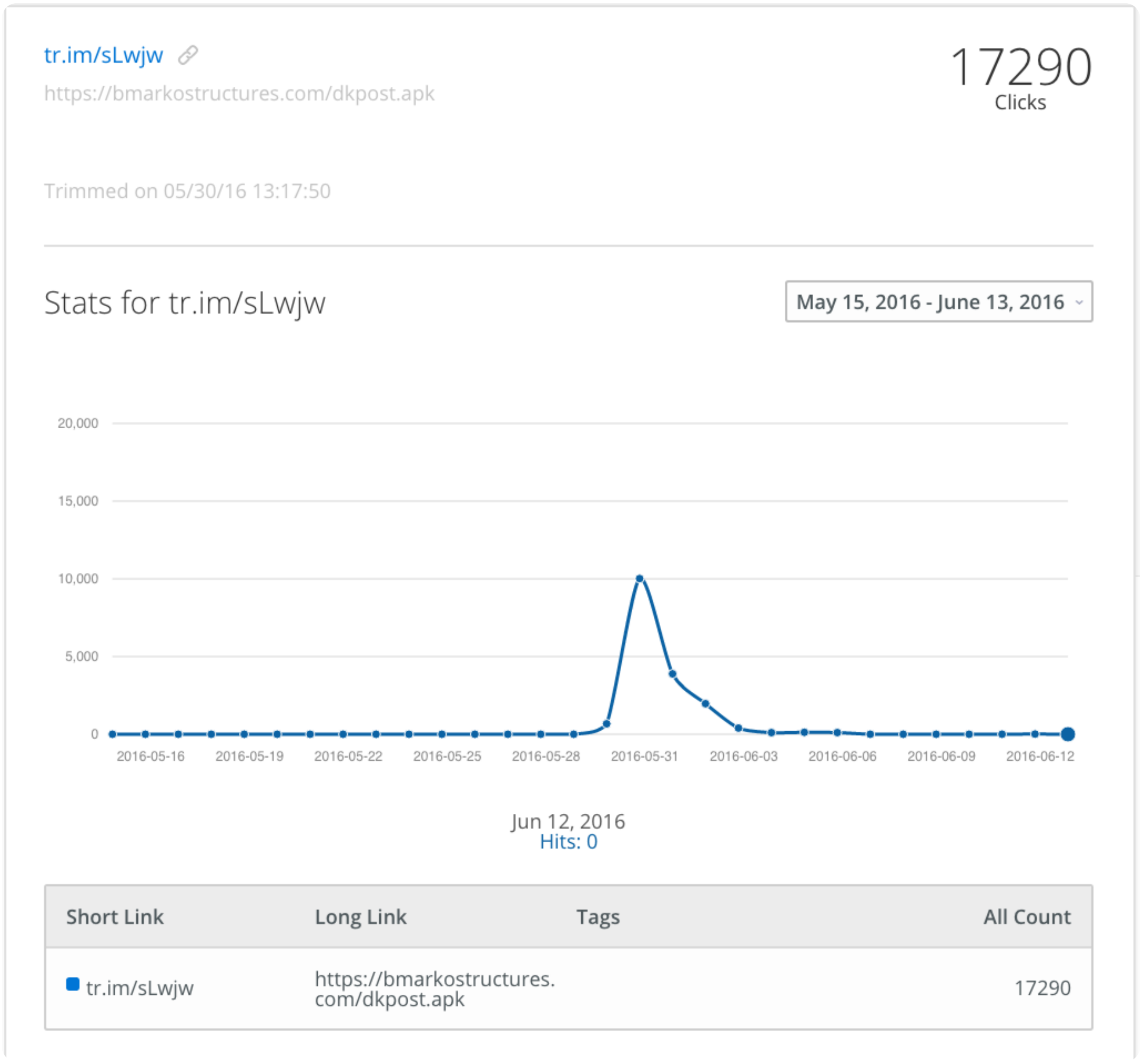
Compromised Websites

It is costly to use self-registered domains to host malware. More capable threat actors might choose to use compromised websites for the same purpose. Despite the risk of the victim site detecting the compromise and removing the malware, this method can be effective: the compromise is often not noticed until some time later, and the number of victim clicks is usually highest at the start of a campaign and decays a few days after the malware goes online.

While monitoring the five Smishing campaigns, we observed compromised websites were used frequently. For example, the analytics page for the shortened URL hxxps://bitly[.]com/1qRey7a+ shows that on April 13, 2016, website kgiexport.com was hosting an Android app with the file named ***post.apk***.

How Many Clicks?

created. Figure 2 showed analytics pages provided by bit.ly. Figure 10 shows a screenshot of the analytics page provided by tr.im. From these pages, we can collect data on how many people clicked the shortened URL at particular dates, and also the countries these clicks came from.



tr.im/sLwjw

https://bmarkostructures.com/dkpost.apk

17290
Clicks

Trimmed on 05/30/16 13:17:50

Stats for tr.im/sLwjw

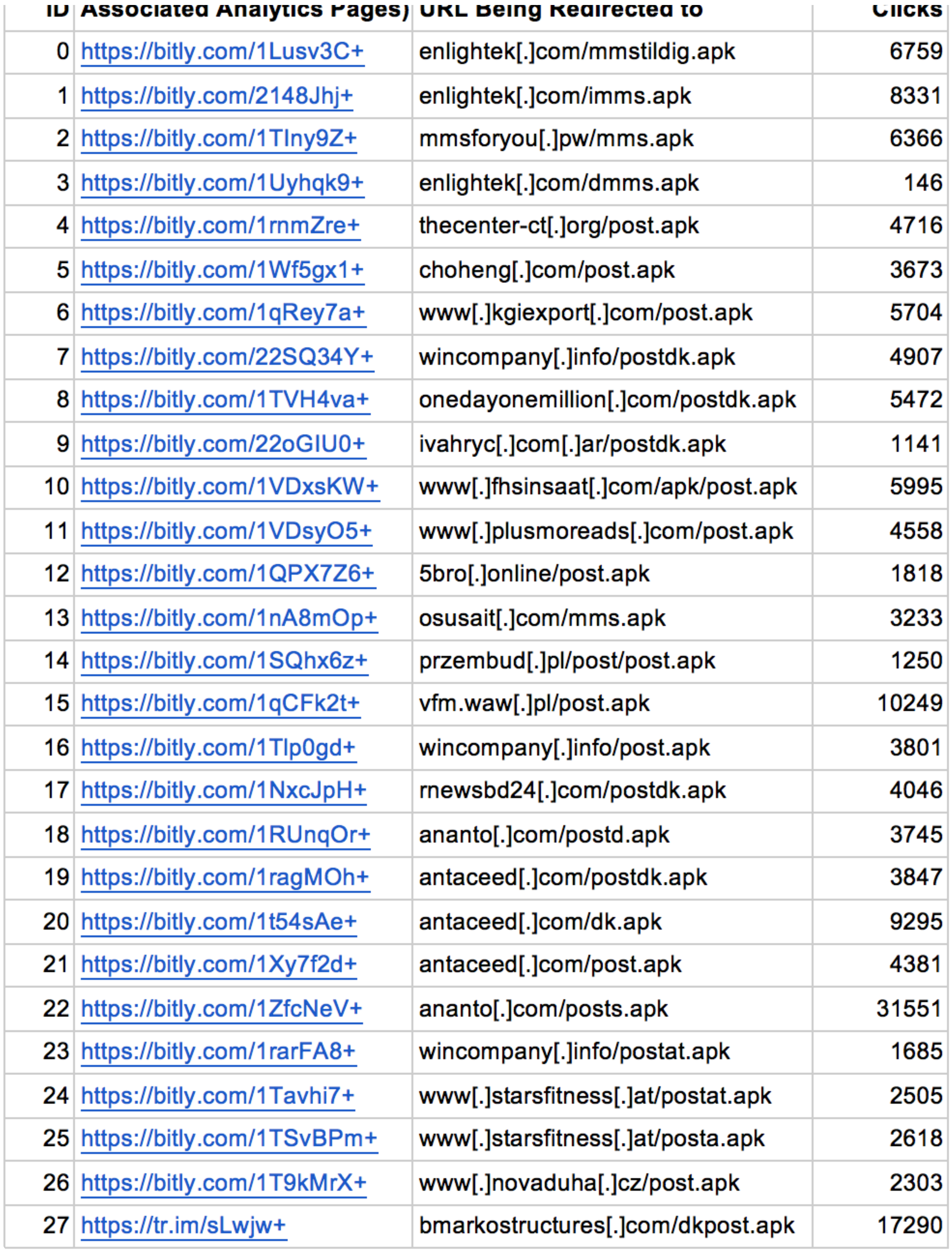
May 15, 2016 - June 13, 2016

Jun 12, 2016
Hits: 0

Short Link	Long Link	Tags	All Count
🟦 tr.im/sLwjw	https://bmarkostructures.com/dkpost.apk		17290

Figure 11: Analytics page provided by tr.im

Table 2 shows relevant information on the 28 short URLs we monitored.



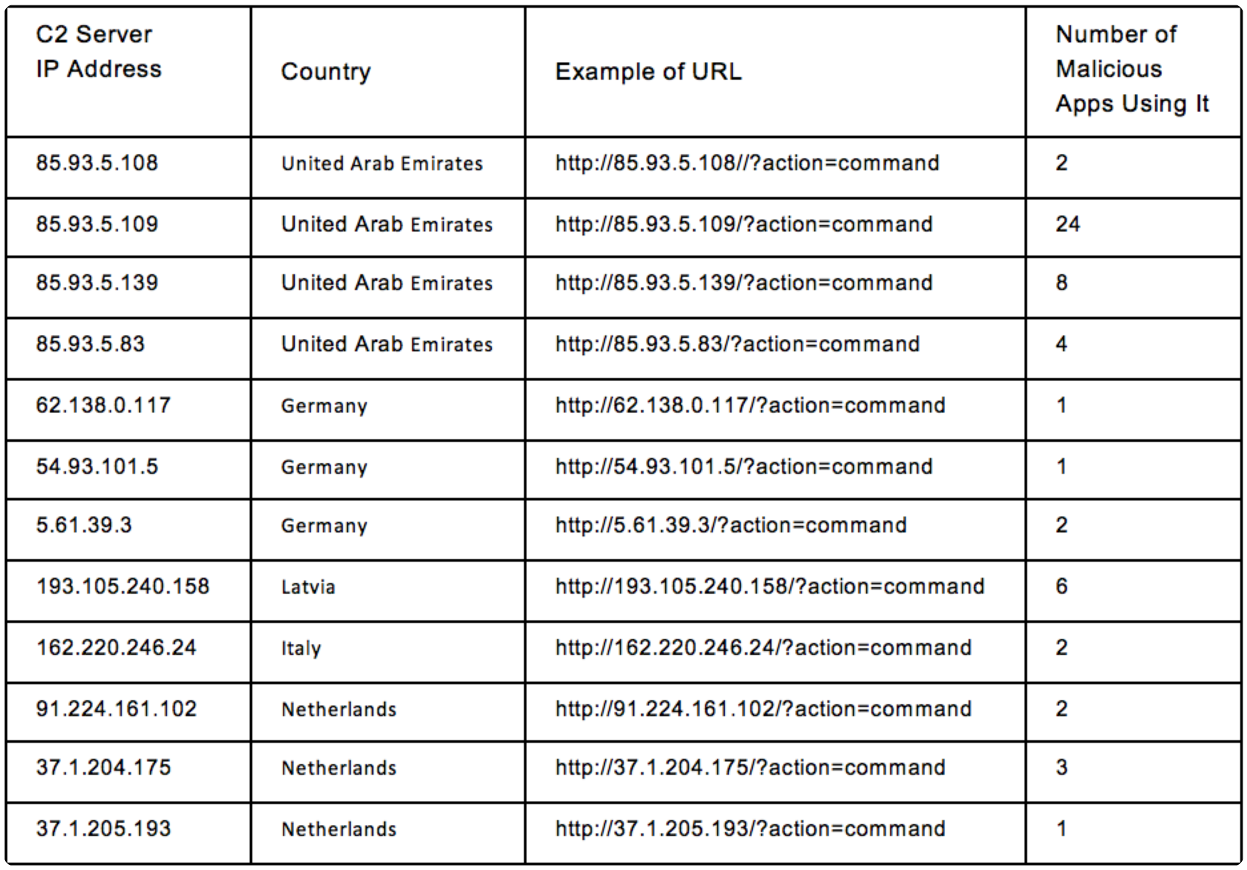
ID	Associated Analytics Pages)	URL Being Redirected to	Clicks
0	https://bitly.com/1Lusv3C+	enlightek[.]com/mmstildig.apk	6759
1	https://bitly.com/2148Jhj+	enlightek[.]com/imms.apk	8331
2	https://bitly.com/1Tlny9Z+	mmsforyou[.]pw/mms.apk	6366
3	https://bitly.com/1Uyhqk9+	enlightek[.]com/dmms.apk	146
4	https://bitly.com/1rnmZre+	thecenter-ct[.]org/post.apk	4716
5	https://bitly.com/1Wf5gx1+	choheng[.]com/post.apk	3673
6	https://bitly.com/1qRey7a+	www[.]kgiexport[.]com/post.apk	5704
7	https://bitly.com/22SQ34Y+	wincompany[.]info/postdk.apk	4907
8	https://bitly.com/1TVH4va+	onedayonemillion[.]com/postdk.apk	5472
9	https://bitly.com/22oGIU0+	ivahryc[.]com[.]ar/postdk.apk	1141
10	https://bitly.com/1VDxsKW+	www[.]fhsinsaat[.]com/apk/post.apk	5995
11	https://bitly.com/1VDsyO5+	www[.]plusmoreads[.]com/post.apk	4558
12	https://bitly.com/1QPX7Z6+	5bro[.]online/post.apk	1818
13	https://bitly.com/1nA8mOp+	osusait[.]com/mms.apk	3233
14	https://bitly.com/1SQhx6z+	przembud[.]pl/post/post.apk	1250
15	https://bitly.com/1qCFk2t+	vfm.waw[.]pl/post.apk	10249
16	https://bitly.com/1Tlp0gd+	wincompany[.]info/post.apk	3801
17	https://bitly.com/1NxcJpH+	rnewsbd24[.]com/postdk.apk	4046
18	https://bitly.com/1RUnqOr+	ananto[.]com/postd.apk	3745
19	https://bitly.com/1ragMOh+	antaceed[.]com/postdk.apk	3847
20	https://bitly.com/1t54sAe+	antaceed[.]com/dk.apk	9295
21	https://bitly.com/1Xy7f2d+	antaceed[.]com/post.apk	4381
22	https://bitly.com/1ZfcNeV+	ananto[.]com/posts.apk	31551
23	https://bitly.com/1rarFA8+	wincompany[.]info/postat.apk	1685
24	https://bitly.com/1Tavhi7+	www[.]starsfitness[.]at/postat.apk	2505
25	https://bitly.com/1TSvBPm+	www[.]starsfitness[.]at/posta.apk	2618
26	https://bitly.com/1T9kMrX+	www[.]novaduha[.]cz/post.apk	2303
27	https://tr.im/sLwjw+	bmarkostructures[.]com/dkpost.apk	17290

Table 2: Click counts on each short URL

In total, the 28 short links were clicked 161,349 times. Of these clicks, 130,636 were from the short links were created, and there were 30,749 clicks (21.33%) on the second day after short links were created. These clicks come primarily from two countries: Denmark (88.66%) and Austria (5.30%). A handful of other countries might be impacted as well, including Germany, Luxembourg, Spain, Sweden, Norway, United Kingdom, Netherlands, Italy, Greece, and Turkey.

C2 Server

All of the malicious apps we analyzed contacted a hard-coded C2 server for sending device relevant information and getting back instructions. The URL used is in the form of http://$C2.$SERVER.$IP/?action=command. In total, we found **12 C2 servers** hosted in five different countries were involved in these campaigns. Table 3 shows relevant information for each C2 server used in these campaigns.

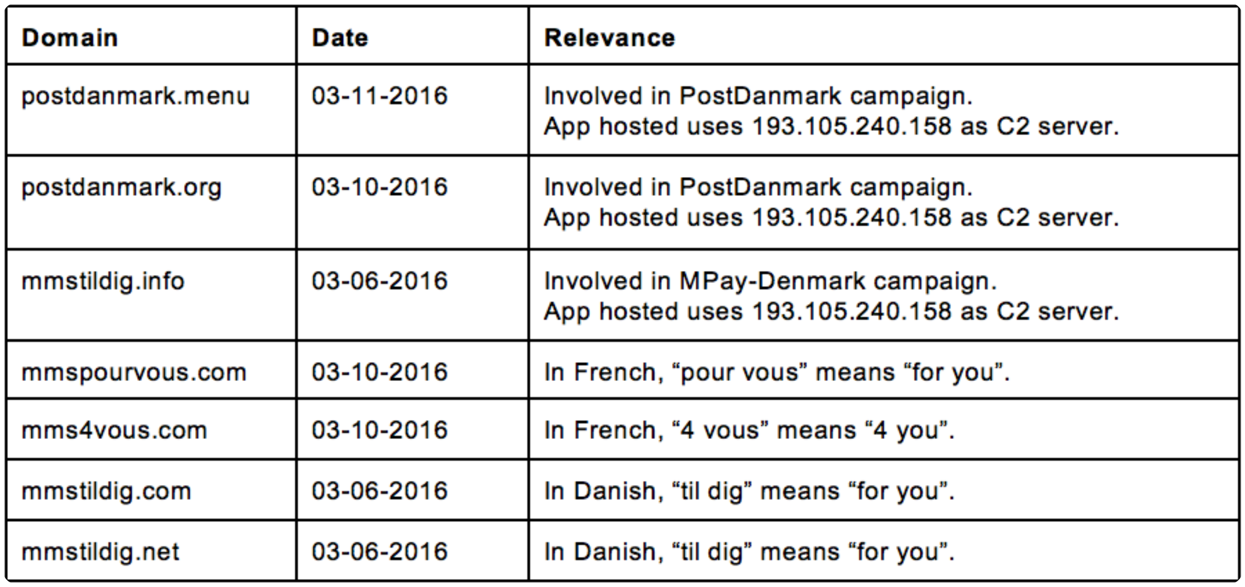


C2 Server IP Address	Country	Example of URL	Number of Malicious Apps Using It
85.93.5.108	United Arab Emirates	http://85.93.5.108//?action=command	2
85.93.5.109	United Arab Emirates	http://85.93.5.109/?action=command	24
85.93.5.139	United Arab Emirates	http://85.93.5.139/?action=command	8
85.93.5.83	United Arab Emirates	http://85.93.5.83/?action=command	4
62.138.0.117	Germany	http://62.138.0.117/?action=command	1
54.93.101.5	Germany	http://54.93.101.5/?action=command	1
5.61.39.3	Germany	http://5.61.39.3/?action=command	2
193.105.240.158	Latvia	http://193.105.240.158/?action=command	6
162.220.246.24	Italy	http://162.220.246.24/?action=command	2
91.224.161.102	Netherlands	http://91.224.161.102/?action=command	2
37.1.204.175	Netherlands	http://37.1.204.175/?action=command	3
37.1.205.193	Netherlands	http://37.1.205.193/?action=command	1

Table 3: C2 Server Relevant Information

In particular, IP address 85.93.5.109 has been used by 24 malicious apps in the **PostDanmark** and **post-Austria** campaigns. IP address 85.93.5.139 has been used by eight malicious apps in the **PostDanmark** campaign. Note that the first four C2 servers are within the same 85.93.5.0/24 network segment. In total, we found 38 malicious samples contacting these four C2

While monitoring the registration records for these self-registered domains, we found something interesting: in March 2016, a single email address (l[REDACTED]a@gmail.com) registered three domains, including ***postdanmark.org***, ***postdanmark.menu*** and ***mmstildig.info***, for two of the five campaigns. Using reverse lookup, we found another four similar domains were also registered by the same email address in March 2016. Table 4 shows the relevant information for these domains.



Domain	Date	Relevance
postdanmark.menu	03-11-2016	Involved in PostDanmark campaign. App hosted uses 193.105.240.158 as C2 server.
postdanmark.org	03-10-2016	Involved in PostDanmark campaign. App hosted uses 193.105.240.158 as C2 server.
mmstildig.info	03-06-2016	Involved in MPay-Denmark campaign. App hosted uses 193.105.240.158 as C2 server.
mmspourvous.com	03-10-2016	In French, “pour vous” means “for you”.
mms4vous.com	03-10-2016	In French, “4 vous” means “4 you”.
mmstildig.com	03-06-2016	In Danish, “til dig” means “for you”.
mmstildig.net	03-06-2016	In Danish, “til dig” means “for you”.

Table 4: Domains registered by the suspected threat actor (l[REDACTED]a@gmail.com)

The first three domains were used to host overlay malware for the MPay-Denmark and PostDanmark campaigns. We found no evidence that the latter four domains were used for similar campaigns, but the same registrant email address and the similar naming convention implies that they may have been created for a similar purpose.

Conclusion

Smishing (SMS phishing) offers a unique vector to infect mobile users. The latest Smishing campaigns spreading in Europe show that Smishing is still a popular means for threat actors to distribute their malware. In addition, threat actors have been using diversified host schemes and different C2 servers, and have been continuously refining their malicious code to keep infecting more users and evade detection.

To protect against these threats, FireEye suggests that users not install apps from outside official app stores, and take caution before clicking any links where the origin is unclear.

To detect and defend against such attacks, we advise our customers to deploy our mobile security solution, FireEye MTP/MSM. This helps our clients gain visibility into threats in their traffic is scanned by NX appliances to extend coverage to include mobile devices.

Appendix: Samples

```
df53b59e354462cd0e704b7b21a750f7
6eb92667ebbbcb2c7ddf6230462222fd
3841abcef2b1b37aa7e2d47c535ca80e
265d37013e1ea39b868515cce157dfeb
49dac3b35afb2e8d3605c72d0d83f631
ffe98d97e7d827aa19abb968a528f3fe
f4b8d64af0a53472901b50621f19d6bf
e1d79608b649c22004ad7cc1cd049528
ef5c9b15755719597481c501f6b603ce
6a300ded487671ef39388b8d28927a83
d33b718737de5aa685672a2004e0fa3c
d83d833092a4fa5ecc436d4246c2f7ce
97c2d04aa0f3c3b446fc228c1dbc4837
82b1006a5f45a6d2baf69544414ada81
9e9d9a3717eed4d558a3f5eddb260901
82d89319fabd998328cc6d4efc4db863
228a4b723bf3d8adc53a69dd0f36c746
e911df33f1d156b3309a4ac220c52070
2b90fca41272bec8b8ffefbb2456c001
40449a2ec48c3e630b2eb8c8089828cf
8d0a03981daa93210e184e7fff02883c
fbdde37d41d12f21c049c570c9bda3de
a18818cb3fb6f189560991cef6d1f929
bf7b72dbb2a9155dabc4eda31d273b92
9762441d52bdec725eff6f2f65e721e9
dba6b4bbf61e054fb978acaf70c3d849
93922ee5fbd149f31b0161deca76df77
035d1f3b7fb532a33de7a8445f9fa325
3f2017a5acb3e57801e2771341287001
06e74df867e9cb5c1bafc98165c6c248
20f4cd2baa09e0bd5e12dab50c0898cd
af7a8d32865e8caf51a99c52834d4422
82d89319fabd998328cc6d4efc4db863
bee3746684b072867a5b202bfc5527dd
a18818cb3fb6f189560991cef6d1f929
8959513f65bcca6f16faef59ad2d152f
cfa92cbcb0674429cc9ce216cc008902
d73d54f6f86c58030477cc9a96eedb85
```

EN

1cb4e[illegible]
05131969af2ae6cbfddf789512f02aa2
6e93a7f7911b3e9b522be4b8f950cca4
542f8f77e101d4e8e5d1ef34a3f0df1c
d0a6ba40e05047dc2cff12935c4cf4fb
23988abad7c7b2ecdda23ae7194b7a0d
2c055d7b5199604cd5cf3441073b36b3
a72aa534973eeaf0782a246d502107a3
f1c8a3337cbd56e01e478774f5d55278
da222d4b7993a62665b9eaef10c1846f
152f626eb92676f940ada4b7077acf16
7a99b60349703aed3ab28f498320f247
1b9e1cd2c7f8e227b2ae5fb5bc735536
d84ff5a7e7c0c33dcfa237299869bc34
d70296d3dc4937dedd44f93bb3b74034
88b23b6a5c1b72aeff2fc42e05c173a7
036258e2c51e21c140b5838ce9bfb4f8

Follow us

Mandiant Advantage Platform	Solutions
Platform Overview	Ransomware
Automated Defense	Industrial Controls & OT
Security Validation	Cyber Risk Management
Ransomware Defense Validation	Digital Risk Protection
Attack Surface Management	Insider Threats
Threat Intelligence	Cyber Security Skills Gap
Digital Threat Monitoring	Election Security

EN

Detection and Response

Services

Services Overview

Incident Response

Strategic Readiness

Cyber Defense Transformation

Technical Assurance

View all Services (48)

Expertise on Demand

Mandiant Academy

Overview

Education Formats

Upcoming Courses

On-Demand Courses

Certifications

ThreatSpace Cyber Range

Free Course Sneak Peaks

Resources

Resource Center

Mandiant Blog

Podcasts

Customer Stories

Reports

Webinars

Insights

Infographics

Datasheets

Company

About Us

Careers

Events

Media Center

Leadership

Investor Relations

Partners

Partners Overview

Service Partners

Cyber Risk Partners

Connect with Mandiant

Contact Us

Report an Incident

Customer Support

EN