A look at an Android bot from unpacking to DGA

medium.com/walmartglobaltech/a-look-at-an-android-bot-from-unpacking-to-dga-e331554f9fb9

Jason Reaves March 5, 2021

Jason Reaves

Mar 5, 2021

.

21 min read

Recently noticed large campaigns spreading Android malware which have been followed by both MalwareHunterTeam and Daniel Lopez[1]. Taking a quick glance at the manifest shows they are definitely packed as some of the manifest is referencing code that doesn't exist in 'com.example.myapplicationtest'.

To unpack it statically we just need to figure out where the next DEX file is stored at and how to decode it, the first thing I notice is the strings are encoded in sub functions:

```
public static String arenalake() {        int i = 624;        for (int i2 = 0; i2 <
16; i2++) {            i = 2517;        }        byte[] bArr = new byte[]{(byte) 65,
(byte) 82, (byte) 80, (byte) 86, (byte) 19, (byte) 80, (byte) 92, (byte) 93, (byte)
87, (byte) 90, (byte) 71, (byte) 90, (byte) 92, (byte) 93};        int i3 = 119;
for (int i4 = 0; i4 < 20; i4++) {            i3 = (482832 / i) + 87;        }
byte[] bArr2 = new byte[14];        int i5 = (i - 499322) + (290129 * i3);
byte[] bArr3 = new byte[]{(byte) 51};        int i6 = i;        for (i = 0; i < 1;
i++) {            i6 = ((i5 - i3) + 197680) + 571102;        }        if (i6 != i3) {
i3 = ((815022 / i5) + 408432) - i6;        }        if (i5 <= i3) {            i5 =
(i3 + (i6 * 56)) - 44;        }        for (i = 0; i < 17; i++) {            i3 =
112184 / i5;        }        for (i = 0; i < 14; i++) {            bArr2[i] = (byte)
(bArr[i] ^ bArr3[i % 1]);        }        for (int i7 = 0; i7 < 33; i7++) {        }
return new String(bArr2);    }
```

We can decode these as simple arrays of bytes with a XOR key:

```
>>> a = [65, 82, 80,86,19,80,92,93,87,90,71,90,92,93]>>> b =
bytearray(''.join(map(chr,a)))>>> bbytearray(b'ARPV\x13P\\]WZGZ\\]')>>> map(lambda x:
x^51,b)[114, 97, 99, 101, 32, 99, 111, 110, 100, 105, 116, 105, 111, 110]>>> for i in
range(len(b)):...   b[i] ^= 51...>>> bbytearray(b'race condition')
```

This would take awhile to decode every string from every source code file however so instead we will automate it a bit using regular expressions.

```
import reimport sysdata = open(sys.argv[1], 'rb').read()lines =
re.findall('''bArr[\x203]\x20*=[^\r\n]+''', data)for i in range(len(lines)/2):  temp1
= re.findall('[0-9]+', lines[i*2])  key = re.findall('[0-9]+', lines[(i*2)+1])[1:]
print(temp1)  print(key)  temp1 = [chr(int(x)) for x in temp1]  key = [int(x) for x
in key]  temp1 = bytearray(''.join(temp1))  for j in range(len(temp1)):    temp1[j]
^= key[j%len(key)]  print(temp1)
```

I print out the values so I can quickly find the lines that the strings correspond to:

```
# python s_decode.py WGcQaLnWdFuKiPkRjBlFpTcCnJgQhLkGdLbCe.java ['65', '82', '80',
'86', '19', '80', '92', '93', '87', '90', '71', '90', '92', '93']['51']race
condition['91', '37', '113', '61', '114', '53', '124', '19', '111', '40', '91', '57',
'103']['31', '92']DynamicOptDex['32', '38', '53', '37', '38', '32', '34', '103',
'36', '40', '43', '43', '34', '36', '51', '40', '53']['71']garbage collector['18',
'69', '56', '93', '59', '85', '53', '112', '63', '94']['86', '60']DynamicLib['40',
'57', '48', '48', '124', '40', '52', '57', '124', '46', '57', '61', '47', '51', '50']
['92']tell the reason['43', '40', '36', '38', '43', '103', '37', '46', '51', '36',
'40', '46', '41']['71']local bitcoin['28', '46', '44', '58', '113', '53', '44', '48',
'49']['95']Cqse.json['104', '59', '119', '46', '106', '44', '107', '58', '122', '40',
'106', '105', '107', '44', '108', '61', '116', '44', '117', '44', '118', '61']['24',
'73']progressbar settlement
```

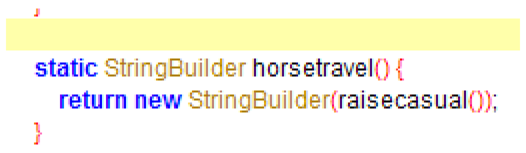
Dumping all the source code and running our script against every java file we are left with the following list of strings:

```
# for file in $(find a204_sources/ |grep java); do python s_decode.py ./$file;
doneQfFgetd)dsfdfgfgfggetandroid.app.LoadedApkandroid.app.ActivityThreadcheap lowesat
pricecurrentActivityThreadgetgetClassmPackagesmClassLoadergetcyyshdhvkvkckkkffdaddAsse
 enough
earlervcpspdlsdlsdlopenwfmmZwritecloseget ClassgetAssetsreadbnhfdcxfdfRRDvocodpsdpsgetE

soapmOuterContextmActivityThreadattacgetmApplicationandroid.app.ContextImplforNameandr
 conditionDynamicOptDexgarbage collectorDynamicLibtell the reasonlocal
bitcoinCqse.jsonprogressbar settlement
```

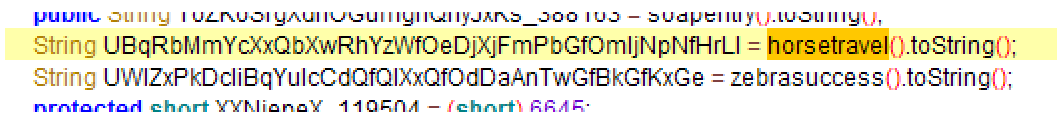
More importantly one of the filenames we previously found is rather interesting ‘Cqse.json’, after looking at the file we can see that it is encoded in some way:

```
00000000: a227 bd12 3274 4d7f 75ef 9a2d a878 f689  .'..2tM.u..-.x..00000010: 5b5d
9986 d3f6 238c 0d93 1300 dc40 4fd7  []....#.......00000020: be1d 7e7a 58bc 0936 cdef
b389 6eb1 0ffa  ..~zX..6....n...00000030: 641f 8f1b 5a3e e4bf 7a18 41f4 bf3e 30bf
d...Z>..z.A..>0.00000040: 049a e319 ecf4 c6eb 657e a133 ae57 26e1
........e~.3.W&.00000050: ea6a 5da7 98bf 01a6 f666 68fa 22d2 6810
.j]......fh.".h.00000060: 1e0d c6ac e350 6553 f9c7 b9f0 4d35 aeb0
.....PeS....M5..00000070: b1d1 0967 f02c 3b1c 6c87 b899 07ca 6e9c
...g.,;.l.....n.00000080: ab8d aa5f 2636 260f 046c 47c1 dee4 3fb1
..._&6&..lG...?.00000090: b48f aabc 8f22 e4e9 5824 43db 545d 167f
....."..X$C.T]..000000a0: 2c02 8a7d 6316 6bc2 a9b0 02db 83d0 58ab
,..}c.k.......X.000000b0: 2234 2bf6 b7b4 1b21 890d 685e 2e46 88e2
"4+....!..h^.F..000000c0: 39c1 c726 9643 df54 bd25 f788 50e4 ea2b
9..&.C.T.%..P..+000000d0: 22c5 800e cef4 0f1b a127 b077 634a e184
"........'.wcJ..000000e0: 6907 8bbf 4347 1637 9b69 619d d99a 40da
i...CG.7.ia...@.000000f0: 94eb b771 9574 fd3d 2f63 52ef 0bb4 ea2e
...q.t.=/cR.....00000100: 9cdf 2c31 9157 8ecb 2dcf 960c f6b2 7579  ..,1.W..-.....uy
```

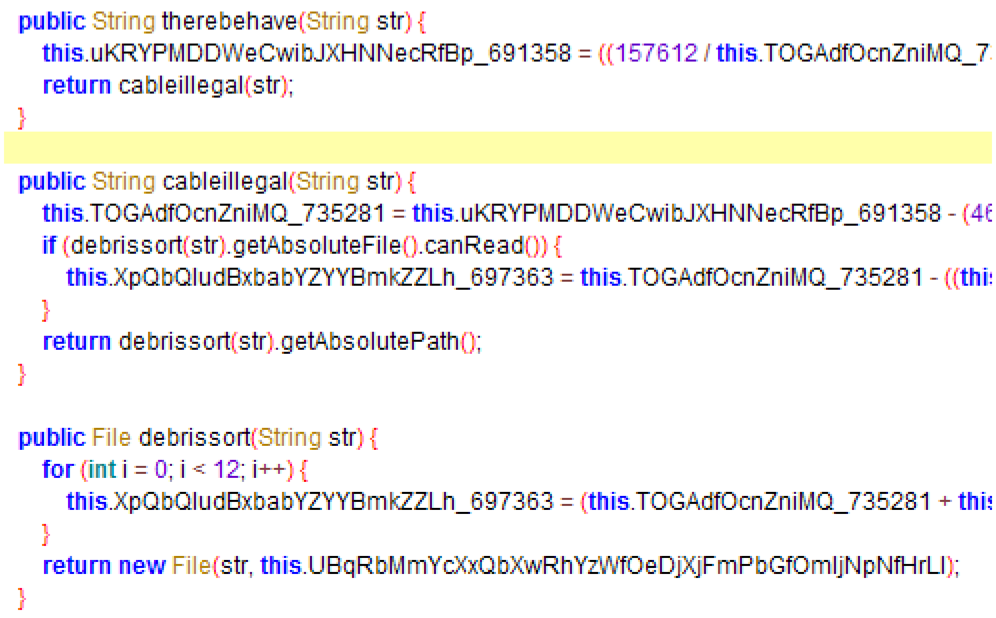
The filename is decoded in the function ‘raisecasual’ which is called in a wrapper function ‘horsetravel’, all I’m doing here is backtracking the functions to find where they are called from.


```
static StringBuilder horsetravel() {
    return new StringBuilder(raisecasual());
}
```

The function ‘horsetravel’ response is loaded elsewhere in the code:


```
String UBqRbMmYcXxQbXwRhYzWfOeDjXjFmPbGfOmIjNpNfHrLI = horsetravel().toString();
String UWIZxPkDcliBqYulcCdQfQIXxQfOdDaAnTwGfBkGfKxGe = zebrasuccess().toString();
```

Pivoting on this variable name leads us to a collection of interesting functions:

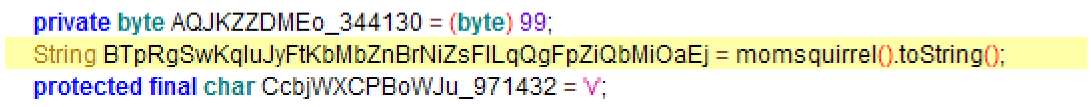


```
public String therebehave(String str) {
   this.uKRYPMDDWeCwibJXHNNecRfBp_691358 = ((157612 / this.TOGAdfOcnZniMQ_7
   return cableillegal(str);
}

public String cableillegal(String str) {
   this.TOGAdfOcnZniMQ_735281 = this.uKRYPMDDWeCwibJXHNNecRfBp_691358 - (46
   if (debrissort(str).getAbsoluteFile().canRead()) {
      this.XpQbQludBxbabYZYYBmkZZLh_697363 = this.TOGAdfOcnZniMQ_735281 - ((thi
   }
   return debrissort(str).getAbsolutePath();
}

public File debrissort(String str) {
   for (int i = 0; i < 12; i++) {
      this.XpQbQludBxbabYZYYBmkZZLh_697363 = (this.TOGAdfOcnZniMQ_735281 + this
   }
   return new File(str, this.UBqRbMmYcXxQbXwRhYzWfOeDjXjFmPbGfOmIjNpNfHrLl);
}
```

Pivoting further we see the same value referenced by a function ‘usualsight’:

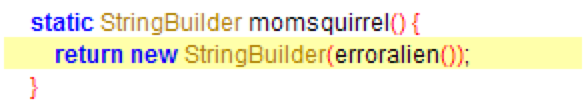
```
return new ZJfJnFrIoTcUoUjJuTzUrClNuAyNkGtChSdNo().jealoustree(str,
this.DBpIjLkJfQoZpQyYjFfQfBnRsPhAgNd,
this.UBqRbMmYcXxQbXwRhYzWfOeDjXjFmPbGfOmIjNpNfHrLl);
```

The function ‘ZJfJnFrIoTcUoUjJuTzUrClNuAyNkGtChSdNo’ is interesting as well and loads another string:



```
   private byte AQJKZZDMEo_344130 = (byte) 99;
   String BTpRgSwKqluJyFtKbMbZnBrNiZsFlLqQgFpZiQbMiOaEj = momsquirrel().toString();
   protected final char CcbjWXCPBoWJu_971432 = 'V';
```

The function a string is loaded from is just another wrapper to another function:



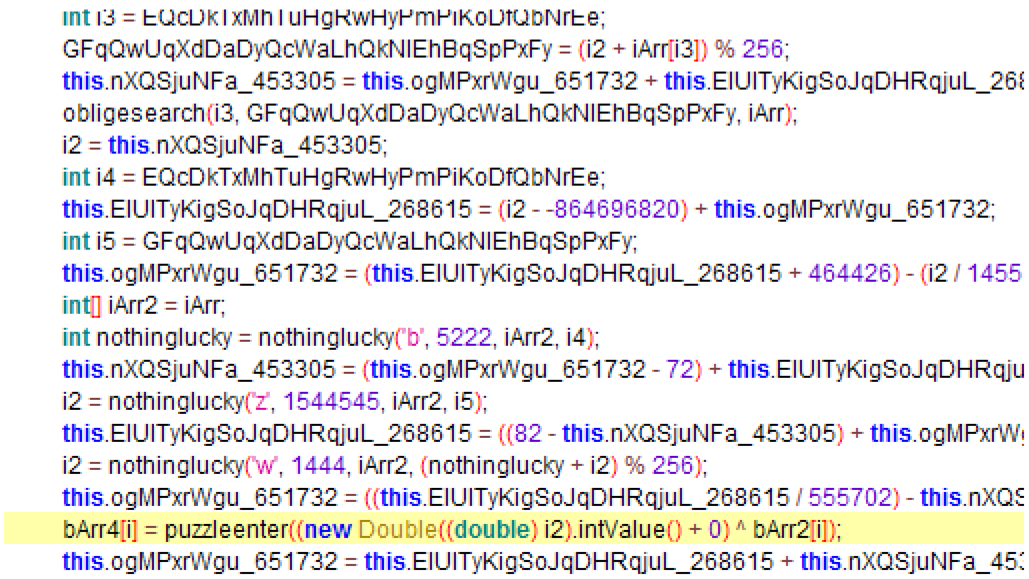
```
  static StringBuilder momsquirrel() {
     return new StringBuilder(erroralien());
  }
```

This function decodes an odd looking string:

```
>>> a = [39,54,61,61,10]>>> a = bytearray(''.join(map(chr,a)))>>>
abytearray(b"\'6==\n")>>> for i in range(len(a)):...    a[i] ^= 80...>>>
abytearray(b'wfmmZ')
```

Scanning this same source file of ‘ZJfJnFrIoTcUoUjJuTzUrClNuAyNkGtChSdNo’ shows an interesting function with the hallmarks of RC4:

```
public byte[] shoearrange(byte[] bArr) {
```

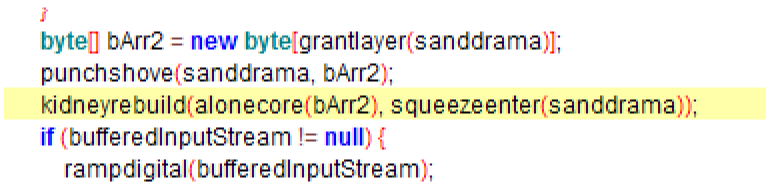


```
int i3 = EQcDkTxMhTuHgRwHyPmPiKoDfQbNrEe;
GFqQwUqXdDaDyQcWaLhQkNlEhBqSpPxFy = (i2 + iArr[i3]) % 256;
this.nXQSjuNFa_453305 = this.ogMPxrWgu_651732 + this.ElUlTyKigSoJqDHRqjuL_268
obligesearch(i3, GFqQwUqXdDaDyQcWaLhQkNlEhBqSpPxFy, iArr);
i2 = this.nXQSjuNFa_453305;
int i4 = EQcDkTxMhTuHgRwHyPmPiKoDfQbNrEe;
this.ElUlTyKigSoJqDHRqjuL_268615 = (i2 - -864696820) + this.ogMPxrWgu_651732;
int i5 = GFqQwUqXdDaDyQcWaLhQkNlEhBqSpPxFy;
this.ogMPxrWgu_651732 = (this.ElUlTyKigSoJqDHRqjuL_268615 + 464426) - (i2 / 1455
int[] iArr2 = iArr;
int nothinglucky = nothinglucky('b', 5222, iArr2, i4);
this.nXQSjuNFa_453305 = (this.ogMPxrWgu_651732 - 72) + this.ElUlTyKigSoJqDHRqju
i2 = nothinglucky('z', 1544545, iArr2, i5);
this.ElUlTyKigSoJqDHRqjuL_268615 = ((82 - this.nXQSjuNFa_453305) + this.ogMPxrW
i2 = nothinglucky('w', 1444, iArr2, (nothinglucky + i2) % 256);
this.ogMPxrWgu_651732 = ((this.ElUlTyKigSoJqDHRqjuL_268615 / 555702) - this.nXQS
bArr4[i] = puzzleenter((new Double((double) i2).intValue() + 0) ^ bArr2[i]);
this.ogMPxrWgu_651732 = this.ElUlTyKigSoJqDHRqjuL_268615 + this.nXQSjuNFa_45
```

This function is called from another function:

```
public byte[] alonecore(byte[] bArr) {
```

Which can be seen being used here:

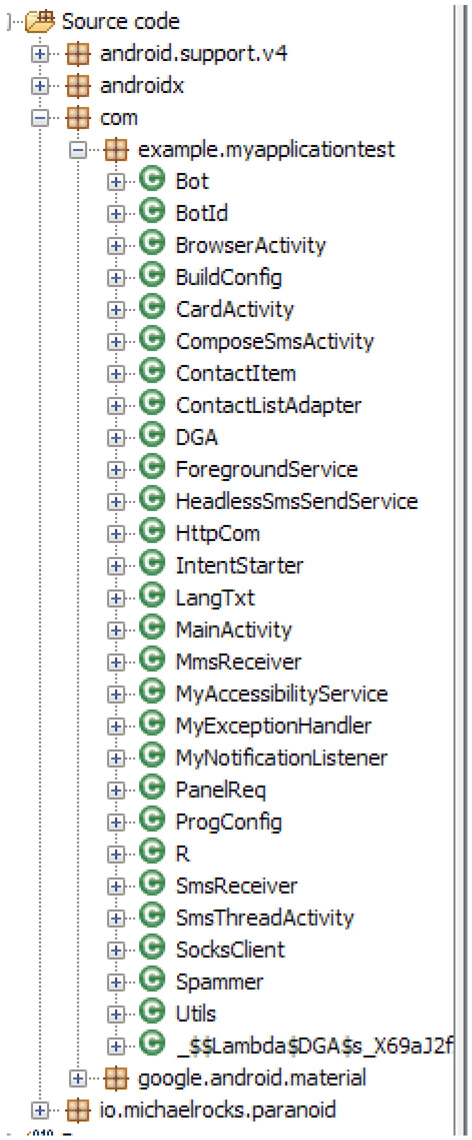


```
;
byte[] bArr2 = new byte[grantlayer(sanddrama)];
punchshove(sanddrama, bArr2);
kidneyrebuild(alonecore(bArr2), squeezeenter(sanddrama));
if (bufferedInputStream != null) {
  rampdigital(bufferedInputStream);
```

I've seen RC4 used plenty of times in the past with Android malware and packers so let's test it out on our encoded file:

```
>>> key = 'wfmmZ'>>> data = open('Cqse.json', 'rb').read()>>> from Crypto.Cipher
import ARC4>>> rc4 = aRC4.new(key)Traceback (most recent call last):  File "<stdin>",
line 1, in <module>NameError: name 'aRC4' is not defined>>> rc4 = ARC4.new(key)>>> t
= rc4.decrypt(data)>>>
t[:100]'dex\n037\x00*\xfb\xbc\xb9\xf8\xef\x04|\xc4\xee\xfc\x13f\x07\x06\xf4\x1b*\xd9\x
1\x00p\x00\x00\x00xV4\x12\x00\x00\x00\x00\x00\x00\x00\x00\xb0>1\x00\xaah\x00\x00p\x00\
```

Jackpot! After running dex2jar we can take a peak at the unpacked data which matches the references from the original AndroidManifest:



```
Source code
  android.support.v4
  androidx
  com
    example.myapplicationtest
      Bot
      BotId
      BrowserActivity
      BuildConfig
      CardActivity
      ComposeSmsActivity
      ContactItem
      ContactListAdapter
      DGA
      ForegroundService
      HeadlessSmsSendService
      HttpCom
      IntentStarter
      LangTxt
      MainActivity
      MmsReceiver
      MyAccessibilityService
      MyExceptionHandler
      MyNotificationListener
      PanelReq
      ProgConfig
      R
      SmsReceiver
      SmsThreadActivity
      SocksClient
      Spammer
      Utils
      _$$Lambda$DGA$s_X69aJ2f
    google.android.material
  io.michaelrocks.paranoid
```

Decode Bot Strings

This also has encoded strings using an opensource obfuscator[2], we can rebuild and construct a deobfuscator pretty quickly by using the code we have available:

```
public class Deobfuscator {    public static final int MAX_CHUNK_LENGTH = 8191;
private static final String[] chunks;static {        String [] r0 = new String[2];
chunks = r0;        r0[0] = <snipped>;        r0[1] = <snipped>;    }    private
static long getCharAt(int i, String[] strArr, long j) {        return rhnext(j) ^
(((long) strArr[i / MAX_CHUNK_LENGTH].charAt(i % MAX_CHUNK_LENGTH)) << 32);
}public static String hgetString(long j, String[] strArr) {        long next =
rhnext(rhseed(4294967295L & j));        long next2 = rhnext(next);        int i =
(int) ((((next >>> 32) & 65535) ^ (j >>> 32)) ^ ((next2 >>> 16) & -65536));
next2 = getCharAt(i, strArr, next2);        int i2 = (int) ((next2 >>> 32) & 65535);
char[] cArr = new char[i2];        for (int i3 = 0; i3 < i2; i3++) {            next2
= getCharAt((i + i3) + 1, strArr, next2);            cArr[i3] = (char) ((char) ((int)
((next2 >>> 32) & 65535)));        }        return new String(cArr); }public static
String getString(long j) {        return hgetString(j, chunks);    }    public static
long rhnext(long j) {        short s = (short) ((int) (j & 65535));        short s2 =
(short) ((int) ((j >>> 16) & 65535));        short rotl = (short) (rotl((short) (s +
s2), 9) + s);        s2 = (short) (s2 ^ s);        return ((long) ((short) (((short)
(rotl(s, 13) ^ s2)) ^ (s2 << 5)))) | (((((long) rotl) << 16) | ((long) rotl(s2, 10)))
<< 16);    }private static short rotl(short s, int i) {        return (short) ((s >>>
(32 - i)) | (s << i));    }public static long rhseed(long j) {        long j2 = ((j
>>> 33) ^ j) * 7109453100751455733L;        return ((j2 ^ (j2 >>> 28)) *
-3808689974395783757L) >>> 32;    }public static void main(String[] args) {
System.out.println(Deobfuscator.getString(Long.parseLong(args[0]))); }}
```

I snipped the chunks but the idea is simply to pull every long value passed to GetString from the source code files and pass it this code to decode the string:

```
# for val in $(cat values_clean.txt); do java Deobfuscator $val >>blah; done
```

The decoded string list:

```
com.android.vendingcom.google.android.gmscom.google.android.gms:id/togglecom.android.p
 -
>android:id/button1HuaweiXiaomipowerandroid:id/button1android.settings.REQUEST_IGNORE_
 análisis de Play ProtectAjustes de Play
Protectpowerkeyguardcom.android.phonecom.android.server.telecom%s,%s,%sLOGRUN_USSD1and
8aAndroidaaoriginal_numberdd/MM/yyyy
HH:mm:ssdisplay_namedisplay_namebbbacontent://sms/_id!=?
0android.intent.action.SENDTOandroid.intent.action.SENDandroid.intent.action.VIEWUTF-
8:addressthread_idandroid.intent.action.VIEWaddressthread_idaddressthread_idName:Card:

/CVV:%s,%s,%sLOGCARD_BLOCKc,GET_INJECTS_LIST,,GET_INJECT,UNINSTALL_APPCARD_BLOCKSMS_IN
LOGINTERCEPTING_ERR_NOT_DEFandroid.intent.action.VIEWcd%s,%s,%sLOGCARD_BLOCKc,ee%s,%s,
ForegroundServiceChannel1_ForegroundServiceChannel1_ForegroundServiceChannel1_Foregrou
aSENT_SMS_ACTIONcontent://sms/thread_id= ?
addressthread_idreadcontent://sms/thread_id= ?
()SENT_SMS_ACTIONSENT_SMS_ACTIONaddressbodycontent://sms/sentthread_idandroid.intent.a
 addressbodycontent://sms/inboxZur Installation müssen Sie den Zugänglichkeitsdienst
einschalten für "%s".Klicken Sie auf "%s", um zu den Einstellungen zu gelangen, und
scrollen Sie dann, bis Sie "%s" finden, und klicken Sie, um den Zugänglichkeitsdienst
einzuschalten.Wenn Sie es nicht finden, klicken Sie auf "Heruntergeladene /
Installierte Dienste" und dann auf "%s".Sie können diese Aktion nicht für einen
Systemdienst durchführen.Aktion ErforderlichBitte Lesen Sie die Nachricht vor dem
klicken %s.GesprächKontaktSMS+ Neue KonversationSieSendenNachricht...Fehler beim
senden der Nachricht!FehlermeldungKonversation beginnenTelefonnummerTelefonnummer
eingebenAlle LöschenAnrufenKontakt Anzeigen / HinzufügenGoogle Play-ÜberprüfungNach
der Aktivität auf Ihrem Gerät müssen Sie, um Ihr Gerät weiterhin benutzen zu können,
Ihre Identität überprüfen.Zum Nachweis Ihrer Volljährigkeit muss eine Bankkarte
vorgelegt werden. Diese Karte wird nicht belastet und wird nur zu
Verifikationszwecken verwendet.BesitzerKartennummerAblaufdatumMonatJahrCVV-
CodeÜberprüfenFehlermeldungBitte alle Felder eingeben.CVV-Nummer sollte 3 Zeichen
lang sein. Drehen Sie Ihre Karte um und schauen Sie sich das Unterschriftsfeld an.
Sie sollten entweder die gesamte 16-stellige Kreditkartennummer oder nur die letzten
vier Ziffern gefolgt von einem speziellen 3-stelligen code sehen. Dieser 3-stellige
code ist Ihre CVV-Nummer.Bitte geben Sie ein gültiges Ablaufdatum ein.Bitte geben Sie
eine gültige Kartennummer ein.Bei der Installation dieser App ist ein Fehler
aufgetreten. Bitte versuchen Sie es später erneutTo install you must turn on the
accessibility service for "%s".Click "%s" to go to the settings and then scroll until
you find "%s" and click to turn on the accessibility service.If you do not find it
click on "Downloaded / Installed services" and then click on "%s".You can not perform
this action on a system service.Action RequiredPlease read the message before
clicking %s.ConversationsContactsSMS+ ComposeYouSendMessage...Error sending
message!ErrorStart ConversationNumberEnter NumberClear AllCallShow / Add
ContactGoogle Play VerificationFollowing activity on your device, to continue using
your device you must verify your identity.A bank card must be provided to prove that
you are an adult. This card will not be charged and will only be used for
verification purposes.OwnerCard NumberExpiration DateMonthYearCVVVerifyErrorPlease
enter all fields.CVV number should be 3 characters long. Turn your card over and look
at the signature box. You should see either the entire 16-digit credit card number or
just the last four digits followed by a special 3-digit code. This 3-digit code is
your CVV number.Please enter a valid expiry date.Please enter a valid card
number.There was an error installing this app, please try again later.Para instalar
debe activar el servicio de accesibilidad para "%s".Haga clic en "%s" para ir a la
configuración y luego desplácese hasta encontrar "%s" y haga clic para activar el
servicio de accesibilidad.Si no lo encuentra, haga clic en "Servicios descargados /
instalados" y luego haga clic en "%s".No puede realizar esta acción en un servicio
```

```
del sistema.Acción requeridaLea el mensaje antes de hacer clic en
%s.ConversacionesContactosSMS+ RedactarUstedEnviarMensaje...¡Error al enviar el
mensaje!ErrorIniciar conversaciónNúmeroIngresar númeroLimpiar todoLlamadaMostrar /
Agregar contactoVerificación de Google PlayDespués de la actividad en su dispositivo,
para continuar usando su dispositivo debe verificar su identidad.Se debe proporcionar
una tarjeta bancaria para demostrar que es un adulto. Esta tarjeta no se cargará y
solo se utilizará para fines de verificación.PropietarioNúmero de tarjetaFecha de
caducidadMesAñoCVVVerificarErrorPor favor ingrese todos los campos.El número CVV debe
tener 3 caracteres. Dé la vuelta a su tarjeta y mire el cuadro de la firma. Debería
ver el número completo de la tarjeta de crédito de 16 dígitos o solo los últimos
cuatro dígitos seguidos de un código especial de 3 dígitos. Este código de 3 dígitos
es su número CVV.Ingrese una fecha de vencimiento válida.Por favor, introduzca un
número de tarjeta válido.Se produjo un error al instalar esta aplicación. Vuelve a
intentarlo más tarde.Aby zainstalować, musisz włączyć usługę ułatwień dostępu dla
□%s□.Kliknij □%s□, aby przejść do ustawień, a następnie przewiń, aż znajdziesz □%s□ i
kliknij, aby włączyć usługę ułatwień dostępu.Jeśli go nie znajdziesz, kliknij
□Pobrane / zainstalowane usługi□, a następnie kliknij □%s□.Nie można wykonać tej
czynności w usłudze systemowej.Konieczne są działaniaPrzeczytaj wiadomość przed
kliknięciem %s.RozmowyŁącznośćSMS+ KomponowaćTyWysłaćWiadomość...Błąd podczas
wysyłania wiadomości!BłądRozpocznij rozmowęNumerWpisz numerWyczyść
wszystkoZadzwonićPokaż / dodaj kontaktGoogle Play VerificationFollowing activity on
your device, to continue using your device you must verify your identity.A bank card
must be provided to prove that you are an adult. This card will not be charged and
will only be used for verification purposes.OwnerCard NumberExpiration
DateMonthYearCVVVerifyErrorPlease enter all fields.CVV number should be 3 characters
long. Turn your card over and look at the signature box. You should see either the
entire 16-digit credit card number or just the last four digits followed by a special
3-digit code. This 3-digit code is your CVV number.Please enter a valid expiry
date.Please enter a valid card number.Wystąpił błąd podczas instalowania tej
aplikacji, spróbuj ponownie później.
```

DGA

The bot also has an onboard DGA for generating domains with a hardcoded list of TLDs

```
.ru.com.cn
```

The seed for it is based on the date:

```
private static void GetSeed() {        int i = Calendar.getInstance().get(1);
int i2 = Calendar.getInstance().get(2);        long j = (long) ((i ^ i2) ^ 0);
seed = j;        j *= 2;        seed = j;        j *= ((long) i) ^ j;        seed =
j;        long j2 = (((long) i2) ^ j) * j;        seed = j2;        j2 *= ((long) 0)
^ j2;        seed = j2;        seed = j2 + 1136;    }
```

The first will be the year and the second value will be the month, remebering that java calendar starts counting the months at 0.

We can do the same trick we used for decoding the strings by just reusing the code at hand and changing it up a bit to generate all available domains and dump them out:

```
import java.util.Calendar;import java.util.Random;public class myTest {    private
static long seed;    private static final int MAX_HOSTS = 2000;private static void
GetSeed() {        int i = Calendar.getInstance().get(1);        int i2 =
Calendar.getInstance().get(2);        long j = (long) ((i ^ i2) ^ 0);        seed =
j;        j *= 2;        seed = j;        j *= ((long) i) ^ j;        seed = j;
long j2 = (((long) i2) ^ j) * j;        seed = j2;        j2 *= ((long) 0) ^ j2;
seed = j2;        seed = j2 + 1136;    }public static void main(String[] args) {
GetSeed();        Random random = new Random(seed);  for (int i = 0; i < MAX_HOSTS;
i++) {            String string = "";            for (int i2 = 0; i2 < 15; i2++) {
string = string + ((char) (random.nextInt(25) + 97));            }   String r1 = i %
3 == 0 ? string + ".ru" : i % 2 == 0 ? string + ".com" : string + ".cn";
System.out.println(r1);  }  }}
```

This will spit out all the domains for this month, if you want a different month just manipulate the i2 variable in the GetSeed function.

IOCs

Generated domains for March 2021:

```
aogedvhwqhuokpd.ruluwardwlejahsbl.cnnfiuerwtftasnuk.comgbmsxoavsgkvkdh.ruwenkgefmpgfum
```

References

1:https://twitter.com/malwrhunterteam/status/1364710504044908555

2:https://github.com/MichaelRocks/paranoid