



## The Citizen Lab

**Research Brief**  
November 2013

### *Asia Chats:*

### *Investigating Regionally-based Keyword Censorship in LINE*

Author: Seth Hardy

This report describes the technical details of client-side censorship functionality in the [LINE messenger client](#) for Android, and a method for disabling it. This is the first in a series of research reports analyzing information controls and privacy in mobile messaging applications used in Asia. An introduction to the project can be found [here](#).

On May 20, 2013, Twitter user @hirakujira reported finding a list of [150 blocked words](#) within Lianwo (连我), the Chinese version of the LINE application. This finding was prompted by a [string in the program](#) related to blocking capability. For more information on what specific keywords are being blocked, Jason Q. Ng translated the keyword lists from Chinese to English and describes context behind them. The first keyword list extracted by @hirakujira is described in a [series of blog posts](#) (full list available [here](#)) and the most recent keyword list uncovered by Citizen Lab and translated by Ng is available [here](#).

A more in-depth analysis of the international LINE client is motivated by similar messages such as this one in the application's resource files (found at `res/values*/strings.xml`), which clearly indicates that it is not just the Lianwo version of LINE that censors messages within the application:

```
<string name="chathistory_chinese_user_word_error">This message contains forbidden words and cannot be sent.</string>
```

Reverse engineering the LINE application reveals that when the user's country is set to China, it will enable censorship functionality: it will download a list of censored words from Naver's website, and then disallow sending any messages that contain any of those words, as well as replace those words with asterisks when received. The region data is encrypted in newer versions of LINE, likely due to users changing it in order to get free in-app downloadable content. Fortunately, it is still possible to disable censorship by changing the encrypted region.

The code analysis in this report was performed on LINE v3.8.5 for Android (APK MD5: 56c9076d56cc20f618df83eaf97a52dc), and the behavior was confirmed on an Android device running v3.9.3 downloaded directly from the Google Play store, current as of November 14, 2013. The presence of

copyright functionality has been confirmed as far back as v3.4.2, released on January 18 2013, using APK files found at [AndroidDrawer](#).

## REGION SETTINGS

When LINE is installed, it asks for the user's country and phone number. The service will send a SMS to this phone number with a four digit code for verification. If the country code of the phone number and the region selected do not match, the program will display an error and suggest the user try again later.

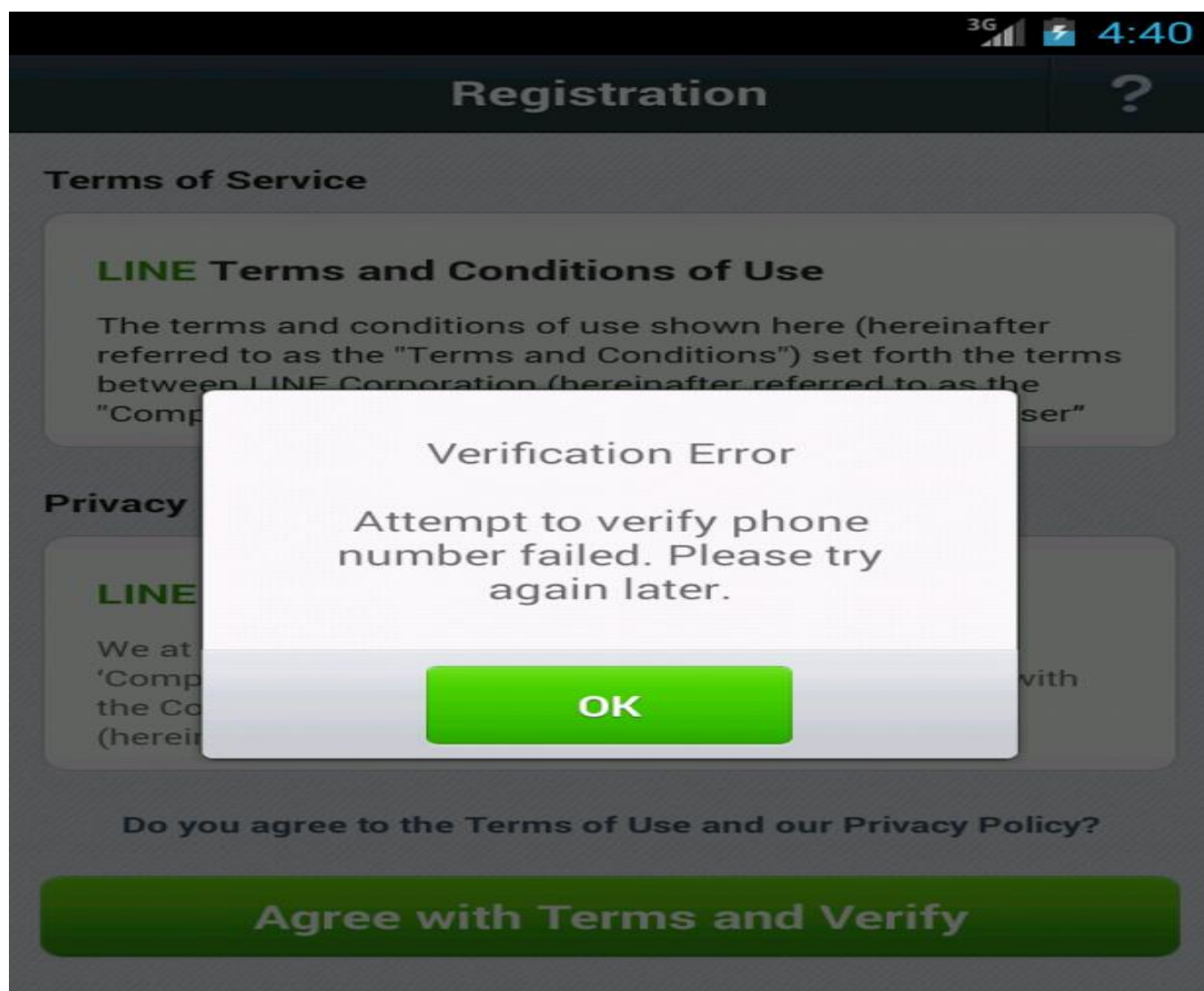


Figure 1: This screenshot shows an error message that is displayed to users in the registration menu if the phone number and the region selected do not match.

This verification makes it difficult for researchers outside of China to test and verify censorship implemented in LINE.

## WORD LISTS

The application uses two URLs to retrieve a word list. The URLs are apparently not entered into the settings database if LINE is installed using a non-Chinese region, but can be found in the file `app-config.properties` at the root of the LINE `.apk` file. These URLs have been present since v3.4.2.

The URLs and their internal descriptions are:

URL\_CHINA\_BAD\_WORDS\_INFO = `http://line.naver.jp/app/resources/bwi`

URL\_CHINA\_BAD\_WORDS = `https://line.naver.jp/app/resources/bwraw`

The file `bwi` has information on the version of the bad words file, containing a string that looks like this:

21,4320,v21, One interesting thing to note is that the file is not directly accessible over HTTP as referenced in the above URL without a user agent string being set to “Android Mobile LA/xx”, where `xx` is a location code. It is, however, accessible via HTTPS without the UA string:

```
$ curl http://line.naver.jp/app/resources/bwi
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="http://line.naver.jp/en">here</a>.</p>
</body></html>$ curl -A 'Android Mobile LA/UK' http://line.naver.jp/app/resources/bwi
21,4320,v21,$ curl https://line.naver.jp/app/resources/bwi
21,4320,v21,
```

The raw bad words file is not actually hosted at the above URL; the filename above has the version number appended to it. For example, the `v21` file is hosted at `https://line.naver.jp/app/resources/bwraw.v21`. As of October 31, 2013, the files for `v21` and `v20` are available. The file can be accessed over HTTPS without the user agent string described above, and over HTTP with it.

The bad word files are Base64 encoded and encrypted using AES in cipher block chaining (CBC) mode with PKCS#7 padding. Decryption uses a static key stored in the binary:

```
private static final byte[] key = {
76, -86, -111, 47, -128, -21, 62, -44, 4, -91, 44, 60, 72, -46, 91, -42, -9, 46, -127, -110, -37, 85, -98, 73, -86,
27, -103, 103, -25, 81, 117, -89
}; public static String decrypt(String encryptedFile) { try { byte[] encryptedBytes =
Base64encoder.decode(encryptedFile); SecretKeySpec secretKey = new SecretKeySpec(key, "AES");
```

```
Cipher c = Cipher.getInstance("AES/CBC/PKCS7Padding");
c.init(Cipher.DECRYPT_MODE, secretKey, new IvParameterSpec(new byte[16])); String decryptedWords =
new String(c.doFinal(encryptedBytes), "UTF-8"); return decryptedWords; }
```

For reference, the 256-bit AES key is: **4c aa 91 2f 80 eb 3e d4 04 a5 2c 3c 48 d2 5b d6 f7 2e 81 92 db 55 9e 49 aa 1b 99 67 e7 51 75 a7**

Decrypted versions of these files are available here: [\[v20\]](#) and [\[v21\]](#).

Once downloaded, the bad words file is stored in the application’s cache directory as `cbw.dat`. If this list is

unavailable, LINE will default to using a smaller internal list:

```
private static Pattern g()
{
File localFile = new File(n.b().getApplicationContext().getCacheDir(), "cbw.dat");
if (!localFile.exists())
try
{
Pattern localPattern2 = getInternalBadWords()1;
return localPattern2; } catch (bwd localbwd2) { a("failed getDefaultWords", localbwd2); return null; }
```

The application also sets preference information on the downloaded word lists. The delay time of 10800000 milliseconds is equivalent to 3 hours:

```
static void setWordListPreferences()
{
long l = System.currentTimeMillis();
SharedPreferences.Editor localEditor = bgb.a(bga.h).edit();
localEditor.putLong("word_list_update_delay", 10800000L); localEditor.putLong("word_list_update_date",
l); localEditor.putLong("word_list_version", 0L);
localEditor.commit(); }
```

The internal list of censored words is also Base64 encoded and encrypted using the same key, and has remained unchanged since its introduction in v3.4.2:

```
5mxphvUu6sGOrG+Qw8EsqIMmx+kNh11mTfMLXZgNmxLfH4grWErsAEsqw+j34zUbdxDUiitSMr77CB
wBdEojcKjIHvdBPtRFduMb5TgvBjbtzmlXHN+UmUsDVmto7rMdCwP6+/AEzLmKB6GsAEs+3Q8xEtejC
qdMnuU262feg4m5OrNxlgX3wpifO/+9Q9lm4KzHAhs7ZSGCOZbE2j6wPOTTMA6TXh9G1gA560ZjGnu
+tLPJStUd08Kkw+AQvLI3ZSC3CexDT83D+FK1MYAUmfc6BIGH/AalsSszJcLyatT7hRUpeNI0/3i+0z2vtE
ZxE06SU4mempEnR9ErjGWLpUIbNdWxFVf32Le1UaLE000+fL0y06lopoFyGIjELHg+dbtmGFRMpxhmZ
auEFmkIeElCVctonsX1866+c3qXVfDnXZg30BXb9Ocrytti5C81/P9M0yDrbtDod2IKX1k39IDrsdqTPUC3+
TYyuAOACGnAN/FNkRQmeBbhzyY7Du2+F2fLm8QIFhm2MczaHyj9aDjMfUAEukNDtYi+Jj3MS+h8/F9
FW3gMHjneviLWBBefRJBbrxRvUKDtvS6vrMO/J2+nxiz/3dBgkG2kXFR6Y9PfV7X0fmBW/ZZLM3MM
Uk8LMmAJIuhcaiKhaHg4721pXqT7v7U2FmHLLiKJiev4a0IG+bxg9nM2m9IcDRSq7Uca08JefErOSSUOW
UoeWhNTtr1dhu07HgbhBaHxdHvurl9Ld+FHikrJZNbZenuNK5+xOrmBVemDc8untuB5IhotpDU3yAnZN
mvZSDJ00/PH05jOCbf4SgW+rkVOjsgodSpljGar7OBSfGR+hMujN4gay3hHm9Ygg5zN8LIME31CcADIFR
Cix0p3b3VxMIAvBHSRItE/B/QwxKsG/IY7y2LCu6wJrctgfPYW/px2ItSOnd4LtzEpCBhQT3/CZbO6c0OL
WaDPufjnZYuyEXyA6677+aMcIAA0RpInRMmzwYg7eG4PnuygTgDWhm6mc5C8Raz2teUysHhsIP26WA
jdBQ+IPWgRvKLDzgpKaOI+moYqXarkTRBBdtMEOp4d4+sU953rE4i8sWtVHSzpV9sprJ5FgdayGOleKn
Zoy9GX1138ey983hX8+LnMsyVKpjw8I9NWCfDknigP9QvAAyADAqE0g4Y0oD3G1YAikkHCV0Foc8IU
ww++9XpECxlED7znaF9USMeXV+ZyzRjztUk/kg==
```

For a decrypted version of this list, see [\[here\]](#).

The bad words check is only performed if the region stored in LINE's settings is "CN":

```
public static final boolean c()
{
add.a();
if (add.c());
```

```
String str;
do
{
    return true;
    add.a();
    if (jl.d(add.a(cya.a)))
        return false;
    af localaf = b();
    if (localaf == null)
        break;
    str = localaf.c();
}
while ((jl.d(str)) && (
Locale.CHINA.getCountry().equalsIgnoreCase(str)
)); return false; }
```

Users in other regions are not subject to censorship, and changing the region code from CN to any other region should prevent the censorship functions from being invoked.

## CHANGING REGIONS

The LINE installation's region is stored in a SQLite database on the Android device, in the file `/data/data/jp.naver.line.android/databases/naver_line`.

To access the database, the Android device must be rooted. The database can be opened using either a SQLite GUI program such as [SQLite Editor](#), or by using the command line program `sqlite3` from a shell on the Android device. This example uses the command line program, which is preinstalled on many Android devices.

Listing the tables shows the type of information stored in the database:

```
root@android:/data/data/jp.naver.line.android/databases # sqlite3 naver_line
SQLite version 3.7.16 2013-03-18 11:39:23 [www.ptsoft.org] [www.ptdave.com]
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
android_metadata    contacts            permanent_tasks
buddy_detail        email_recommend    push_history
channel             esk_history        setting
chat                groups             sns_friends
chat_history        membership         sticker
chat_member         more_menu_item_status sticker_package
chat_notification   noti_center        version
```

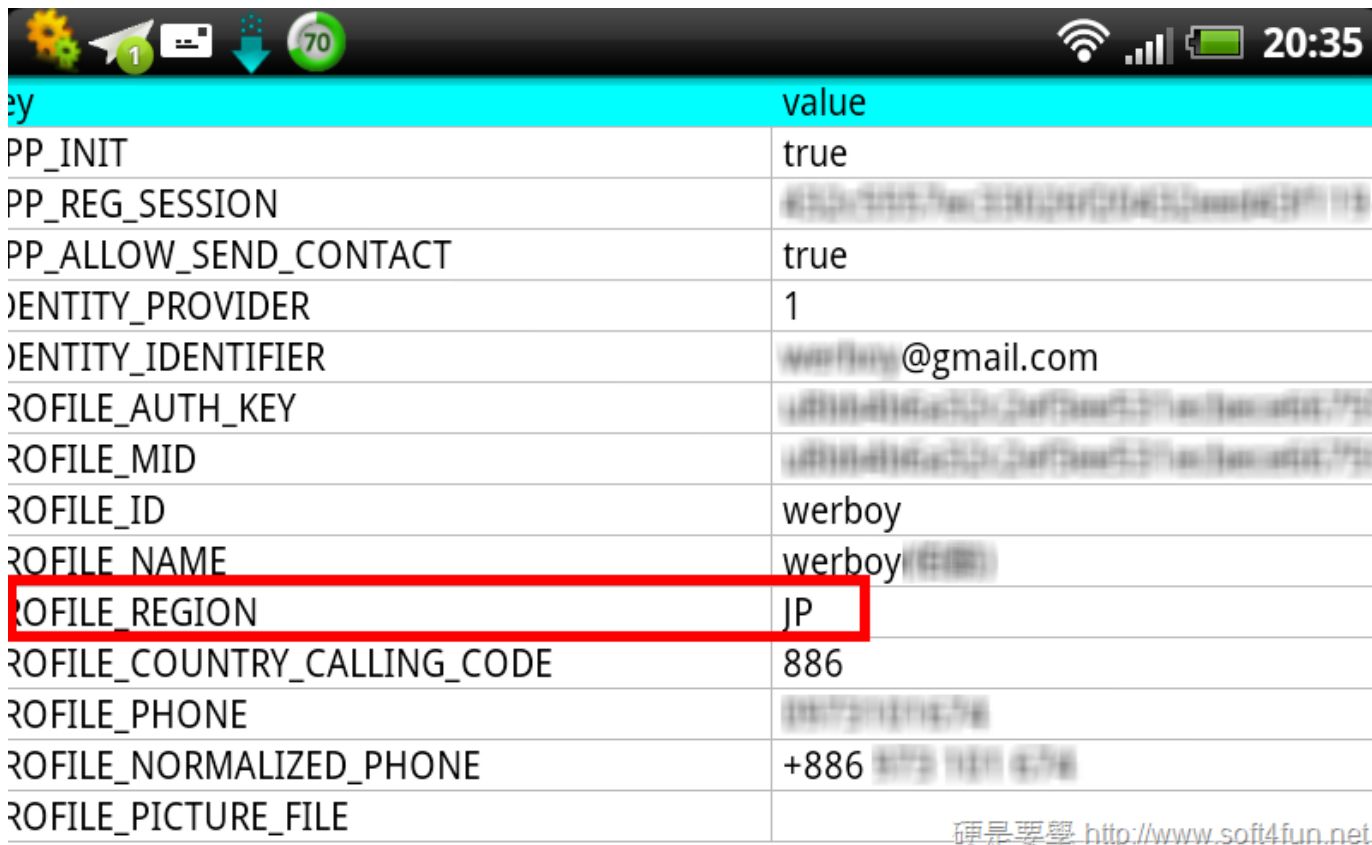
The database setting contains the region as well as other user account details. One interesting thing to note is that some of these values are Base64<sup>2</sup> encoded:

```
PROFILE_NAME|IUrTpRKtKUwg8riwl6b5Dw==
PROFILE_REGION|TnPF6Y1PAAuVhWt17IMRtA==
PROFILE_COUNTRY_CALLING_CODE|2BP0qhgFrPzM0qPSibkoKw==
```

PROFILE\_PHONE|kE/KhY6QGb/B2yuqPoQn3i==

PROFILE\_NORMALIZED\_PHONE|+wJ7t10qbHqR5yRezaJd0c==

This change is a recent development: as of mid-2012, users were changing their region in order to download free stickers and in-app enhancements only available in other areas. Two examples of this can be found [here](#) and [here](#). Compare the Base64 values above to a screenshot of the database from a blog post on July 25, 2012:



key	value
PP_INIT	true
PP_REG_SESSION	[REDACTED]
PP_ALLOW_SEND_CONTACT	true
IDENTITY_PROVIDER	1
IDENTITY_IDENTIFIER	werboy@gmail.com
PROFILE_AUTH_KEY	[REDACTED]
PROFILE_MID	[REDACTED]
PROFILE_ID	werboy
PROFILE_NAME	werboy (作圖)
<b>PROFILE_REGION</b>	<b>JP</b>
PROFILE_COUNTRY_CALLING_CODE	886
PROFILE_PHONE	[REDACTED]
PROFILE_NORMALIZED_PHONE	+886 [REDACTED]
PROFILE_PICTURE_FILE	[REDACTED]

硬是要學 <http://www.soft4fun.net>

Source: [Frank's Blog](#)

When unencoded, these settings do not show up as the expected plaintext; the reason they are Base64 encoded is not just for obfuscation, but also to convert encrypted binary data into printable characters. It is likely that the region is stored encrypted to prevent people from changing their region to have access to more downloadable content.

For example, decoding PROFILE\_REGION gives 16 bytes of binary data:

```
00000000 26 c8 9b 97 37 68 d1 8c 83 42 43 f5 84 f2 83 0c |&...7h...BC....|
```

By looking through the decompiled binary, we can see that this data is encrypted. Like the bad words file, it uses AES, but with slightly different options.

Unlike the static key used for the bad words file, the settings are encrypted using a key derived from the android\_id value of the phone. More information on this value can be found [here](#), and the value can be determined using a program such as [Android Id Info](#).

```
public final byte[] generateKey(long magicNumber)
```

```
{
    String str = Settings.Secure.getString(f.c().getApplicationContext().getContentResolver(), "android_id");
```



```

if (str == null) str = ""; int i = str.hashCode();
return generateKeyFromHash((byte)i, magicNumber);
}

```

This function generates the AES decryption key by taking the android\_id identifier, hashing it using a standard Java function, then applying another transformation algorithm to it to derive the key. The magicNumber value is hardcoded into the program and is equal to 15485863.

AES decryption is performed in electronic codebook (ECB) mode with no padding:

```

public static byte[] decrypt(byte[] key, byte[] ciphertext)
{
    SecretKeySpec sk = new SecretKeySpec(key, "AES");
    Security.addProvider(new BouncyCastleProvider());
    try {
        Cipher c = Cipher.getInstance("AES", new BouncyCastleProvider());
        c.init(Cipher.DECRYPT_MODE, sk); return c.doFinal(ciphertext); }
}

```

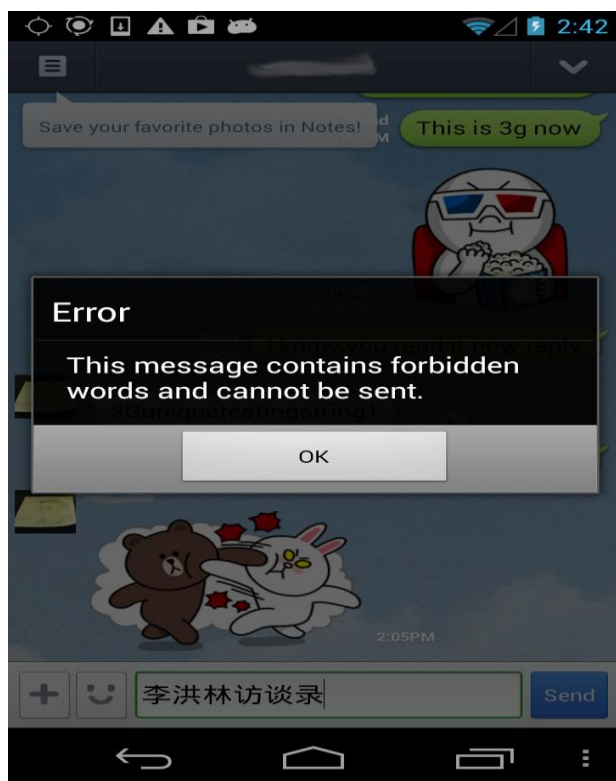
Using our example from above, we can decrypt the string to determine the region:

```

your region is : CN
decrypted region = 43 4e
encryption key = 3b 93 dd 57 52 b8 38 7e 07 e8 b0 0f c3 f5 89 12
encrypted region = 4e 73 c5 e9 8d 4f 00 0b 95 85 6b 75 ec 83 11 b4
encoded region = TnPF6Y1PAAuVhWt17IMRtA==

```

Changing the PROFILE\_REGION value in the setting database to TnPF6Y1PAAuVhWt17IMRtA== will change the region to China. This change can be confirmed by looking at the terms of service or privacy information in the LINE application. It can also be confirmed by attempting to send a keyword from the bad words list:



**Figure 2:** This screenshot shows a LINE user with region set to China receiving an error notification after attempting to send keywords on the blocked keyword list. This error message indicates client-side outbound keyword censorship is present in LINE.

To turn off censorship, we want to go in the reverse of the above example. On a China-region installation, changing the region code from CN to another region (e.g. CA or US) will disable censorship functions. To assist users we are releasing [LINE Region Code Encrypter Tool](#) developed by Seth Hardy and Greg Wiseman (Senior Data Visualization Developer, Citizen Lab) for changing regions in the LINE client to disable regionally-based keyword censorship in the application.

## UPDATE - NOVEMBER 19TH, 2013

An [article](#) by Kaylene Hong of TheNextWeb provides a response from LINE Corporation regarding the presence of regionally-based keyword censorship in their chat program: “A LINE spokesperson drew a clear line between Lianwo and Line, emphasizing that the Chinese version of LINE is different from the global version.” However, all of the censorship functionality described in this report is implemented in the global version of LINE, and it is unclear whether the Lianwo version of LINE has additional functionality or is merely rebranded.

## Inbound Censorship

Incoming messages in LINE are also subject to censorship when the region is set to China. The LINE client will replace any words found in bad word files (internal or downloaded) with asterisk characters. As with outbound messages, inbound censorship requires a strict match, and can be bypassed by adding spaces or other punctuation.



On a phone with LINE installed where the region is set to Canada, messages received which contain keywords on the censorship list come through properly:



**Figure 3: This screenshot shows a LINE user with region set to Canada successfully receiving two messages with keywords on the censored keyword list.**

But if the application's region is set to China, these words are blocked. The screen shot below shows that messages with the keyword “李洪林访谈录” (Li Honglin Interview) are replaced by a series of asterisks, but when the same keyword is sent with periods separating the individual characters the message is not blocked.



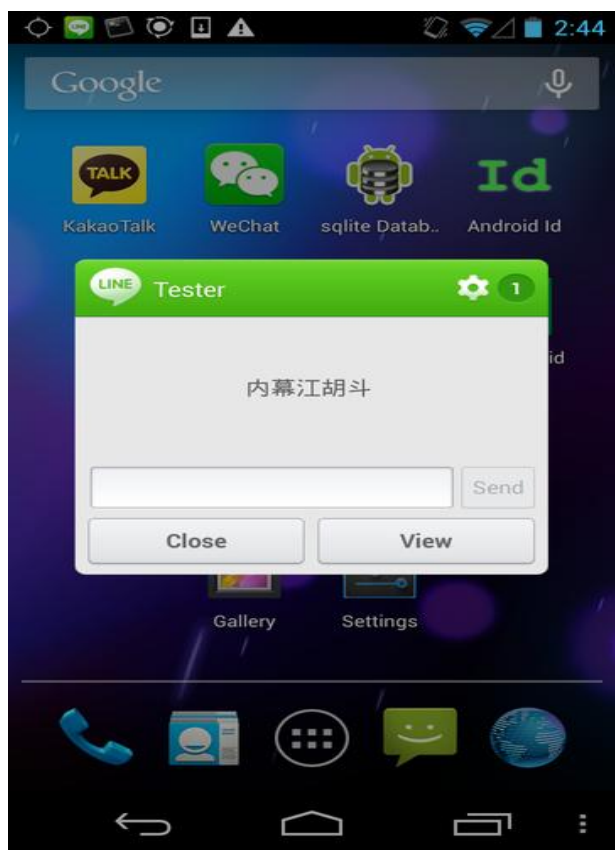
**Figure 4:** This screenshot shows a LINE user with region set to China receiving messages with keywords on the censored keyword list as a series of asterisks.

This method of censorship is particularly problematic, because to the message sender, it appears as if the content was received properly.



**Figure 5:** This screenshot shows the view of a LINE user with region set to Canada when sending censored keywords to a user with region set to China. From the Canada-based user's point of view the messages appear to have been received and read by the China-based user.

The code that does incoming censorship is interesting. It uses a regular expression to match incoming messages against the bad words file(s), and replaces the characters individually with asterisks. However, because of how Android programs run, the code takes effect when the string is displayed in the main program, but not in notifications. This function allows the recipient to see the censored message in a preview notification:



**Figure 6:** This screenshot shows a message preview notification for a LINE user with region set to China receiving a message with censored keywords.

**About The Author** Seth Hardy is a Senior Security Analyst at the Citizen Lab, Munk School of Global Affairs, University of Toronto. Prior to the Citizen Lab, he worked for a large anti-virus vendor. Seth has worked extensively on analysis of document-based malware and AV evasion methods. Other areas of experience include: provably secure cryptography, random number generators, and network vulnerability research. Seth has spoken at a number of security conferences including Black Hat, DEF CON, SecTor, and the CCC. He holds degrees from Worcester Polytechnic Institute in Mathematics and Computer Science.

## FOOTNOTES

1. Names such as this are not the original names assigned to the functions, which were removed from the program when it was compiled. These informative names are assigned by an analyst while exploring the program.
2. Values we are not looking at in this analysis have been changed to protect user data.