Quarkslab's blog

Quarkslab's website

SOCIAL

atom feed

twitter

github

CATEGORIES

Android

Android, ReverseEngineering

Challenge

Cryptography

Development

Exploitation

Fuzzing

Hardware

Hardware, ReverseEngineering

Kernel Debugging

Life at Quarkslab

Maths

Obfuscation

PenTest

Program Analysis

Programming

ReverseEngineering

Software

Vulnerability

TAGS

How a Security Anomaly was Accidentally Found in an EAL6+ JavaCard

Date Tue 12 May 2020 By Philippe Teuwen Category Hardware. Tags javacard timing attack iso7816

In the context of the Inter-CESTI 2019 challenge, we "accidentally" found a timing difference disclosing the length of a PIN handled via the standard `OwnerPIN.check` JavaCard API. Here is the story.

Context

Quarkslab is one of the 10 ITSEF (Information Technology Security Evaluation Facility, CESTI in French) licensed by the ANSSI's Centre de Certification National (CCN, French for National Certification Body) for performing CSPN (Certification de Sécurité de Premier Niveau, French for First Level Security Certification) evaluations in the software domain.

My apologies for this stodgy introduction. Basically, we make from time to time official security evaluations and, to keep track of the ITSEFs performances, the ANSSI organizes every year an "Inter-CESTI" challenge. In this context, for the last challenge started in 2019, the ANSSI chose one of its own "products" to be evaluated, the open-source and open-hardware Wookey project [1], presented at SSTIC 2018 [2].

The interesting point of this edition is that the barriers between the software and hardware domains were blurred to push ITSEFs out of their comfort zone. Test plans were established for each ITSEF to spread the load with a mix of software and hardware.

By the way, methodologies and most interesting results will be presented at SSTIC 2020 [3] in a joint work of the 10 ITSEFs and the ANSSI. We warmly invite you to at least have a look at the joint article which will be available in English.

So, this is how we ended up with the *timing attack on UserPIN validation* item in our test plan.

Clearly, at this point, the goal was not to ask a software ITSEF to perform a hardware attack in the hope of finding a vulnerability in a card already evaluated as Common Criteria EAL6+ [4] by some hardware ITSEF. The intention was only to make sure no trivial mistake was present in the Wookey applet and to see if we, software ITSEF, would be at ease to mount hardware attacks that don't require pricy hardware equipment.

UserPIN validation

We won't cover the Wookey usage extensively, simply consider the Wookey as an embedded device with a touchscreen that you use to enter a PIN code validated by a JavaCard applet installed on a JavaCard inserted into the Wookey.

A review of the Wookey applet source code showed that their `check_pin` method [5] properly uses the `OwnerPIN.check` JavaCard API function [6] to validate a PIN. Even the length of the ISO7816 (APDU [7]) command is kept constant for all supported PIN lengths so an attacker eavesdropping on the ISO7816 communication, encrypted in a *secure channel*, can't get any information.

At this stage, there was no indication that a timing attack on UserPIN validation could be possible.

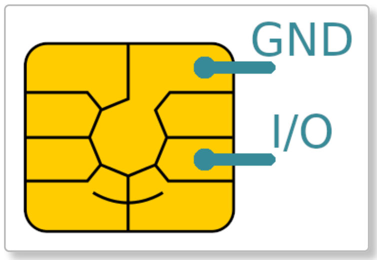
Nevertheless, in the Inter-CESTI context of demonstrating that we, software ITSEF, are able to conduct low budget hardware attacks, we moved on to the practical aspects. As a first step, we modified the applet to test more easily PIN attempts without a need for mounting first a *secure channel*.

The Wookey demonstration setups we got for the evaluation were using (arbitrarily) J3D081 NXP cards (featuring JCOP2.4.2R2 [8]) but, when we wanted to conduct the experiments on the modified applet, we only had more recent J3R200 cards lying around (featuring a SmartMX3 P71D321 [9] with JCOP4). So we conducted the experiments on a J3R200, not that it should matter, right?

Physical setup

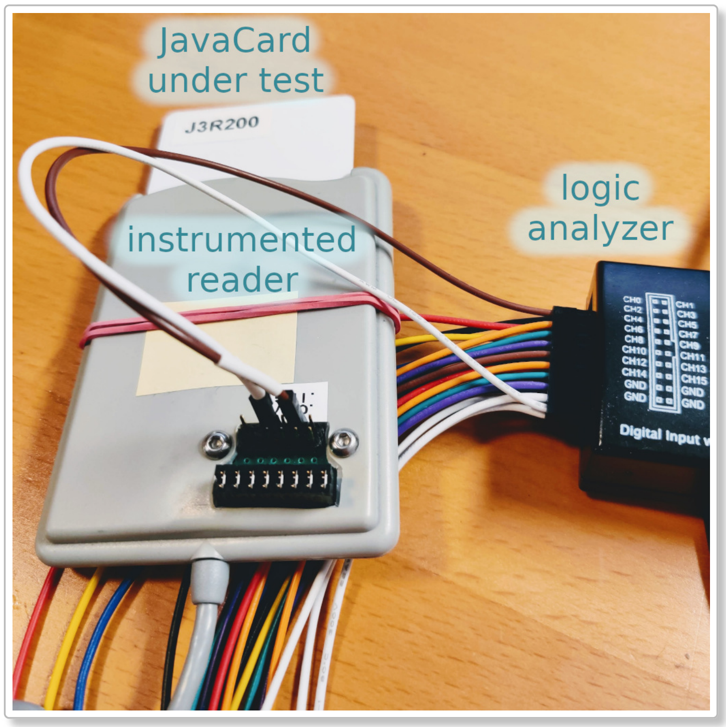
Conducting such experiments on contact cards doesn't require much hardware. You need an ISO7816 reader and a way to measure the response time precisely enough. Measuring response time on the host is not precise enough as USB schedules the transmission of packets in one millisecond time cycles or frames so we use a simple logic analyzer.

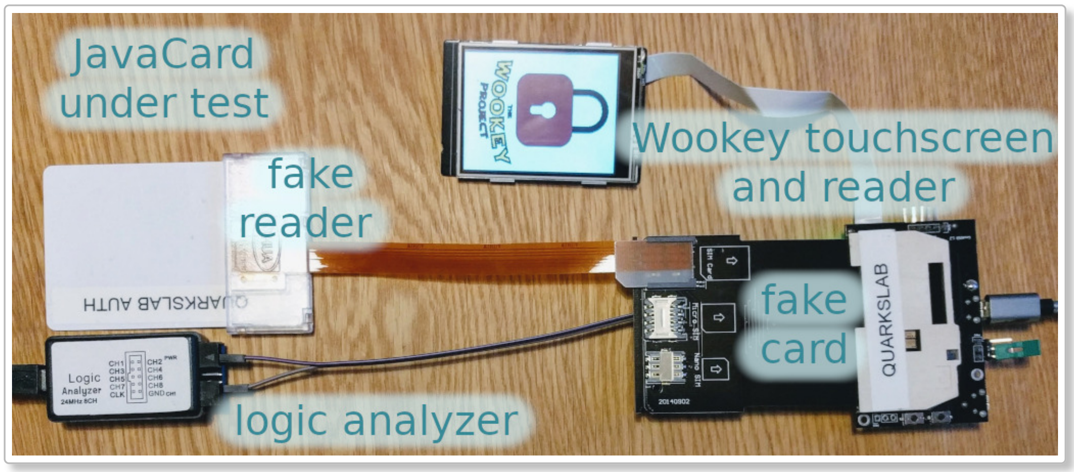
ISO7816 communication is quite similar to an 8-bit UART with even parity and one stop bit, working at a baudrate depending on the frequency of the provided clock signal, and with one single I/O pad for both RX and TX.



But actually you don't care about all these details, you just need to know where to tap on the I/O and to measure the response time of the card.

Using an external analyzer requires to be able to access the I/O pin, which can be achieved either by combining cheap adapters to extend the physical connection out of the reader, or by directly tapping into the reader with some hardware modifications.





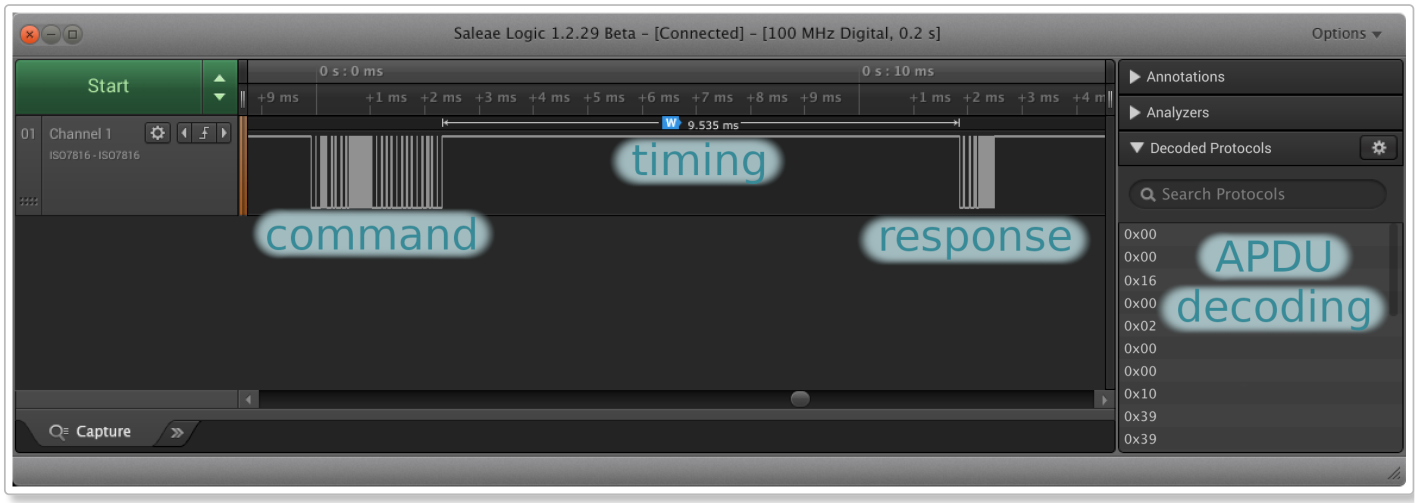
In our setup, we chose to connect a logic analyzer to the I/O and Ground pins of the card and we controlled both the logic analyzer and the card reader from a Python script. Indeed a Python module called `saleae` [10] is available to control its GUI [11]. Maybe a similar setup could be achieved with the open-source Sigrok project [12], we didn't explore that path.

The script was very basic:

- Connect to the card, get its ATR
- Setup the card with a test PIN, e.g. 1234
- Start a capture on the logic analyzer at 100 Ms/s

- Send the APDU to check a PIN, e.g. 9999
- Stop and save the capture
- Extract timings from the capture and keep the first one which is *long enough* to be the response time (the other ones are the timings between individual bits)

Here is an example of such capture.



Testing

Firstly, we wanted to see if there was a timing difference depending on the PIN length. So we configured a 4-digit PIN (1234) and we tried to authenticate with a set of wrong PINs of various lengths (9, 99, 999, ...). We repeated the operations for 8-digit and 12-digit PINs.

	Response delay in ms		
Tried PIN length	Configured PIN length = 4	Configured PIN length = 8	Configured PIN length = 12
1	9.59842	9.61192	9.60721
2	9.61701	9.60351	9.61363
3	9.64341	9.64240	9.65246
4	9.70651	9.60401	9.61189
5	9.66192	9.64365	9.65071
6	9.64476	9.64452	9.64737
7	9.65265	9.64267	9.65846
8	9.59802	9.68900	9.59189
9	9.61667	9.61141	9.61171
10	9.61275	9.60951	9.61639
11	9.61390	9.60492	9.60722
12	9.61625	9.60976	9.70940
13	9.61140	9.60391	9.60423
14	9.60750	9.60726	9.60414
15	9.61616	9.61841	9.61573
correct PIN	12.15997	12.15688	12.15196

Repeated measures showed some jitter, nevertheless when a wrong PIN was tested *with the correct length*, the response delay was clearly standing out as the slowest one with one single measurement per tried PIN.

So, if enough PIN attempts are allowed by an applet on such card, or if one can make one or two tests between usages by the legitimate owner, one could learn what the correct PIN length is.

To follow the Wookey test plan, we then tested the same attack on the original applet and within the *secure channel*. Response time within the *secure channel* was much longer (about 153 ms) and jitter increased, still on average (we took 3 measures for each test and kept the *middle* one) there was still a timing difference when the right PIN length is used.

The next step was to try different PINs of the correct length. Luckily we didn't notice any timing difference when e.g. the first or the last digit was correct. Therefore this timing attack allows only to recover the PIN length.

Later, we also tried the timing measures on a few other cards, including the J3D081 as used in the Wookey prototype. We discovered the same issue on a J3H081 (JCOP3) with even a much more visible timing difference (16 ms vs. 22 ms). We tested a J3H145 (JCOP3) and the difference wasn't there at all. Older J3D081 and J3A081 seem fine too.

Security discussion and responsible disclosure

These findings are quite strange on such high security products.

Clearly, it shouldn't happen.

Still, it's hardly a practical security issue: on most applications, the PIN length is already known by the context and most applications will tolerate a maximum of three attempts before locking the card. Bruteforces are anyway highly impractical even if e.g. up to 16 PIN tries are tolerated and knowing exactly the PIN length saves only 10% of the bruteforce space (no need to test PINs 0 to 99999 if we know the PIN is 6 digits).

Nevertheless, we decided to contact NXP PSIRT (Product Security Incident Response Team) to discuss this anomaly prior to any publication.

The discussion was open and they checked if it could have an impact on the Common Criteria certification. Their conclusion is that it is not. There is no requirement in the Security Target for the products that would be violated.

Anyway, they agreed these cards shouldn't behave like they do and they'll fix it in future versions of their products.

JCOP identification

We've seen that a J3H145 wasn't affected at all and a J3H081 was strongly affected, while both cards are supposed to differ only by the size. Similarly, initially NXP couldn't reproduce our findings on their J3R200. Actually, behind each generation (e.g. J3A, J3D, J3H, J3R) there are still quite some implementations.

JCOP cards implement a proprietary `IDENTIFY` APDU to extract precise information such as EEPROM ID, Platform ID, Custom Mask ID etc. The command and its interpretation are easy to find on the Internet for the JCOP2.4.x generations (J3A, J3D) [13] and were also part of the `jcshell` tool distributed with the Eclipse IDE. The JCOP proprietary IDENTIFY command APDU is `00A4040009A000000167413000FF00`.

But JCOP3 and JCOP4 don't support this historical `IDENTIFY` APDU (they reply with a `6A82` *File not found*) and we were confused to orient NXP towards the exact model of J3R200 we were using.

Apparently the IDENTIFY command and the answer changed since JCOP3, and after searching for a while on the Internet, we could find it in a public document from NXP used in its certification process, the *ChipDoc P60 on JCOP 3 SECID P60 (OSA) SSCD* [14].

The much sought-after new IDENTIFY command is `80CA00FE02DF2800` and it returns a bunch of TLV-formatted data partly described in the document. May this information help some of you in the future.

Testing it on our J3R200:

```
$ scriptor
Trying T=1 protocol
Reading commands from STDIN
80CA00FE02DF2800
> 80 CA 00 FE 02 DF 28 00
< FE 45 DF 28 42 01 0C 00 01 A1 C2 82 34 97 80 44
93 39 CA 02 08 00 00 00 00 00 00 00 01 03 18 4A
33 52 33 35 31 30 31 46 41 39 45 30 34 30 30 DD
09 84 59 3B 00 48 EF 05 01 00 07 01 01 08 08 2E
5A D8 84 09 C9 BA DB 90 00 : Normal processing.
```

Following the above-mentioned document to parse the answer, we learn that *Platform ID* is `J3R35101FA9E0400`

It's an EAL6+ chip, according to its *JCOP 4 P71 Certification Report* [15] and the Platform ID corresponds apparently to a JCOP 4 P71 v4.7 R1.00.4.

The other affected card, the J3H081, has as Platform ID `JxHyyy00E4D80300` while the J3H145 has as Platform ID `JxHyyy0019790400` which explains their JavaCard OS implementation was differing somehow.

Conclusion

We've seen how one can easily tap into an ISO7816 communication to measure precisely the response times of a smart card. Such example of a very low budget hardware attack shows how the distinction between the historical software and hardware domains of ITSEFs tends to blur.

Our tests revealed an information leakage on J3R200 and J3H081 JavaCards: a simple test of a few different PIN lengths allows to know the correct PIN length. Note that even if it shouldn't happen in such security product, this is hardly a practical security issue as the number of PIN attempts is typically very limited in real applications.

It was also quite a turn of events: if we had executed the test plan with the initially foreseen J3D081, the tests would have failed as expected and we would never have tested other cards.

So, don't be intimidated by high certifications of products and still try out stuff by yourself, who knows...

The Inter-CESTI challenge on the Wookey was really fun to do on many aspects. Don't miss the joint talk [3] at SSTIC 2020!

Thanks to all our Quarkslab colleagues who proofread this article.

Timeline

- 2020/03/16: => we alert NXP PSIRT about J3R200
- 2020/03/17: <= NXP cannot reproduce, they ask for IDENTIFY info
- 2020/03/18: => we can only give partial info (CPLC)
- 2020/03/23: => we send our Wookey CSPN report (RTE) to ANSSI
- 2020/04/13: => we finally find how to IDENTIFY our J3R200 and we transmit the information to NXP
- 2020/04/14: => we test the J3H081 and we alert NXP on that one too
- 2020/04/16: <=> we have a call with NXP: they acknowledge the issue and ask some time to check the impact on certification
- 2020/05/04: <= NXP allows us to publish
- 2020/05/05: => we share the status with ANSSI and ask for permission to cite the Inter-CESTI context
- 2020/05/11: <= ANSSI allows us to mention the inter-CESTI challenge context for our results

References

[1] https://wookey-project.github.io/
[2] Ryad Benadjila, Mathieu Renard, Philippe Trebuchet, Philippe Thierry, Arnauld Michelizza, and Jérémy Lefaure. *WooKey: USB Devices Strike Back* https://www.sstic.org/2018/presentation/wookey_usb_devices_strike_back/
[3] *(1, 2) Inter-CESTI: Methodological and Technical Feedbacks on Hardware Devices Evaluations* https://www.sstic.org/2020/presentation/inter-cesti_methodological_and_technical_feedbacks_on_hardware_devices_evaluations/
[4] https://en.wikipedia.org/wiki/Evaluation_Assurance_Level
[5] https://github.com/wookey-project/javacard-applet/blob/master/src/wookey/common/WooKey.java#L171
[6] https://docs.oracle.com/javacard/3.0.5/api/javacard/framework/OwnerPIN.html
[7] https://en.wikipedia.org/wiki/Smart_card_application_protocol_data_unit
[8] https://en.wikipedia.org/wiki/Java_Card_OpenPlatform
[9] https://www.nxp.com/products/security-and-authentication/security-controllers/smartmx3-p71d321-secure-and-flexible-microcontroller:SMARTMX3-P71D321
[10] https://pypi.org/project/saleae/
[11] https://www.saleae.com/downloads/
[12] https://sigrok.org/
[13] https://wiki.nfc.im/books/%E6%99%BA%E8%83%BD%E5%8D%A1%E6%89%8B%E5%86%8C/page/nxp-jcop-javacard
[14] https://www.ssi.gouv.fr/uploads/2017/09/anssi_cible2017_43en.pdf
[15] https://www.commoncriteriaportal.org/files/epfiles/certification-report-nscib-cc-180212-cr2.pdf

Comments