REPORT, IMRICH NAGY, DANIEL RYCHLÝ

APPROACH

We examined the code manually. The check by hand yielded good results and we were ultimately able to extract the PIN from the card, so we did not try additional inspection using static or dynamic analysis. There is little code to analyze anyway and almost no "real" Java card code.

RESULTS - THE GOOD

The implementation was easily comprehensible and the phase 2 report of the target team could be used as a good form of documentation. The target team chose SPAKE2 as a protocol to authenticate the ECDH shares. It is a good choice since the protocol (theoretically) protects against offline brute-force attacks against the PIN, which is desired due to its inherently low amount of entropy. The target team used GitHub Issues and Projects to track the progress (although to a limited degree). Target team members had their separate forks of the repository, where they developed their implementation and then submitted the changes using pull requests, discussing the changes in the comments.

RESULTS - THE BAD

The code is left without documentation. The few (and quite long) methods could use Javadoc comments. Some magic constants could have been named, mainly instruction class bytes. Numerous formatting mistakes are present, like missing spaces and method names not following Java coding standards. Some tools should have been used to prevent these mistakes, as less readable code could result in more buggy code.

A lot of applet resources are allocated outside the installation phase. Several arrays that could be allocated in RAM are allocated in EEPROM. New objects are being created in methods rather than reusing the old ones, which would result in poor performance on a real smart card.

One of the bigger issues was the usage of full-blown Java libraries in the applet. Over half of the imports in the applet are Java libraries, that have no implementation for Java Card. The applet would not compile as a regular Java Card applet. The RNG and EC implementations, for instance, were implemented using these "big" libraries rather than Java Card ones and a significant amount of code is developed as a regular Java application (not like an actual applet). Therefore, the target project contains almost no real Java Card code, which was the main part of the project, which we struggled with the most and we were looking forward to reviewing the most as well.

Symmetrically, some Java Card libraries are used to calculate crypto in the client application. That is not a good practice, as these libraries are provided by JCardSim and could be just placeholders for development purposes (working correctly, but prone to side-channel attacks), because these libraries are commonly implemented by smart card vendors using specialized hardware coprocessors. Even if the libraries just wrap the common Java crypto libraries, those should be directly used for the PC client application instead, because they are made specifically for that purpose, their APIs are better suited for regular Java applications and classic Java developers understand them better.

Furthermore, the implementation does not contain any secure channel management. The secure communication is shown using a "hard-coded" method that manages the secure channel, key updates, encryption and decryption all by itself. There is no secure channel wrapper functionality, so any new instruction that would need to utilize the secure channel would need to implement the secure channel functionality again.

The session cannot be explicitly ended. No session management is implemented. Commands can be processed by the card out of order. Many checks (PIN, invalid attempts, the current state of the session) are not present.

CHOSEN VULNERABILITIES

PIN EXTRACTION EXPLOIT

Problem: Unlimited PIN attempts card-side
Severity: Critical, **Risk (probability):** High, **Business impact:** High, **Effort to fix:** Low
Prerequisites: Gain possession of the card, or ability to send/receive data to/from the card.
Description: The card never checks whether the PIN was correct when starting the ECDH session. Therefore, the attacker can try to establish a session with all possible PINs and then simply verify whether the PIN was correct. This way, the PIN can be directly extracted from the card in almost no time.
Remediation: The card should check the correctness of the ECDH session every time and have its internal counter or update OwnerPIN with an incorrect value.
GitHub issue link: https://github.com/mgrabovsky/pv204-spring2020-project/issues/11

PIN HASH EXTRACTION FROM MEMORY

Problem: PIN hash is kept in RAM on the PC client even when not needed/used.
Severity: Medium, **Risk (probability):** Medium, **Business impact:** High. **Effort to fix:** Low
Prerequisites: Attacker needs to get access to the machine or ability to scan the memory
Description: The PIN hash is stored into an attribute array and never cleared. Once the PIN is used, it can be retrieved at any time, provided the attacker can access the contents of the RAM. Computing the PIN from the hash is then trivial.
Remediation: Use locally defined arrays to store the PIN hash and pass it to the `ecdhchannel()` method as an argument.
GitHub issue link: https://github.com/mgrabovsky/pv204-spring2020-project/issues/12

UNLIMITED SESSION LENGTH + SESSION KEY EXTRACTION FROM THE MEMORY

Problem: ECDH session is never terminated by the card.
Severity: Medium, **Risk (probability):** Low, **Business impact:** High, **Effort to fix:** Medium
Prerequisites: Attacker needs to get access to the machine or ability to scan the memory
Description: Single ECDH session can be infinitely long and the card never requires a new session and the AES key changes predictably. Once it is obtained by the attacker, it can be used to decrypt all the following communication.
Remediation: Limit each ECDH session card-side to a fixed number of messages before the next fresh ECDH session is required.
GitHub issue link: https://github.com/mgrabovsky/pv204-spring2020-project/issues/13

PAKE PROTOCOL NOT IMPLEMENTED CORRECTLY

Problem: Multiple deviations from SPAKE2 are protocol present in the implementation.
Severity: High, **Risk (probability):** Medium, **Business impact:** High, **Effort to fix:** High
Prerequisites: Possession of the card or ability to eavesdrop or spoof APDU packets.
Description: There are several problems with the protocol implementation:
1. It is said that K must not be used as a shared secret directly. It should be rather used together with the protocol transcript to derive a key. The provided paper states it is not secure otherwise.
2. Upon receiving on the card, T should be multiplied by cofactor h and checked against the unit element I of G. Otherwise, it allows for the small subgroup attack.
3. The newest release of the protocol paper could have been used (*Spake2-08* instead of *spake2-04*).
Remediation: Implement the protocol according to the specification.
GitHub issue link: https://github.com/mgrabovsky/pv204-spring2020-project/issues/21

CORRECT APPLET CODE EXECUTION ORDER IS NOT ENSURED.

Problem: APDU commands might be executed out of order
Severity: Medium, **Risk (probability):** High, **Business impact:** Medium, **Effort to fix:** Medium
Prerequisites: Possession of the card or being able to spoof APDU packets.
Description: Applet APDU command execution is not automata-based. Commands can be processed out of order, for example, `aescommunication()` will run even without establishing shared secret, producing an exception. Also, the PAKE protocol demands to execute certain commands in order. This might be used as an attack vector.
Remediation: At the start of each command execution, there should be a check for whether all logically preceding commands were executed and finished successfully. Additionally, the check for correct PIN should be implemented, decreasing the counter at the beginning of `pinandecdhchannel()` to prevent exploiting multiple consecutive runs of the protocol instances covered by this method.
GitHub issue link: **https://github.com/mgrabovsky/pv204-spring2020-project/issues/22**

LACKING FAILURE CHECKS.

Problem: The checks against stored secret failure or incorrect input from the client are insufficient.
Severity: High, **Risk (probability):** Medium, **Business impact:** Medium, **Effort to fix:** Medium
Prerequisites: Possession of the card or ability to spoof APDU packets.
Description: Checksum of stored secrets should be computed at each startup of the card, as bit-flip induced by an attacked or by fault might occur. Also, the card checks for correct input from the client only sporadically, resulting in exceptions, when the input is not correct. For example, the applet keeps running and finishes the exchange example even when the PIN is not correct.
Remediation: Implement various checks to ensure the data received from the client are correct and not malicious.
GitHub issue links: **https://github.com/mgrabovsky/pv204-spring2020-project/issues/18**, **https://github.com/mgrabovsky/pv204-spring2020-project/issues/23**

OTHER SECURITY ISSUES

There is no functionality on the card to terminate the session explicitly. The PIN is stored on the card using an array instead of secure storage like OwnerPIN, which endangers the PIN and the checking for the correct PIN would have to be done manually (much more prone to bugs than `OwnerPIN.check()`). It would have been nice if the card would check at least for PIN like 0000 and refuse it during the installation phase.

There are also other GitHub issues opened by our team members that did not make the cut into this report (https://github.com/mgrabovsky/pv204-spring2020-project/issues).

CONCLUSION

We thoroughly examined the codebase and discussed several security issues, both in protocol implementations and in the practical implementation of the functionality. We compiled a total of 13 issues in the target repository, which summarize the most important security issues we were able to find.

In the end, we were able to extract PIN from the card only by issuing APDU commands, thus defeating the whole system and gaining owner rights to the card. We also proposed several possible attacks on the security protocols and attacks on the PC client machine, where it could be possible to extract either the PIN or the shared symmetric key, thus being able to eavesdrop all the communication and operate the card indefinitely, because the session is never terminated by the card. Additionally, we pointed out several issues with the execution logic in the scope of the applet command processing. Several qualitative issues were found, that would cause the code working improperly on real smart card hardware, even causing the applet code to not compile as a correct Java Card applet at all. The PC client implementation was shown to be lacking in the quality as well.