# PV204 Project Security Analysis

## Introduction and tools used

This is the report of our analysis of NimRo97 repository, specifically the JavaCard Applet and the Java desktop application for establishing secure channel and sending and receiving messages once the secure channel is established. The report first outlines the methods used during analysis, followed by attack vectors that we have considered during analysis but that are successfully protected against by the developers already. Lastly a section on found bugs together with proposed corrective actions is also included.

To begin with, the project is generally well written, and most development and security practices were adhered to. Correct logic, checks and protections are mostly used, and the functionality represents what could be a real-world JavaCard application. Overall, the quality of the project was high, and we enjoyed the challenge of finding flaws in the project.

Because of what is stated above we have focused on the following analysis methods when looking for vulnerabilities in the code: manual review (scope of variables, manual testing of functionality, manual review of checks, incrementations, etc.), semi-automated code quality review (reports by Amal can be found in our repository), security primitives review (checking if there are any know attacks against used algorithms) and automated static analysis.

For the automated analysis we have used a tool called [SonarQube](https://www.sonarqube.org/) which was originally an OWASP project that is now standalone. This tool was shared in PV260 Software Quality and we found it to be versatile with a surprisingly good identification of both bugs and security weak spots. The tool has also confirmed two of the issues we have discovered previously during manual review and it has correctly identified security “hotspots” for additional review. Overall, we found it useful and worth possibly including in the course as a tip for when a code review is needed. Sadly, report from this tool can only be exported in the paid variant but screenshots are included for the two identified issues later

## Mitigated attacks

Let us briefly go over the layers of protection mentioned to show what we have considered and what is already protected against.

PIN value is properly protected from retrieval. It is only ever kept locally for ECDH session initiation and is never exchanged in cleartext or any other decipherable form. In theory, it could be extracted from memory, but only during a session initiation phase. We see this as general flaw of any such encryption software. As for flow control it is properly implemented and even though some places do not check all of the received content (for example only return code is checked, not the apdu content), we have tried exploiting this in a number of ways and additional checks implemented elsewhere (such as exception handling of crypto functions) have caused detection and pin counter decrement every time. Regardless we would recommend decreasing the PIN counter before an operation rather than on detected fault.

Key handling is done properly as well. Session keys have a limited lifetime during which they can be extracted from memory, which does not compromise previous sessions in any way. Two things worth noting here are the storage of “secrets” on the card and the oracle padding attack that could be used to retrieve the shared session key.

The application includes functionality to put secrets on the card and retrieve them from the card. The way this is implemented is that the encrypted secret is decrypted and stored on the card “in plaintext”. To retrieve it, the user has to send encrypted retrieval command and is then served the encrypted secret. The issue we see here is that if session keys are compromised (short term PC access) they can be used to retrieve even those “old” files as the card only checks the current session key and does not require the old one. We understand this is an architectural decision and would make key management very complicated, but we still wanted to mention it.

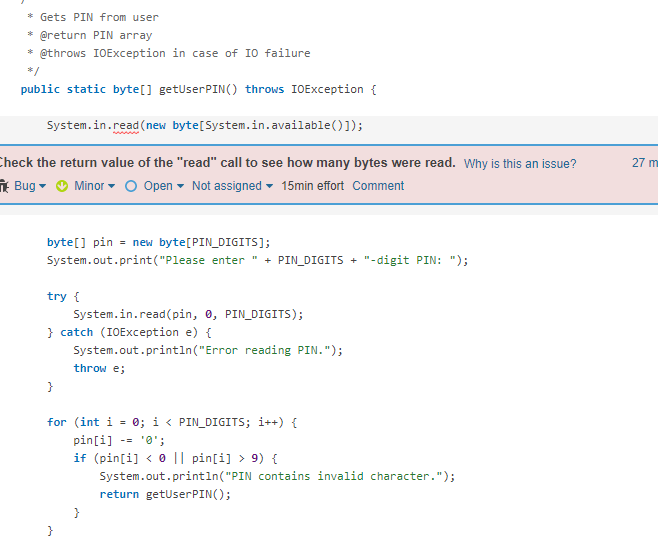
For the padding oracle attack, which was mentioned also on the presentation call, the attack, while technically possible, is extremely unlikely. Technically, if the attacker had access to the card for extended periods, side channel analysis could reveal when padding exception is being thrown by the cypher, however, even under these unlikely circumstances. The session counter means the attacker would only have 20 attempts overall, making the attack as likely as guessing a PIN within the 3 tries.

Interestingly padding oracle attack was also identified by automated tools:  


Authentication of both sides is properly handled. Multiple instances of the application do not pose any additional vector of attack from our findings. Older sessions (already expired) can not be replayed. Within a current session data can be replayed or manipulated, as discussed in the following chapter.

## Found issues

Despite the overall high quality, we were able to identify the following 2 issues we view as having impact on potential use.

**Problem identification: incorrect handling of input data by PC side**  
Severity: high - critical   
Risk: high – high chance of affecting legitimate users and causing disruption of service   
Problem description: The application incorrectly reads the PIN value entered by user as a byte array and does not check the amount of data read. As a result the data can be incorrectly parsed and further processed, resulting in either a) need for repeated authentication or b) multiple PIN attempts deducted as a part of one attempt, possibly blocking the card unintentionally.  
Remediation: either check the amount of read data or read string and parse it to an integer.  
SonarQube output:  


**Problem identification: no data integrity and message replay checking**  
Severity: critical  
Risk: high/critical   
Problem description: The application does not implement message integrity and replay checking and instead relies on encryption/decryption failing to detect message manipulation. While the fact that sessions are short and decryption is checked mitigate the issue to some extent, it means that, for example, messages can be held from the card. We have successfully tired this with the functionality for storing data, where we can hold back the encrypted secret and simply report back that we have received the data properly as card (either by replaying the message a card sent previously or sending anything as only return code was checked here.   
Additionally as no message counters or unique data is included in every message, an attacker can observe traffic for identical messages, gaining additional knowledge about the communication.  
Remediation: add message counter and hash to plaintext message with synchronization of counters and checking of hash on both ends. Example of new plaintext format: hash || counter || text.  
Add these protections to plaintext so they cannot be easily modified.

Issues for both findings were opened on GitHub and acknowledged there by Imrich.