Common Security Evaluation &
Certification Consortium
of GBIC and UKF

Common
Security Evaluation & Certification
Consortium

Common.SECC

Rule Book

Annex 4

Source Code Analysis Requirements

Version 0.91 (for trial use)

30 July 2020

Document History

Version	***Date***	***Change***
0.1	17.08.2016	First creation
0.2	16.09.2016	Consideration of the result of the JTEMS meeting in September 2016
0.3	16.09.2016	Remarks of BSI
0.9	18.04.2017	Remarks of JTEMS and Common.SECC
0.9	01.01.2018	New Logo and Disclaimer aligned
0.91	30.07.2020	Removal of unnecessary duplication with other annexes

Table of Contents

1 Introduction

Common Criteria evaluations are mainly based on the assurance requirements design analysis (ADV), testing (ATE) and vulnerability analysis (AVA). The evaluator checks and examines all related work units. These assurance requirements are assumed to cover a lot of attack paths to be considered with regard to payment transaction assets.

In order to prove resistance against software attacks as part of AVA_POI the evaluator is expected usually to perform penetration attacks via the external interface.

However, a set of software attacks can be excluded only or in a more efficient way by a code analysis. According to AVA_POI the evaluator is free to decide whether the evaluator performs a code analysis or not.

This document has been created to provide minimum requirements and recommendations for a code analysis in a CC evaluation according to JTEMS PPs.

For this purpose the document formulates work units for the evaluators.

The verb *shall* which is presented in italic letters is used when the provided source code analysis requirement is mandatory. The verb *should* which is presented in italic letters is used when the provided source code analysis requirement is required unless not applicable or not being performed for any other reason. If the requirement is not applicable or not performed for any other reason the evaluator shall provide a rationale.

It is understood that evaluators will use their discretion in applying relevant tests to cover emerging threats.

Code analysis shall be performed by reading code lines in single steps. Each table reflects an attack path and related to the attack path the code analysis step is given to prove by source code analysis that the related attack is countered by the implementation.

The evaluator shall document the performed code analysis.

2 Requirements

2.1 Evaluator checks and verifications applicable for each product

No	General
1.	The evaluators ***shall check*** whether the implementation is well-structured and has sufficient comments in order that the evaluators can understand the implementation.
2.	The evaluators ***shall check*** compiler settings in order that test or debug settings are deactivated.
3.	The evaluators ***shall check*** whether third party software is used according to related security guidelines if such security guidelines are available.
4.	The evaluators ***shall verify*** the requirements of the underlying security hardware (setting and reading of security related flags, correct parameters of sensitive hardware calls, ...) according to the hardware security guideline. This holds especially for switches to turn on counter measures against side channel analysis or glitches.
5.	The evaluators ***shall verify*** the self-test mechanisms of the payment terminal by code analysis.
6.	The evaluators ***shall verify*** mechanisms to perform updates of the firmware by code analysis.

No	The attacker discloses confidential data because of a **data buffer misuse**.
7.	The evaluators ***shall check*** that dependent on the security architecture of the implementation any confidential data is not stored or used (as far as possible) in cleartext.
8.	The evaluators ***shall check*** that erasure of confidential data is done in a way that the compiler does not deactivate the erasure procedure. The corresponding risk is that the compiler may deactivate the erasure procedure because the deleted buffer is not used anymore afterwards. This holds for PINs (after verification) and private keys as well as symmetric keys (after usage).
9.	The evaluators ***shall check*** for external interfaces input/output buffer management (byte counters, boundary checks, ...). Alternatively penetration tests can be applied.

2.2 Evaluator checks and verifications typical for payment terminals

No	The attacker discloses confidential data or manipulates the security related process flow by a **timing attack**.
10.	The evaluators ***shall verify*** by code analysis that any verification of PINs and password is implemented timing independent. Thus e.g. the full data shall pass the verification procedure.
11.	The evaluators ***shall verify*** by code analysis that any verification of authentication codes (like MAC, …) is implemented timing independent. Thus e.g. the full data shall pass the verification procedure. Compiler optimization shall not lead to timing dependence (compiler flags to be checked).

No	The attacker discloses confidential data because of a **data buffer misuse**.
12.	The evaluators ***shall check*** by code analysis that any confidential data is explicitly erased after usage (e.g. by overwriting the buffer). This holds for PINs (after verification) and as well as private and symmetric keys (after usage).
13.	The evaluators ***shall check*** that well defined memory areas are used to store PINs, private keys and symmetric keys. Whenever possible pointers are to be used instead of copy them from one memory area to another.
14.	The evaluators ***shall check*** PIN block padding as well as PIN formatting by code analysis if random data is used for padding.

No	The attacker discloses confidential data, manipulates sensitive data or manipulates the security related process flow **sending fuzzy data to the terminal**.
15.	The evaluators ***shall identify*** at least functions processing cryptographic data, PINs, security verification results, security related counters, security related flags and PINs in the design specifications and ***shall check*** the implementation of these functions.
16.	The evaluators ***shall check*** that only specified commands are allowed to be implemented. This holds especially for smartcard APDU commands and online authorisation commands (if they are controlled by the platform).

No	The attacker discloses confidential data, manipulates sensitive data or manipulates the security related process flow by **analysis of the random number generator (RNG)**
17.	The evaluators ***shall check*** by code analysis whether the noise source (e.g. generated by a security processor compliant to the latest version of ISO/IEC 18031 and NIST SP 800-90) is processed in order that the evaluators are convinced that unpredictable random numbers are generated.
18.	The evaluators ***shall check*** that security functions use randoms only provided by RNGs examined by the code analysis.

3 Recommendations

3.1 Evaluator checks and verifications applicable for each product

No	General
19.	Compiler settings ***should be taken into consideration*** in order to see whether code optimization may lead to vulnerabilities in the executable code or the setting of warnings during compilation.
20.	The evaluators ***should check*** whether the used compiler creates vulnerabilities and whether e.g. a newer version not creating vulnerabilities exist.
21.	The evaluators ***should check*** that third party software is free of known vulnerabilities.
22.	Make files or similar configuration files ***should be taken into consideration*** in order to check whether only the intended source code files, libraries etc. are linked together to the final product.
23.	The evaluators ***should consider*** static code analysis tools or the vendors should use them and the evaluators ***should check*** the results of these tools.
24.	The evaluators ***should check*** the recommendations given in the CEM V3.1 R4, sections 1202 – 1204.

3.2 Evaluator checks and verifications typical for payment terminals

No	The attacker discloses confidential data, manipulates sensitive data or manipulates the security related process flow because of **internal illegal code instructions**.
25.	The evaluators ***should check*** security related case switches in order to verify that only the specified functions are called with the specified parameter. In particular a default block should be always present. This holds especially for the command processing and its parameter dependent called functions.
26.	The evaluators ***should consider*** drawing "trees" of main security functions with the called functions as branches in order to see which subroutines are called to implement the main security function. This holds especially for functions working on cryptographic keys and PINs.
27.	The evaluators ***should check*** whether the software is free of dead code or whether there is no branch to execute dead code.
28.	The evaluators ***should check*** whether return values reflect the behavior of the security related function.

No	The attacker discloses confidential data, manipulates sensitive data or manipulates the security related process flow **sending fuzzy data to the terminal**.
29.	The evaluators ***should check*** that processing of input data is well-structured.
30.	The evaluators ***should check*** based on functional specification that undocumented commands are not used.

No	The attacker discloses confidential data, manipulates sensitive data or manipulates the security related process flow **performing a power interrupt**.
31.	The evaluators ***should check*** that sensitive data to be stored in non-volatile memory areas is written in a way being independent from a power interrupt.
32.	The evaluators ***should check*** that error counters are decreased before operation and increased afterwards if no error appears.
33.	The evaluators ***should check*** that usage counters are increased before the operation, not afterwards.