



How helpful can be the security assessment of TOE associated tools during a Common Criteria evaluation.

11th ICC, Antalya, 21-23 September 2010

- ▶ Introduction
- ▶ TOE generation: Java Card applet example
 - ▶ CAP file generation
 - ▶ Compiler and Converter
 - ▶ Security Issues
- ▶ TOE environment: Byte Code Verification example
 - ▶ Byte Code Verifier Functionalities
 - ▶ Security Issues
 - ▶ Off-Card Verifiers analysis
- ▶ Conclusion



Thales ITSEF:

- ➔ HW & embedded SW ITSEF
- ➔ Under ANSSI agreement

- ▶ Author: **Emilie Faugeron**
 - Common Criteria Evaluator since 2009
 - Java Card expert



Thales ITSEF:

- ➔ HW & embedded SW ITSEF
- ➔ Under ANSSI agreement

▶ **Speaker: Jean-Yves BERNARD**

- Evaluator for 9 years
- Thales ITSEF technical manager
- Lead auditor ISO/IEC 27001:2005 (certified by LSTI)
- Risk manager ISO/IEC 27005:2008 (certified by LSTI)

- ▶ Introduction
- ▶ TOE generation: Java Card applet example
 - ▶ CAP file generation
 - ▶ Compiler and Converter
 - ▶ Security Issues
- ▶ TOE environment: Byte Code Verification example
 - ▶ Byte Code Verifier Functionalities
 - ▶ Security Issues
 - ▶ Off-Card Verifiers analysis
- ▶ Conclusion

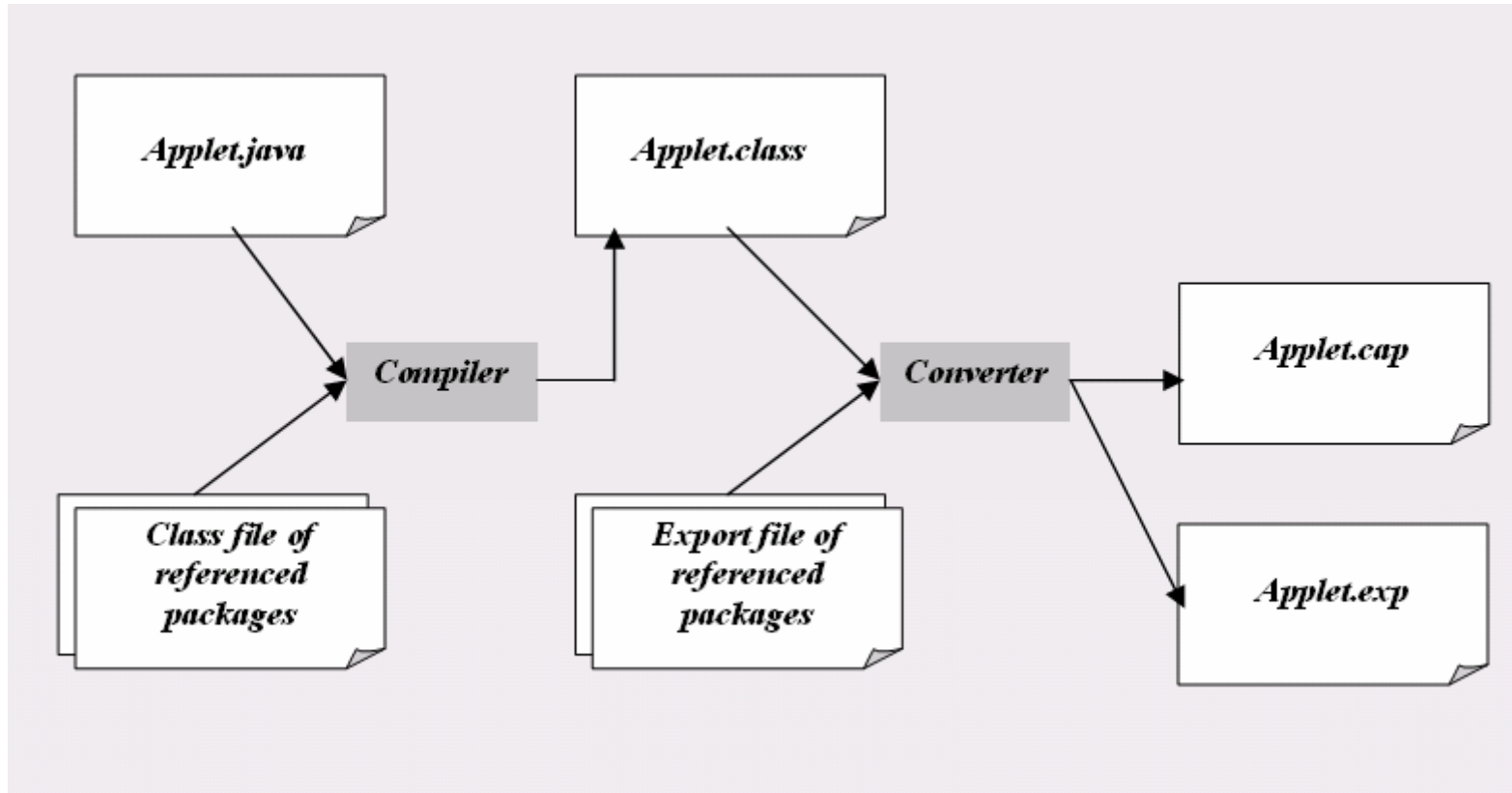
- ▶ In this presentation, we use “javacard” example, this could be also applicable to other languages.
- ▶ Several tools are used to generate the source code associated to the TOE
 - ▶ Compiler
 - ▶ Converter
 - ▶ ...
- ▶ Other tools are included in the TOE environment
 - ▶ Off-Card Verifier
 - ▶ On-Card Verifier
 - ▶ ...
- ▶ Such tools can have a security impact on the TOE.
- ▶ However those tools are rarely deeply analyzed in the evaluation context

- ▶ Introduction

- ▶ TOE generation: Java Card applet example
 - ▶ CAP file generation
 - ▶ Compiler and Converter
 - ▶ Security Issues

- ▶ TOE environment: Byte Code Verification example
 - ▶ Off-Card Verifier Functionalities
 - ▶ On-Card Verifier Functionalities
 - ▶ Security Issues
 - ▶ Off-Card Verifiers analysis

- ▶ Conclusion





- ▶ **Compiler**
 - ▶ Creates one “.class” per class
 - ▶ Could apply some optimizations in order to reduce time execution and/or CLASS file size

- ▶ **Converter**
 - ▶ Creates one “.cap” per package
 - ▶ Due to Java Card constraints, some modifications and/or optimizations could be applied to Java source file

- ▶ Optimizations and/or modifications applied by the compiler or by the converter could avoid some developer countermeasures

Security Issues: During CC evaluation



- ▶ The work unit ALC_TAT.x-3
 - Requires that tool documentation unambiguously defines the meaning of all implementation-dependant option.
 - Compiler/converter options must be described in the associated documentation. This must allow both the developer and the evaluator to determine impact of each option on source code

Evaluator work:

The evaluator should check that the options selected are consistent with the source code security level

Security Issues: During CC evaluation



- ▶ During ADV_IMP and AVA_VAN activities the evaluator performs source code audit, makefile audit.

Evaluator work:

The evaluator should use the compiled assembly source code to ensure that the counter measure will be actually present. For that, the evaluator should adopt a sampling strategy.

NB: implicit options can be implemented by such tools

- ➔ In some cases, a real test is the only answer to understand if a counter measure was “optimized” or not !!!

- ▶ Introduction

- ▶ TOE generation: Java Card applet example
 - ▶ CAP file generation
 - ▶ Compiler and Converter
 - ▶ Security Issues

- ▶ TOE environment: Byte Code Verification example
 - ▶ Byte Code Verifier Functionalities
 - ▶ Security Issues
 - ▶ Off-Card Verifiers analysis

- ▶ Conclusion



- ▶ A Byte Code Verifier includes the following checking *
 - ▶ Correctness of the CAP file format
 - ▶ Bytecode instructions represent a legal set of instructions used on the Java Card platform
 - ▶ Adequacy of bytecode operands to bytecode semantics
 - ▶ Absence of operand stack overflow/underflow
 - ▶ Control flow confinement to the current method
 - ▶ Absence of illegal data conversion and reference forging
 - ▶ Enforcement of the private/public access modifiers for class and class members
 - ▶ Validity of any kind of reference used in the bytecodes
 - ▶ Enforcement of rules for binary compatibility

Byte Code Verifier Functionalities



- ▶ A Byte Code Verification can be performed
 - ▶ Before application loading using an Off-Card Verifier
 - ▶ Before application installation using an On-Card Verifier
 - Analyze the structure of the CAP File
 - Analyze de byte code and components statically
 - ▶ Before application execution using an On-Card Verifier
 - Analyze the byte code dynamically

▶ On-Card Verifier

- ▶ It must deal with Java Card limitation (restrictive memory and execution time)
- ▶ But it can be included in the scope of a Platform evaluation, covered by:

OSP → Objective the TOE → SFRs

▶ Off-Card Verifier

- ▶ No on-card constraint: can implement, a complete verification
- ▶ CAP file could be modify between the verification and the installation process
- ➔ A signature could be added in the CAP file at the end of the verification to resolve this issue

This is outside the scope of the evaluation, covered by:

OSP → Objective on the operational environment → Guidance

- ▶ Javacard Platform, Common Criteria evaluation context:
 - ▶ It is not required that the Off-Card Verifier used is identified in development tools or in guidance.
 - ▶ If Off-Card Verifiers tools are not evaluated or qualified
 - Conformity to the specification could be not complete
 - Bugs that represent potential weaknesses could be found
- An attacker could take benefit of this.

- ▶ Javacard Platform, Common Criteria evaluation context:
 - ▶ The Off-Card / and On-Card options are both good for security.
 - ▶ But, in case of usage of an off card verifier, The evaluator should check:
 - 1 → That the “off-card” that must be used are clearly identified in the security guidance of the TOE (AGD_OPE activity for applet issuers/loaders).
 - 2 → That the identified “off-card” tools are conformant to the specification (even if it is covered by an OE in the ST).
 - 3 → That the identified “off-card” tools does not contain bugs that could be easy to exploit in order to bypass verification (even if it is covered by an OE in the ST).

Points 2 & 3 could be evaluated/qualified independently in a dedicated “off-card” evaluation.



- ▶ In order to “qualify” the “off-card” verifier, the evaluator can (in the frame of AVA_VAN activity):
 - ▶ developed applets which include different kinds of errors in order to retrieve:
 - Verification performed between different CAP file components
 - Verification performed inside a CAP file component
 - Verifications applied to byte code
 - ...
 - ▶ Perform a complementary source code analysis if necessary (the evaluator must then have an access to the tool source code).

- ▶ Introduction

- ▶ TOE generation: Java Card applet example
 - ▶ CAP file generation
 - ▶ Compiler and Converter
 - ▶ Security Issues

- ▶ TOE environment: Byte Code Verification example
 - ▶ Byte Code Verifier Functionalities
 - ▶ Security Issues
 - ▶ Off-Card Verifiers analysis

- ▶ Conclusion

- ▶ The compiler and the converter could have an impact on byte code security
- ➔ Compiler option must be considered by the evaluator and eventually generated assembly in addition to implementation representation.
- ▶ Off-Card Verifier tools could include some vulnerabilities:
 - ▶ Therefore, analysis of the tool could be included within AVA_VAN activity (even if it is covered by an OE).
- ➔ Assess the security impact of such tools, either for TOE generation or TOE environment is necessary. It could enrich the vulnerability analysis, and will increase consistency between all mechanisms involved in product security.

