Evidence Based Evaluations – Chances and Challenges

Helmut Kurth
atsec information security corp.

Content

- What is “Evidence based Evaluation”?
- How differ most evaluations from this approach?
- Reality versus CC expectations
- What “evidence” can be expected?
- How to manage “tons” of evidence
- What if the “evidence” is not “CC compliant”?
- Benefits of “Evidence based Evaluations”

What is “Evidence Based Evaluation”?

- First a remark: Every evaluation is based on some kind of “evidence”!
- In many cases (most of) the “evidence” is created specifically for the evaluation:
  - “Design” documentation
  - “Developer” test plans and test cases
  - Guidance documents
  - Development process documents
  - ….
- **As a result the developer has a significantly increased effort!**

Why this “Evidence Creation”?

- The CC have very specific expectations on the content of specific documentation!
  - Those expectations are not always realistic and not always necessary to perform an evaluation!
- Some schemes and labs expect documents that almost exactly match the CC expectations
  - This is a highly unrealistic expectation!
- Some third parties see a good business opportunity in selling the creation of “CC compliant documentation”
  - This is counterproductive!
- **Some vendors don’t have useful design documents**
  - They should accept that they have a problem!

Reality versus CC

Functional Specification

- TSFI versus API in the case of an operating system

  Developer documentation describes the API (usually a library or a macro interface) and not the TSFI

  Hopefully the API is “close enough” to the TSFI (which needs to be validated as part of the evaluation)

  Problem: nothing enforces that the developer provided library or macros are used by a programmer!

  - TSFI may have additional parameter not presented by the library/macro
  - TSFI parameter list structure is hidden by the library/macro
  - Library functions/macros may perform security relevant checks
  - Exact TSFI function called is hidden by the library/macro

Reality versus CC – One example

Functional Specification

ADV_FSP.4.2C

The functional specification shall describe the purpose and method of use for all TSFI.

What is the “method of use” for a system call in an operating system?

- The library/macro? Their use can not be enforced!
- The system call instruction used? There may be several different ones even with a single processor!
- A single library function or macro may “branch” to different system calls based on the parameters passed to it!

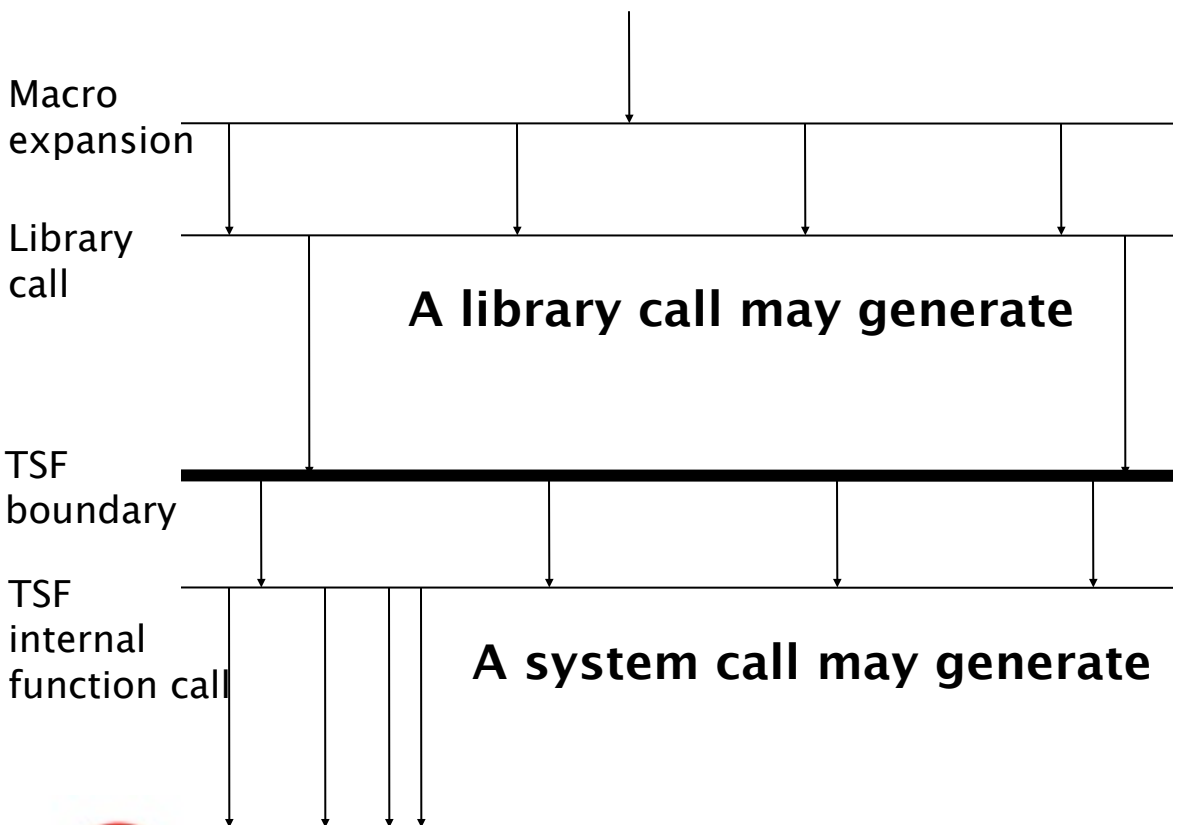
How can such a situation been handled in an evaluation?

What is the “method of use”?

Documentation Example

- Developer programming interface description is dedicated to program developers
  - Describes what the program developer is supposed to use
  - Does not describe what the programmer is not supposed to use
- Example:
  - OS functions are often described via the macro interface
  - This is the “intended method of use” – but it is not enforced by the TSF!

Example: System Call Interface



different library calls (depending on macro parameter)

different system calls (depending on function parameter)

different calls to TSF internal functions (depending on system call parameter)

Reality versus CC

Functional Specification

Existing Evidence:

- Description of the library/macro interface (part of the documentation for developers)
- Description of the system call entry points in the operating system “kernel” (often as comments in the code)

Task of the evaluator:

- Compare both descriptions to see if they “match”

Existing Evidence – What to Expect

- **CC view:** Developer provides “security focused” design documentation
  - Classification into “SFR-enforcing”, “SFR-supporting” and SFR-non-interfering”
    - No developer does this
  - Different level of detail for “SFR-enforcing” subsystems and modules
    - Not useful from a developer point of view

- **Reality:** Developer provides “product functionality focused” design documentation
  - Describes the whole functionality
  - Focuses on multiple aspects – security is just one of them

Existing Evidence – What to Expect

- **Usually the developer has:**
  - A very large number of design related documents
    - In one example: more than 500
  - Design, interface description and guidance not in separate documents
    - Written with software developers and users in mind, not evaluators
  - Different documentation style, document format, level of detail
    - In one evaluation we had
      - 3 different word processor formats
      - PDF
      - HTML
      - Pure ASCII text
      - and more …. (spreadsheets, XML, developer internal formats)
  - Security aspects spread over all documents
  - Mixture of public documents and internal documents

Existing Evidence – What to Expect

- **Quality of documentation**
  - Not all documents are completely up-to-date
  - Not all documents are easy to read
    - Especially internal documents are written for people with detailed knowledge of the product
  - Not all details are described to the level the CC expects
    - In internal documents for developers, it is expected that the reader obtains the details from the source code (and the comments there)

Existing Evidence – Summary

- Existing evidence is often very large
  - Thousands of documents with a total of several hundred thousand pages!
  - Existing evidence comes in a large variety of formats, different documentation style, different level of detail
  - Existing evidence not structured into “design”, “guidance”, “process”, and “testing” but often a mixture of several of those
  - Finding the existing evidence is often not easy
    - There is usually not a single person that knows where to find all the existing evidence

- **How can one handle a CC evaluation based on such evidence?**

Evidence based Evaluation – Work Units

- **Evidence collection**
  - Finding the evidence (together with the developer)
- **Evidence classification**
  - Which CC aspects are addressed, which functions, subsystems, modules, test plans etc are addressed
- **Format conversion**
  - For easier comparison and navigation
- **Evidence management**
  - Maintaining an “evidence data base”
- **How to “navigate” the evidence**
  - Finding the information that matters and put it into context

Evidence Management and Navigation

- **This is the art of evidence based evaluation!**
- Requires dedicated tools
  - Some can be of general use, some are specific for the evaluation of one product
- Must support tracing
  - From the ST to the FSP down to the design down to the code
  - From FSP and the design to testing
  - From the ST to guidance
- Must allow “cross-referencing” and intelligent searching
  - Example task: Provide all evidence documents related to a specific term (e. g. an interface or a data structure)

An example

- z/OS
  - About 30 SFRs in the ST
  - Broken down to about 600 “detailed functional claims” in the TOE summary specification
  - Each of the detailed functional claims is mapped to
    - The TSFIs were the effect can be observed and tested
    - The TSFI were the functionality can be configured/managed (when appropriate)
    - The test cases for the detailed functional claim
  - Each TSFI is mapped to the document that defines the interface
  - Each TSFI is mapped to the subsystem, module and the entry point name (source code module name) in the TSF (as far as possible)

An example

- z/OS
  - Collected about 1000 documents
    - Mainly design related
    - More than 150 process related
  - Total size: more than 200,000 pages
    - Excluding source code of course
    - Excluding design/guidance documents for functions outside of the TSF
    - Excluding test cases and detailed test procedures
  - Several different document formats
    - Word, WordPro, ASCII text, PDF, HTML, Bookmaster, …
    - Very different documentation styles

Chances

- Evaluator needs to get a detailed view and understanding of the whole product (or at least the whole TSF) to perform the evaluation
  - Not just the security view
  - Requires an evaluator with very high skills
  - Provides the chance to get a significantly better knowledge of the product than with dedicated “CC documents”
  - Evaluation is more related to the practical use of the product
- The evaluator gets a significantly better basis for his vulnerability analysis
  - Most vulnerabilities are in non-security related functions!

Challenges

- Finding and managing the evidence
- “Transforming” the evidence into a “navigatable form”
- Finding the answers to specific questions within a huge set of evidence documents
- Using the evidence for efficient tracing from the ST down to the source code and the test cases
- Identification of gaps in the documentation and filling the gaps
- Work with partly incomplete and partly outdated documentation
- Evidence is at least a factor of 10 larger than dedicated “CC documents”

Challenges

- Evidence sometimes a factor of 100 larger than dedicated “CC documents”!
  - Example z/OS:
    - Number of evidence documents (without source code, “module prologs” and test cases): ~ **1000**
    - Number of pages in all those documents: > **200,000**
  - One specific example aspect:
    - “Error messages”: spread over 25 documents that solely contain error and warning messages and an additional 10 chapters in other documents (not including return codes of functions!)
    - Sum of pages just of the 25 dedicated error/warning messages books:

14334

Further Challenges

- Dealing with highly unrealistic CC/CEM requirements
  - And there are many of those – I could give half a day of presentation on those!
  - See the examples in the slides not presented
- Dealing with imprecise CC/CEM requirements
  - Talking about those would fill the other half of that day!
- Preparing the evaluation reports
  - They often start with a significant number of pages just explaining the structure of the evidence used
  - They get very lengthy just by explaining how the product is implemented/tested/developed and how to navigate through the evidence

CC and Evidence based Evaluations

- Individual “simple” CC requirements often satisfied only by a large number of evidence documents
  - See the example of the “error messages”
- Quite a number of CC expectations are naïve and sometimes even counterproductive
  - Developer can satisfy aspect easier with less secure products
    - If IBM significantly reduced the detailed error message into some ore general (and less useful) ones, evaluation of that aspect would be significantly simplified – but result in a less usable product!
- **Requires quite some flexibility by the scheme**
  - Need to understand that the essence of the CC are satisfied

Summary

- Evaluations based on existing evidence is not that simple
  - Developer evidence usually not aligned with “CC expectations”
  - Developer evidence can be very large
- Finding the relevant parts is the challenge
  - Requires sophisticated “evidence management”
- “Evidence based” evaluations usually give a better view of the whole product
  - Not just the “security functionality”

CC need significant rework for better support of evidence based evaluations

Some specific CC Problem Examples

- **Functional Specification**
  - The functional specification shall describe the purpose an method of use for all TSFI.
    - System calls are usually described with their library or macro interface – this is not the “real” TSFI!
  - The functional specification shall describe all actions associated with each TSFI.
    - This is a requirements that can never be fulfilled!

Some specific CC Problem Examples

- **Functional Specification – Error Messages**
  - The functional specification shall describe all direct error messages that may result from an invocation of each TSFI.
    - In z/OS one has:
      - Return codes, most of which are described with the function description, some are described separately and apply for a set of functions
      - ABEND codes and error messages
        - » Description for the core system in 6 separate documents, total xxxx pages
        - » Description for specific subsystems in subsystem specific documents
        - » Identifier in the message leads to the document describing the message (for a well-trained system programmer)

Some specific CC Problem Examples

- **Functional Specification – Error Messages**
  - Are those descriptions complete?
    - No, there are error messages for IBM field service!
  - Can all of them be related to the invocation of a TSFI
    - No, only a subset can (which is normal)!
  - Does the IBM documentation distinguish between “direct error messages that may result from an invocation of each TSFI” and “other” error messages?
    - Of course not!
- Formally the CC requirement is hard to verify for z/OS
  - In practice IBM’s approach is highly user friendly and better than many other products

Some specific CC Problem Examples

- **Functional Specification – Actions**
  - The functional specification shall describe all actions associated with each TSFI.
    - Just think about audit records generated as the result of the TSFI invocation
    - In most cases they are **not** described with the interface
      - Because they are not relevant for the caller
      - Because in some cases the caller should not be aware of the audit records generated
      - Because audit records will only be generated with specific configurations of the audit system
    - In most cases they are described in specific documents for the audit component
      - That is the place where the naturally belong to!

Some specific CC Problem Examples

- **TOE Design**
  - CEM, ADV_TDS.3-9:
    - "They could then check to see if the processing appears to "touch" any of the global data areas identified in the TOE design. The evaluator then determines that, for each global data area that appears to be "touched", that global data area is listed as a means of input or output by the module the evaluator is examining."
  - OS modules usually "touch" hundreds of "global data areas"
  - They are usually not "listed" if they are used for input only
    - Why should they?
  - **CC ignores the more important security issues:**
    - How are the global data areas protected?
    - How is access to them synchronized?

Some specific CC Problem Examples

- **TOE Design**

  ADV_TDS.3.5C The design shall provide a description of the interactions among all subsystems of the TSF.

  - Design information is usually provided describing how the TSF provides its services
  - Per service, the interaction between the subsystems are described – unless they are obvious, like:
    - When an OS subsystem uses the file system subsystem to store data, no further description on this interaction is usually provided
    - Most OS subsystems use the memory management subsystem, but this is usually not further described in the design
      - There is no reason to describe obvious interactions

Some specific CC Problem Examples

- **TOE Design**

  ADV_TDS.3.8C The design shall describe each SFR-enforcing module in terms of its SFR-related interfaces, return values from those interfaces, interaction with and called interfaces to other modules.

  - Design documentation is usually not SFR-centric
  - Module design documentation usually describes in general terms how functions from other modules are called, not in terms of interfaces
  - It is more important for an evaluator to know what functionality is called for which purpose. Details which interface is called and the parameter of the call can often be obtained from the source code.

Some specific CC Problem Examples

- **TOE Design**

  ADV_TDS.3.10C The mapping shall demonstrate that all behaviour described in the TOE design is mapped to the TSFIs that invoke it.

  - There is a lot of behaviour described in the TOE design that can not be mapped to TSFI!
    - Error/exception handling
    - Handling of hardware failures
    - Synchronizing access to resources
    - Internal optimization
    - …..
  - **Thanks God, the CEM completely reinterprets this component!**