# Applying the Draft CC Version 3.0 to Linux
# - Experience from a Trial Evaluation -

Helmut Kurth

atsec information security

# Outline

- The task – try Draft CC Version 3.0
- The target – SLES9
- The Security Target
- The work items – Classes ADV and AVA
- The experience
- What has changed in draft CC Version 3.1
- Conclusion

# The Task

- Trial evaluation of a real product – not just a toy or just a Protection Profile
- Focus on comparison with CC Version 2.3 experience and re-use of evidence and evaluation results (no change in functionality)
- Focus on classes ADV and AVA (as two major aspects that have changed)
- Provide suggestions for improvement

# The Target

- Novell/SUSE Linux Enterprise Server 9
  - Fairly complex TOE previously evaluated at EAL4 augmented
  - Large set of complex security functions
  - Full set of evidence available
    - Parts of it are freely available
  - Evaluation done recently
    - Still very familiar with the TOE and the evidence

Was considered to be an ideal test candidate

# The Security Target

- Attempt to re-write ST using part 2 of draft CC Version 3.0
  - Attempt basically failed
    - No easy mapping of SFRs
    - Many SFRs from the existing ST could not be expressed without definition of extended SFRs
    - Some SFRs could be rewritten easily and sometimes the readability of the ST was enhanced
    - Overall part 2 seemed to be inadequate to address the SFRs of an operating system

# The Security Target

- Decision was:
  - Modify ST using part 2 of CC version 3.0 where an easy mapping was possible
  - Leave all other SFRs (define the SFRs used from CC version 2.3 as extended SFRs)
  - Don't adapt the rest of the ST to the structure and requirements of draft CC version 3.0
    (no implication on the planned evaluation work)

# The Work Items

- Focusing on ADV and AVA
  - Two classes where major changes had been made
  - Part of the “core” of CC evaluations
- Handle as a “re-evaluation”
  - Common scenario to be expected
  - Re-use of evidence and evaluation results needs to be possible

# Experience (ADV_ARC)

- ADV_ARC is a new family
  - Required when FDP_SEP and FDP_RVM is claimed (as is the case for most operating systems)
  - Focusing on TSF internals
  - Evidence expected to be in the high-level and low-level design documents for the v 2.3 evaluation

# Experience (ADV_ARC)

- Evidence
  - Most of it could be identified in the existing HLD and LLD documents (which includes the specifications of the hardware/firmware)
  - Not sure this is true for all evaluations (SLES9 evidence for v 2.3 was quite detailed on architecture details)
  - ADV_ARC forces to take a different view than v 2.3
    - One aspect was identified where the existing documentation was insufficient. Could be fixed easily.

# Experience (ADV_FSP)

- Many parts from previous reports could be re-used
- Change of work unit text and order made re-use unnecessarily complicated
- Requirements for “error messages” are unrealistic
  - Error message may pop up at an interface from events mainly unrelated to the function called
  - Requiring to list all possible error messages for an interface is therefore unrealistic
  - Many systems describe the error messages independent from the functional interface description – for good reasons!

# Experience (ADV_TDS)

- ADV_TDS combines the old ADV_HLD and ADV_LLD
  - Many v 3.0 requirements are similar to v 2.3 requirements
  - Structure of requirements and work units is different causing unnecessary complications in a re-evaluation
  - Now includes requirements to describe the “algorithms” used
    - This is good (CC v2.3 focused too much on interfaces)
    - The way this was done was horrible!
  - Now includes requirements for description of common data structures
    - This is good
    - Requirements too strong (requires identification of all modules that read specific global data. This is unrealistic!)

# Experience (ADV_TDS)

- Nonsense in requirements for algorithmic description
  - From CEM, ADV_TDS.4-14
    - The requirement is that the developer must provide a full algorithmic description, and so the evaluator is not obligated to accept anything less, and is indeed not obligated to justify accepting anything less; the requirement provides the necessary justification.

      The algorithmic description is complete if it describes (in an algorithmic fashion) all of the functionality performed by the module.

  - The result would be a horrible documentation effort just to please the evaluators.
  - There is no indication what this information is used for in other work units!

# Experience (ADV_TDS)

- Classification of modules in
  - SFR enforcing
  - SFR supporting
  - SFR non-interfering
- Nice idea (though not new) – but it doesn't work on the module level!
  - The same module may be used for different purposes
  - Vendor will usually not provide such a classification

# Experience (ADV_IMP)

- Requirements were low in CC Version 2.3
  - Got even lower in draft CC Version 3.0!
  - No real work item to analyze the implementation representation
  - ADV_IMP.1-2 is counterproductive!
    - As an evaluator I want the implementation representation in a form best usable for analysis!
    - For Linux we used the Linux Cross Reference
    - Other vendors have specific tools helpful for analysis
    - Of course the evaluator needs to verify that what he sees what is actually implemented

# Experience (AVA_VAN.3)

- Misuse analysis has been moved to ADG
  - This is good. There was too much overlap
- No strength of function any more
  - Can be addressed as part of the AVA_VAN work items
- No developer vulnerability required
  - Counterproductive, since it told you, which vulnerabilities the developer looked for and how he did it
  - Made it easy to identify the areas the developer had not thought of
- In total vulnerability analysis is weakened while the effort for the evaluator is increased!

# Corrections made in CC V3.1

- ADV_FSP
  - Section about error messages has been reworded. Problem has been addressed.
- ADV_TDS
  - Requirements on algorithmic description and global variables mainly removed
    - Fixing them would have been better
  - Back to the structure into “subsystems” and “modules”
    - Avoids unnecessary confusion
  - Much closer to V2.3 than to draft V3.0
- ADV_IMP
  - Problems not addressed

# Corrections (not) made in CC V3.1

- AVA_VAN
  - Problems not addressed
  - Still no vendor vulnerability analysis required
  - Contradicts the approach to honor a vendor's security processes
  - Other evaluation activities do not provide a sufficient basis for a thorough evaluator vulnerability analysis

**In total vulnerability analysis is weakened**

# Conclusion

- New and modified items in Draft CC Version 3.0 considered in the evaluation
  - Part 2: found to be mainly unusable
  - ADV_ARC: found to be useful
  - ADV_FSP:
    - except for the aspect of error messages usable (corrected)
    - Restructuring complicated re-use of evaluation results
  - ADV_TDS:
    - Requirements partly unrealistic (corrected)
  - ADV_IMP:
    - Requirements lowered (not corrected)
  - AVA_VAN:
    - Requirements lowered (not corrected)

# Conclusion

- Draft CC V3.1 corrected some problems introduced in draft CC V3.0
  - Part 2 was withdrawn
  - Some still remain in part 3 and the CEM
- Draft CC V3.1 should allow a smoother transition from CC V2.3 than draft CC V3.0

**This still needs to be confirmed in practice**