Taking White Hats to the Laundry: How to Strengthen Testing in Common Criteria

Apostol Vassilev,
Principal Consultant

September 23,2009.

Product Testing in Common Criteria

Product Testing in Common Criteria

- Functional and penetration testing are important tools for gaining assurance in the evaluated product

Product Testing in Common Criteria

- Functional and penetration testing are important tools for gaining assurance in the evaluated product

Product Testing in Common Criteria

- Functional and penetration testing are important tools for gaining assurance in the evaluated product

Product Testing in Common Criteria

- Functional and penetration testing are important tools for gaining assurance in the evaluated product

- **Problem**: the testing methodology defined in CC is underspecified
  - results are difficult to reproduce
  - affects the public's perception of the value of evaluations

Outline

- Introduction
  - Current situation with product testing in CC
  - Recent advancements in testing and their potential use in CC
- Proposal
  - Modular assurance packages based on interface-specific attacks
  - Benefits from using such packages
- Conclusions and future work

Product testing according to CEM

Product testing according to CEM

- The goal is to test the behavior of TOE
  - as described in ST and as specified in the evaluation evidence
  - the focus is on testing the security functionality, defined by the SFRs

Product testing according to CEM

- The goal is to test the behavior of TOE
  - as described in ST and as specified in the evaluation evidence
  - the focus is on testing the security functionality, defined by the SFRs
- Evaluators test TSF by
  - devising own test cases
  - re-running a subset of developer's test cases

Product testing according to CEM

- The goal is to test the behavior of TOE
  - as described in ST and as specified in the evaluation evidence
  - the focus is on testing the security functionality, defined by the SFRs
- Evaluators test TSF by
  - devising own test cases
  - re-running a subset of developer's test cases

Product testing according to CEM

- The goal is to test the behavior of TOE
  - as described in ST and as specified in the evaluation evidence
  - the focus is on testing the security functionality, defined by the SFRs
- Evaluators test TSF by
  - devising own test cases
  - re-running a subset of developer's test cases
- CEM suggests alternate approaches only when it is impractical to test directly specific functionality
  - such as source code analysis

Limitations of testing defined in CEM

Limitations of testing defined in CEM

- Traditionally, emphasis is given to “functional testing” of security features
  - deterministic positive and negative testing prevails in the software industry
  - accepted by CEM and prioritized by relevance to SFRs:
    - SFR-enforcing TSFIs are covered
    - SFR-supporting or SFR-non-interfering TSFIs are largely ignored

Limitations of testing defined in CEM

- Traditionally, emphasis is given to “functional testing” of security features
  - deterministic positive and negative testing prevails in the software industry
  - accepted by CEM and prioritized by relevance to SFRs:
    - SFR-enforcing TSFIs are covered
    - SFR-supporting or SFR-non-interfering TSFIs are largely ignored
- The deterministic functional testing is good for confirming the overall security architecture and design of the product.

Limitations of testing defined in CEM

Limitations of testing defined in CEM

- Recent advances in testing technology have shown that deterministic functional testing is not sufficient for gaining assurance in the security features of a product
  - hackers pioneered random fuzzing of interfaces intended to penetrate them
  - fuzz testing is becoming more and more accepted by major software vendors and incorporated in product development
  - introduces the concept of probabilistic assurance

Fuzz Testing

Fuzz Testing

- Fuzz testing has evolved as black box testing to uncover hidden vulnerabilities and implementation bugs

Fuzz Testing

- Fuzz testing has evolved as black box testing to uncover hidden vulnerabilities and implementation bugs
- Fuzz testing of a given interface (API, protocol, etc) can be
  - Brute-force
    - invoke the interface with a completely random input data
  - Adaptive
    - use semi-random/semi-malformed input data

Fuzz Testing

- Fuzz testing has evolved as black box testing to uncover hidden vulnerabilities and implementation bugs
- Fuzz testing of a given interface (API, protocol, etc) can be
  - Brute-force
    - invoke the interface with a completely random input data
  - Adaptive
    - use semi-random/semi-malformed input data
- Open questions:
  - What is the proper cost/benefit ratio for this type of testing?
  - Can we map Fuzz testing results to EAL levels?

Fuzz testing

Fuzz testing

- Fuzz testing has been used successfully to uncover implementation bugs responsible for
  - system crashes
  - memory leaks
  - unhandled exceptions
  - buffer overflows
  - dangling threads
  - dangling pointers

- Most of these are code quality indicators, but they have direct security implications

Fuzz testing

- Fuzz testing has been used successfully to uncover implementation bugs responsible for
  - system crashes
  - memory leaks
  - unhandled exceptions
  - buffer overflows
  - dangling threads
  - dangling pointers

- Most of these are code quality indicators, but they have direct security implications

Fuzz testing

- Fuzz testing has been used successfully to uncover implementation bugs responsible for
  - system crashes
  - memory leaks
  - unhandled exceptions
  - buffer overflows
  - dangling threads
  - dangling pointers

- Most of these are code quality indicators, but they have direct security implications

Fuzz Testing

Fuzz Testing

- Open questions:
  - What is the proper cost/benefit ratio for this type of testing?
    - Hackers, developers have different perspectives
    - Where do evaluators stand?
  - Can we incorporate this type of testing in CC?
    - Can we map Fuzz testing results to EAL levels?

Limitations of testing defined in CEM

Limitations of testing defined in CEM

- Observation:
  - TSFIs cannot be reliably prioritized for CC testing as
    - SFR-enforcing
    - SFR-supporting
    - SFR-non-interfering
  - This issue is particularly relevant for low (<4) EAL evaluations

Limitations of testing defined in CEM

- Observation:
  - TSFIs cannot be reliably prioritized for CC testing as
    - SFR-enforcing
    - SFR-supporting
    - SFR-non-interfering
  - This issue is particularly relevant for low (<4) EAL evaluations

- Observation:
  - Any TOE interface exposed to attackers may be security relevant
  - Hence, it should be tested thoroughly

Limitations of testing defined in CEM

- Observation:
  - TSFIs cannot be reliably prioritized for CC testing as
    - SFR-enforcing
    - SFR-supporting
    - SFR-non-interfering
  - This issue is particularly relevant for low (<4) EAL evaluations
- Observation:
  - Any TOE interface exposed to attackers may be security relevant
  - Hence, it should be tested thoroughly
- Observation:
  - Fuzzing and interface-specific tests provide a good framework for this

Interface-specific testing

Interface-specific testing

- Why Interface-specific testing?
  - Interface-specific classes of attacks have emerged
    - e.g., XSS for Web interfaces
  - As software technology standardizes, so do the attacks
    - Just recently hackers pulled off a major break-in using a classic SQL injection

      Heartland Payment Systems 2009 breach compromised 130+ Mil accounts data

Interface-specific testing

- Why Interface-specific testing?
  - Interface-specific classes of attacks have emerged
    - e.g., XSS for Web interfaces
  - As software technology standardizes, so do the attacks
    - Just recently hackers pulled off a major break-in using a classic SQL injection

      Heartland Payment Systems 2009 breach compromised 130+ Mil accounts data

- Well-known classes of interface-specific attacks lead to standard frameworks of tests that are
  - naturally adapted to the type of interface
  - allow for state-of-the-art coupling with fuzzing for testing multilayered interfaces/protocols

Example: Well-known attacks/testing techniques for Web Interfaces

- Cross-Site Scripting (reflected, Stored, DOM based XSS)
- Session Hijacking (session fixation, session side-jacking)
- Cross-site Request Forgery (also known as session-riding)
- Path Reversal
- Code Injection (PHP, HTML, SQL Injection)
- Command injection (LDAP, XPath, XSLT, HTML, XML, OS)
- File inclusions
- Use of poor encoding practice (base 64)/ Insecure cryptographic storage
- Insecure direct object reference
- Information Leakage and Improper Error Handling

Combining Fuzzing w/ Well-Known Tests for Discovering Input-Based Vulnerabilities

- (Pseudo-)Randomly choose an input from the entire input space
- Invoke the application with that input
- Observe the resulting output
- Look for 'odd' behavior
- Exploit odd behavior

Example: HTTP Header Fuzzing

7K:>6]"=:&X<ZE`,`)7?:0=/'53#.DMO:/_2`RZN6QB9

GET M?40G);>@!5#/>L5P_`+\@V3WB+_2_ HTTP/1.0

GET http://www.foobar.com/M?40G);>@!5#/>L5P HTTP/1.0

GET http://www.foobar.com/so6gyhsiwgic.html HTTP/1.0

GET http://www.foobar.com/so6gyhsiwgic.pl HTTP/1.0

GET http://www.foobar.com/so6gyhsiwgic.ado HTTP/1.0

GET http://www.foobar.com/so6gyhsiwgic.jsp HTTP/1.0

GET http://www.foobar.com/so6gyhsiwgic.hs HTTP/

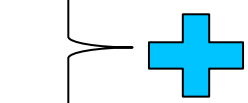
Our Goal

- Promote the development of an interface-based testing methodology for CC that
  - complements the general interface-independent testing methodology of CEM
  - maps easily to EAL levels
  - improves reproducibility of test results
  - enhances the value of the evaluation

Approaches to Adopting Interface-Based Testing in CC

- Develop testing-related assurance packages
  - combining fuzzing with interface-specific knowledge-based tests

- Modular assurance packages tailored to specific product types
  - e.g., Web product test package
    - Cross-Site Scripting (reflected, Stored, DOM based XSS)
    - Session Hijacking (session fixation, session side-jacking)
    - Cross-site Request Forgery (also known as session-riding)
    - Path Reversal
    - Code Injection (PHP, HTML, SQL Injection)
    - Command injection (LDAP, XPath, XSLT, HTML, XML, OS)
    - File inclusions
    - Use of poor encoding practice (base 64)/ Insecure cryptographic storage
    - Insecure direct object reference
    - Information Leakage and Improper Error Handling



Fuzzing on interface parameters

Modular assurance packages and EAL

Some Interfaces Tested by **Some** Interface-Specific Tests With **Some** Fuzzing

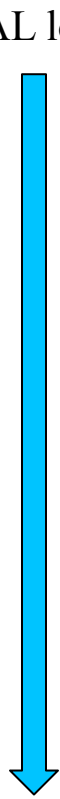
Most Interfaces Tested by **Some** Interface-Specific Tests With **Some** Fuzzing

Most Interfaces Tested by **Most** Interface-Specific Tests With **Some** Fuzzing

Most Interfaces Tested by **Most** Interface-Specific Tests With **More** Fuzzing

All Interfaces Tested by **Most** Interface-Specific Tests With **More** Fuzzing

All Interfaces Tested by **All** Interface-Specific Tests With **Most** Fuzzing



Benefits from modular test assurance packages

Benefits from modular test assurance packages

- For developers
  - Adopting state-state-of-the-art tests early in development cycle saves expensive bug fixes during product evaluation
  - Improves the quality of the product and helps avoid embarrassing post-release “discoveries”

Benefits from modular test assurance packages

- For developers
  - Adopting state-state-of-the-art tests early in development cycle saves expensive bug fixes during product evaluation
  - Improves the quality of the product and helps avoid embarrassing post-release “discoveries”
- For evaluators
  - Improves the likelihood of the discovery of critical security problems by shifting the focus for known attacks from AVA to ETE
  - Improves the repeatability of evaluations and addresses a weakness in the standard

Benefits from modular test assurance packages

- For developers
  - Adopting state-state-of-the-art tests early in development cycle saves expensive bug fixes during product evaluation
  - Improves the quality of the product and helps avoid embarrassing post-release “discoveries”
- For evaluators
  - Improves the likelihood of the discovery of critical security problems by shifting the focus for known attacks from AVA to ETE
  - Improves the repeatability of evaluations and addresses a weakness in the standard
- For consumers
  - Increases the security assurances provided by the product
  - Increases the value of certification

Conclusions

Conclusions

- Rigorously defined testing modules lead to state-of-the-art testing techniques

Conclusions

- Rigorously defined testing modules lead to state-of-the-art testing techniques

Conclusions

- Rigorously defined testing modules lead to state-of-the-art testing techniques

- Evaluators can reliably identify more security flaws and systematically increase the rigor of CC testing

Conclusions

- Rigorously defined testing modules lead to state-of-the-art testing techniques

- Evaluators can reliably identify more security flaws and systematically increase the rigor of CC testing

Conclusions

- Rigorously defined testing modules lead to state-of-the-art testing techniques
- Evaluators can reliably identify more security flaws and systematically increase the rigor of CC testing
- The definition of modular test packages can be formalized to integrate in CC