Open Ended Vulnerability Testing

Alec Yasinsac
Co-Director, Security and Assurance in Information Security Laboratory
Florida State University
Tallahassee, Florida 32306-4530

December 10, 2007

Abstract

In this paper, we examine fundamental Open Ended Verification Technology properties, noting encouraging areas where it can contribute to secure voting systems. We also identify misconceptions about OEVT virtues and capabilities.

Our contention is that OEVT can help to measure development process maturity, which is the best long term approach to ensure voting systems security.

Keywords: Electronic voting, verification, software testing

1. Introduction

Investigations such as those resulting in the Hopkins Report [1], California VSTAAB Report [2], SAIT Reports [3, 4], and the California Top To Bottom Review [5] have become the de facto standard electronic voting system security analysis approach. There is little doubt that these reviews have improved voting system security and positively advanced the voting system security discussion overall. Most of these reviews were a form of open-ended vulnerability tests (OEVT), also known as “penetration tests” or “red team tests”. The test teams were given access to software, hardware, or both and analyzed the system based on some basic testing principles and their expert skills and instincts.

Like any other test, OEVT can be effective for finding faults and vulnerability; however, it is woefully inadequate of demonstrating their absence [6]. The early OEVTs have been uniformly successful at finding faults, possibly because the nature of software makes it difficult to remove all faults, or more likely because of the low quality of the software that was under investigation. The reports to date reflect poor system design, implementation, and documentation practices that are reflected in voting system software. This signals that these developers are operating at a low process development maturity level.

As voting system developers mature their processes, software quality will improve and OEVT will be less likely to detect faults in the quantity that we see today. However, Dijkstra’s rule suggests that a negative testing result is necessarily vague, thus it is questionable what OEVT testing offers to the certification process in the long term.

As a testing technique, OEVT leverages the investigator’s strengths. Systems and security experts can often spot subtle red flags that identify or suggest underlying vulnerability. We suspect that there is no deterministic, or possibly even measurable, technique that can replicate or approximate open-ended analysis. The theoretical strength of OEVT is founded in the following four premises:

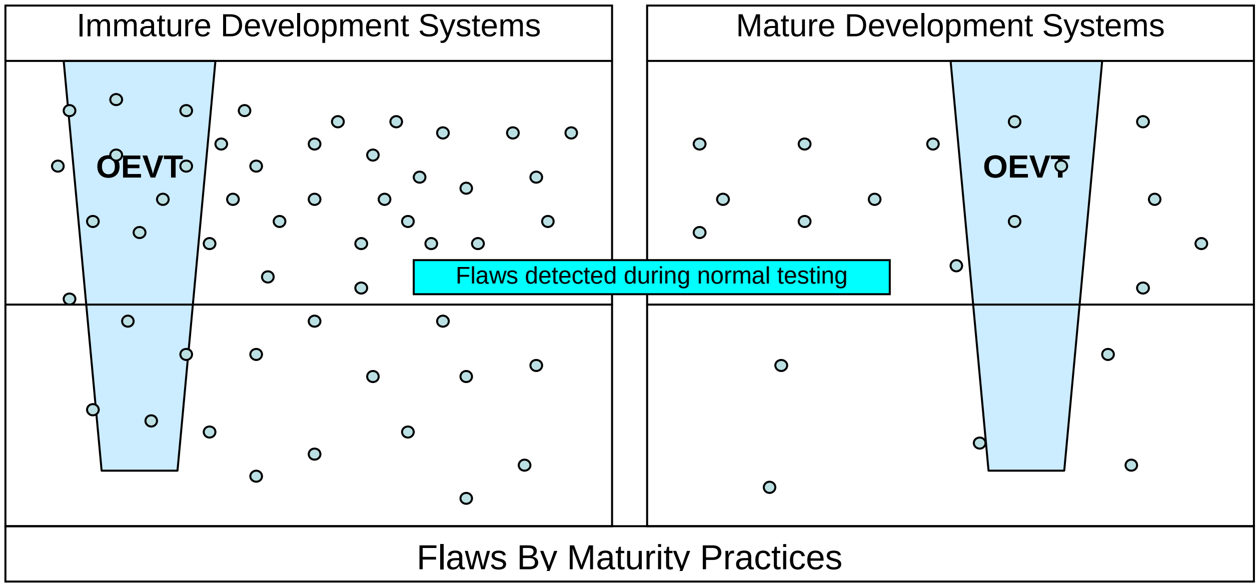
1. No testing technique can provide absolute security, reliability, or correctness assurance. Routine testing techniques generally provide broad, but shallow[1] fault detection.
2. An expert OEVT team can drill down well below the depth of traditional testing techniques in a narrow, focused analysis area defined by the team's skills to identify security vulnerability at depths that normal testing techniques cannot, or routinely do not, identify.
3. It is possible to consistently select OEVT teams that are sufficiently skilled and if they are provided with proper resources, if they do not find faults in their specific focus area, their inability to find faults is strong evidence that the developer's techniques are sufficiently mature to provide broad and deep security assurances.

These first three premises reveal that the true value of OEVT is to assess the vendor's development maturity by examining the product that manifests from their processes. The specific problems they find are incidental to this primary purpose.

4. The OEVT team plan cannot be known by the vendor a priori, else the primary OEVT result is invalid. If the vendor knows the OEVT test plan, they may target their development activities and their own testing processes to fit a specific, narrow OEVT technique, while leaving all other areas with routine development practices and shallow test coverage!

Thus, OEVT is, in a sense, a random sample approach, where the team's skills determine the channel-of-faults the team will detect, as is illustrated in the figure below. To be effective, the OEVT team cannot be preselected or identified to the developer in any way, so developers must produce systems that are uniformly high quality, rather than targeted for a narrowly deep security channel. The OEVT team's skill sets and evaluation approaches should only be decided after the product is submitted for certification and has cleared preliminary evaluation hurdles.

Similarly, allowing a vendor to correct faults identified in OEVT and then to resubmit to the same OEVT team would simply move the time when the vendor knows the test plan until after the first OEVT. Thus, OEVT would become nothing more than a narrowly focused debugging team and the most important value of OEVT would be lost.


Immature Development Systems
Mature Development Systems
OEVT
OEVT
Flaws detected during normal testing
Flaws By Maturity Practices

[1] This assumes that some fault types or classes are more difficult to detect than others. Our analogy here is that the analyst need only dig shallow to find the easy to detect fault classes, and more deeply for difficult to detect faults.

2. Why Test?

2.1.Theoretical Limits to Testing

Theoretically, testing is a fundamentally flawed security verification technology. Since there are an infinite number of potential flaws and/or attacks in non-trivial systems, testing can provide only infinitely small quantitative confidence in system quality or security.

Dijkstra famously codified this concept: "...program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence." [6]. Worse yet, testing cannot produce any well-founded estimate of the number of remaining faults, reinforcing the well-known cliché: "You can't know what you don't know".

Practically, on the other hand, rigorous testing allows us to produce systems that are fully acceptable in many areas. One challenge is to know how much, and what kind of testing produces optimal systems. Certainly, we must have very high testing standards for voting systems. Quantifying how much testing should be applied is extremely difficult. Too little testing and corresponding systems will not be trustworthy; too much and they will not be feasible.

Software Engineering theory has reacted to this fundamental testing challenge by introducing systematic attention to development process maturity to improve quality, e.g. through CMMI. There is no strong theoretical reason to incorporate OEVT into the certification process other than to measure the product's development process quality through assessment of the final product.

2.2.Testing for Certification

By its nature, OEVT a fundamental testing best practice in that it is subjective. While most OEVT teams will examine many of the same easy-to-detect errors, varying investigator skills will direct them to different channels for difficult-to-detect flaws. Thus, it is likely that different teams will find different flaws, illustrating the subjectivity.

These principles are important for development testing, but they are critical for acceptance or certification testing. If vendors do not have a standard to guide development, the development risk may exceed the potential return and even the highest quality vendors may abandon the market.

The only way that we see to mitigate this risk and for OEVT to be effective in a certification environment is if mature development processes are required for electronic voting system vendors. That is, only certified mature process developers should be allowed to submit products for certification. Under this scenario, OEVT can provide a back-end check for product quality that will be meaningful whether the team does or does not find faults.

2.3. Level of Effort

As we noted earlier, OEVT is an inherently subjective process, which is not usually a positive property of testing processes, let alone certification processes. The draft VVSG gives fixed numbers for team size and review duration, which gives an objective flavor to the standard. For example, vendors could conduct OEVT themselves according to the level of effort standard to help determine if their product is ready to be submitted for certification.

However, if there is no foundation for selecting the given fixed numbers, it may do more harm than good. If the goal is to establish a meaningful baseline, these numbers cannot be artificial or arbitrary. Rather, we may consider an "effort heuristic" based on a concrete property that can be consistently applied across systems. For example, Lines of Code (LoC), number of modules, and Halstead number are some well-known metrics that could individually or in combination provide the basis for establishing a consistent baseline across different products.

2.4. Interpreting OEVT Results

2.4.1. Fail Criteria

The purpose of the fail criteria is no clear in the VVSG draft. If this criteria is established as a firm certification failure, it seems to be far out of place, as certification should be decided based on a wide view of all processes, including mitigating procedures that may accompany the product. An OEVT team is not in a position to make this decision.

Nonetheless, as it is written, the fail criteria as described for the OEVT process is vague and provides analysts wide authority to make arguments regarding security properties and prospective attacks. Under this fail criteria, it may be possible for a team to fail any voting system (including PCOS and even paper ballots-hand counted) based only on its architecture.

OEVT occurs at the end of the development cycle, so vendors will hesitate to develop new systems using novel architecture, or even new systems under a previously proven architecture, if it is possible for OEVT failure based only on describing a plausible attack scenario, where the OEVT team does not have to develop an attack and demonstrate the exploitation of the vulnerabilities or errors they find.

We propose that the VVSG include a corresponding, foundational "pass criteria" that would provide a "trusted floor" where vendors could begin development with a reasonable expectation of not-failing the OEVT.

2.4.2. Other-Than-Fail Interpretation

Dijkstra’s principle stands unchallenged regarding the meaning of a test’s failure to find faults. Thus, it is not clear what it would mean if an OEVT team does not find faults that meet the fail criteria.

Since testing cannot detect all, or even a reliably predictable percentage of remaining faults, its only value is to determine if the development process naturally produces robust systems.

2.4.3. Retesting With OEVT

The value of OEVT is that it is a general approach whose techniques cannot be predicted. This prevents vendors from shielding low quality software by meeting deep but narrow testing standards.

The effect of direct re-testing would be that the vendor is given the testing standard after initial OEVT reporting, which relegates OEVT to nothing more than beta testing. Certification based on fixing the identified faults violates the foundational premise of OEVT.

Re-testing also injects naturally conflicting interests from which the present Independent Test Authority (ITA) process suffers.

Thus, if a product fails OEVT, it must be returned to the vendor for fundamental redevelopment.

3. What’s Missing: Structured Note Taking.

We contend that it is hard to overstate the value of structured note-taking during the OEVT process and then making the notes database a work-product of each review. The level of continuity it provides between reviews justifies including it as a VVSG requirement. There are two other benefits that may be just as important:

1. Process Improvement. Understanding the details of the process that each team goes through can be a gold mine of best practices.
2. Accountability. OEVT is critically dependent on the investigators. Structured note taking provides an avenue to analyze the team's effort.

4. Summary

Open-ended testing is an excellent tool for identifying system flaws, even flaws that are well-hidden and that may not be revealed through more structured testing processes. The value is that if a system is built with weak development standards, OEVT can detect flaws hidden beneath the standard-testing surface.

We contend that the specific flaws or faults that OEVT detects are incidental to the primary result; rather, the most important OEVT result is to assess maturity of the development process by analyzing the product that it produces. These results should be tracked and matched with a development process maturation standard that can provide the security properties demanded of voting systems.

5. Acknowledgment

Thanks to Matt Bishop and David Wagner for their helpful discussions and comments.

6. References

1 Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach, “Analysis of an Electronic Voting System”, IEEE Symp. on Security & Privacy, May 9-12, 2004, pp. 27-40

2 David Wagner, David Jefferson, Matt Bishop, "Security Analysis of the Diebold AccuBasic Interpreter", Voting Systems Technology Assessment Advisory Board, http://www.ss.ca.gov/elections/voting_systems/security_analysis_of_the_diebold_accubasic_interpreter.pdf

3 A. Yasinsac, D. Wagner, M. Bishop, T. Baker, B. de Medeiros, G. Tyson, M. Shamos, and M. Burmester, “Software Review and Security Analysis of the ES&S iVotronic 8.0.1.2 Voting Machine Firmware, Final Report”, Security and Assurance in Information Technology Laboratory, Florida State University, February 23, 2007, http://election.dos.state.fl.us/pdf/FinalAudRepSAIT.pdf

4 Ryan Gardner, Alec Yasinsac, Matt Bishop, Tadayoshi, Kohno, Zachary Hartley, John Kerski, David Gainey, Ryan Walega, Evan Hollander, and Michael Gerke, “Software Review and Security Analysis of the Diebold Voting Machine Software”, Final Report For the Florida Department of State, July 27, 2007, http://election.dos.state.fl.us/pdf/SAITreport.pdf

5 California Secretary of State, UC Final Reports for the Top-to-Bottom Review (July-Aug. 2007); available at http://www.sos.ca.gov/elections/elections_vsr.htm.

6 Edsger W. Dijkstra, “The Humble Programmer”, Commun. ACM 15 (1972), 10: 859–866, http://www.cs.utexas.edu/~EWD/ewd03xx/EWD340.PDF