On the notion of "software-independence" in voting systems

By Ronald L. Rivest
John P. Wack

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology (MIT)
Cambridge, MA 02139
`rivest@mit.edu`

National Institute of Standards and Technology (NIST)
Information Technology Laboratory
Gaithersburg, MD 20899
`john.wack@nist.gov`

This paper defines and explores the notion of "software-independence" in voting systems:

> A voting system is *software-independent* if an undetected change or error in its software cannot cause an undetectable change or error in an election outcome.

For example, optical scan and some cryptographically-based voting systems are software-independent. Variations and implications of this definition are explored. We propose that software-independent voting systems should be preferred, and *software-dependent* voting systems should be avoided.

An initial version of this paper was prepared for use by the Technical Guidelines Development Committee (TGDC) in their development of the Voluntary Voting System Guidelines (VVSG), which will specify the requirements that U.S. voting systems must meet to receive certification.

Keywords: Security, voting, software-independence.

1. Introduction

The main purpose of this paper is to introduce and carefully define the terminology of "software-independent" and "software-dependent" voting systems, and to discuss their properties. This paper is definitional in character; there are no "results" as such. Our goal is to provide crisp terminology for discussing voting systems, with the view that such terminology will be useful in the next version of the VVSG, currently under development by the U.S. Election Assistance Commission (EAC), the TGDC, and the National Institute of Standards and Technology (NIST). The TGDC has recommended software-independence as a requirement in the VVSG (Technical Guidelines Development Committee, 2007).

We start by describing the problem that software-independence addresses: the difficulty of assuring oneself that voted ballots will be recorded accurately by complex and difficult-to-test software in all-electronic voting systems. We emphasize that the problem is providing such assurance: the software may well be correct, but convincing oneself (or others) that this is the case is effectively impossible. We then define what constitutes a software-independent approach to voting system design. We provide examples and discuss relevant issues.

This paper is intended to stimulate discussion. It is not an official document of any organizations we may be associated with (e.g. MIT, NIST, or the TGDC) nor does it pretend to represent official positions of any of these organizations.

2. Problem: Software complexity of voting systems

Electronic voting systems are complex and continue to grow more so. The requirements for privacy for the voter, for security against attack or failure, and for the accuracy of the final tally are in serious conflict with each other. It is common wisdom that complex and conflicting system requirements leads to burgeoning system complexity.

Voting system vendors express and capture this complexity via software in their voting systems.

As an example, consider a Direct-Recording Electronic (DRE) voting system, which typically provides a touch-screen user interface for voters to make selections and cast ballots, and which stores the cast vote records in memory and on a removable memory card. A DRE may display an essentially infinite variety of different ballot layouts, and may include complex accessibility features for the sight-impaired (e.g., so that a voter could use headphones and be guided to make selections using an audio ballot).

At issue, then, is how to provide assurance, despite the complexity of the software, that the voting system will accurately record the voter's intentions. A pure DRE voting system produces only electronic cast ballot records, which are not directly observable or verifiable by the voter.

Consequently, no meaningful audit of the DRE's electronic records to determine their accuracy is possible; accuracy can only be estimated by a variety of other (imperfect) measures, such as comparing the accumulated tallies to pre-election canvassing results, performing software code reviews, and testing the system accuracy before (or even during) the election.

(*a*) *The difficulty of evaluating complex software for errors*

It is a common maxim that complexity is the enemy of security and accuracy, thus it is very difficult to evaluate a complex system. A very small error, such as a transposed pair of characters or an omitted command to initialize a variable, in a large complex system may cause unexpected results at unpredictable times. Or, it may provide a vulnerability that can be exploited by an adversary for large benefits.

Finding all errors in a large system is generally held to be impossible in general or else highly demanding and extremely expensive. Our ability to develop complex software vastly exceeds our ability to prove its correctness or test it satisfactorily within reasonable fiscal constraints (extensive testing of a voting system's software would certainly be cost-prohibitive given how voting in general is funded). A voting system for which the integrity of the election results intrinsically depends on the correctness of its software will always be somewhat suspect.

As we shall see, the software-independent approach follows the maxim, "Verify the election results, not the voting system."

(*b*) *The need for software-independent approaches*

With the DRE approach, one is forced to trust (or assume) that the software is correct. If questions arise later about the accuracy of the election results (or if a recount is demanded), there is again no recourse but to trust (or assume) that the voting system did indeed record the votes accurately. We feel that one should strongly prefer voting systems where the integrity of the election outcome is not dependent on trusting the correctness of complex software.

The purpose of this paper is to define a new notion, that of "software-independence," that captures this desirable characteristic of providing election results that are verifiable, without having to depend on the assumption that the software is correct.

For users of "software-independent" voting systems, verification of the correctness of the election results is possible. There need be no lingering unanswerable concern that the election outcome was affected or actually determined by some software bug (or worse, e.g., by a malicious piece of code).

3. Definition and rationale for software-independence

We now repeat the definition of software-independence, and explore its meaning.

> A voting system is *software-independent* if an undetected change or error in its software cannot cause an undetectable change or error in an election outcome.

A voting system that is not software-independent is said to be *software-dependent*—it is, in some sense, vulnerable to undetected programming errors, malicious code, or software manipulation, thus the correctness of the election results are dependent on the correctness of the software.

To illustrate the rationale for software-independence, let us run a few "thought experiments." Put yourself in the place an adversary and imagine that you have the power to secretly replace any of the existing software used by the voting systems by software of your own construction (you may assume that you have the source code for the existing software).

With such an ability, can you (as the adversary) change an election outcome or "rig an election" without fear of detection?

If so, the system is *software-dependent*—the software is an "Achilles heel" of the voting system. Corrupting the software gives an adversary the power to secretly and silently steal an election.

If not, the system is *software-independent*—the voting system as whole (including the non-software components) has sufficient redundancy and potential for cross-checking that misbehavior by the software can be detected. The detection might be by the voter, by an election official or technician, by a post-election auditor, by an observer, or by some member of the public. (Indeed, anyone but the adversary.)

In these "thought-experiments," we are considering the adversary as some evil agent that could load fraudulent software into voting systems. More realistically, we may consider this adversary to be an abstraction of the limitations of the software development process and testing process. (As such, for the purposes of determining whether a system is software-independent, one should presume that the software errors were present when the software was written and were not caught by software development control processes or by the certification process.)

As we have stated, complex software is difficult to write and to test, and will therefore contain numerous unintentional "bugs" that occasionally can cause voting systems to report incorrect election results. It would be extremely difficult and expensive to determine with certainty that a piece of software is free of bugs that might change an election outcome. Given the relatively small amounts of funding allocated for developing and testing voting system software, we may safely consider it as *effectively impossible.*

These notions are not new—others have discussed the problems associated with using complex software in voting systems. Yet, we have heretofore lacked crisp terminology for talking about the dependence of election outcomes on such complex software.

(*a*) *Refinements and elaborations of software-independence*

There are a number of possible refinements and elaborations of the notion of software-independence. We now motivate and introduce the distinction between *strong software-independence* and *weak software-independence.*

Security mechanisms are typically one of two forms: *prevention* or *detection.* Detection mechanisms may also be coupled with means for *recovery.* When identification of participants and accountability for actions is also present, then detection mechanisms are also the foundation for *deterrence.* Given the importance of recovery mechanisms in addition to detection mechanisms, we propose the following two definitions:

A voting system is *strongly software-independent* if an undetected change or error in its software cannot cause an undetectable change or error in an election outcome, and moreover, a detected change or error in an election outcome (due to change or error in the software) can be corrected without re-running the election.

A voting system that is *weakly software-independent* conforms to the basic definition of software-independence, that is, there is no recovery mechanism.

(*b*) *Examples of software-independent approaches*

Currently, there are two general categories of software-independent approaches. *Voter-verifiable paper record (VVPR) approaches* constitute the first category, since the VVPR allows (via a recount) the possibility of detecting (and even correcting) errors due to software. Accordingly, these voting systems can be strongly software-independent.

The most prominent example in this category is the optical scan voting system used by most U.S. voters since the 2006 elections. The paper ballot is voter-verifiable because the voter completes the ballot and can attest to its accuracy before it is fed into the optical scanner; the paper ballot thus serves as an audit trail that can be used in post-election audits of the optical scanner's electronic results. An electronic ballot marking system (EBM) may also be used to record the voter's choices electronically with a touch-screen interface and then to print a high-quality voter verifiable paper ballot for feeding into the optical scanner.

Another example in this category is the voter-verified paper audit trail (VVPAT) voting system, similar to a DRE but with a printer and additional logic. It produces two records of the voter's choices, one on the touch-screen display and one on paper (a VVPR). The voter must verify that both records are correct before causing them to be saved.

Cryptographic voting systems constitute the second category of software-independent voting system approaches. They can provide detection mechanisms for errors caused by software changes or errors (Adida 2006, Chaum 2004, Chaum et al. 2004, Karloff et al. 2005, Neff 2004, Ryan & Peacock 2005, Ryan & Schneider 2006) At one level, they can enable voters to detect when their votes have been improperly represented to them at the polling site, and a simple recovery mechanism (re-voting) is available. At another level, they can enable anyone to detect when their votes have been lost or changed, or when the official tally has been computed incorrectly. Recovery is again possible. Most of the recently proposed cryptographic voting systems are strongly software-independent.

Receipt-based cryptographic voting systems involve a physical, e.g., paper receipt that the voter can use to verify, during the process of voting, whether his or her ballot was captured correctly. The contents of the receipt, in general, employ cryptography in some form so that the voter is able to verify that the votes were recorded accurately; the receipt does not show how the voter voted.

Approaches to software-independence other than pure use of VVPR or cryptographic voting systems are potentially possible, although beyond the scope of our paper.

4. How does one test for software-independence?

This brings up a more subtle point in the definition. What aspects of the voting system make it "software-independent?" Is it just the hardware and software, or does it also include the surrounding procedures? For example, is a voting system still software-independent if no post-election audits are performed?

The answer is that a voting system is software-independent if, after consideration of its software and hardware, it enables use of any election procedures needed to determine whether the election outcome is accurate without having to trust that the voting system software is correct. The election procedures could include those carried out by voters in the course of casting ballots, or in the case of optical scan and VVPAT, they could include election official procedures such as post-election audits.

The detection of any software misbehavior does not need to be perfect; it only needs to happen with sufficiently high probability, in an assumed ideal environment with alert voters, pollworkers, etc.

As an example, consider the EBM which prints out a filled-in optical scan ballot. Some voters may not review the printed ballot at all. Yet the EBM is still software-independent; there is a significant probability that software misbehavior by the EBM will be detected (this is similarly true of VVPAT). For the purposes of the definition of "software independence," we assume that (enough) voters are sufficiently observant to detect such misbehavior. (If this assumption were discovered to be false in practice, some increase in voter education might be necessary.) Although some forms of such detectable misbehavior may leave no tangible proof of misbehavior, the definition of software independence does not require that all misbehavior have tangible proof; it is sufficient that the relevant misbehavior be detectable and reportable.

Continuing with this example, we note that there is also software in the optical scanner used to scan the ballots that might produce incorrect output. But such misbehavior is detectable by a post-election audit procedure that hand-counts the paper ballots, thus the optical scan voting system is software-independent. (Note that such audits are typically statistical in nature and are thus not perfect detectors of misbehavior. But a well-designed audit will catch such misbehavior with reasonable probability (Aslam et al. 2007, Hall 2007).

To illustrate further, then, say that no post-election audit of an optical scan-based election is required if the apparent margin of victory is more than 10%. An optical scan system would be still be considered software-independent in such an election, since the original voter-verified paper ballots are available for review, and software misbehavior can still in principle be detected. (As a side note: we feel that such post-election audits are always a good idea and that "no audit" should not be an option. If an apparent margin of victory is large, a smaller audit is appropriate.)

As a final example, say that electronic pollbook systems are used in an optical scan-based election, but the electronic pollbooks do not create a contemporaneous paper record for each voter. Thus, their software must be trusted to show that the number of optical scan records (paper and electronic) accurately reflect the actual number of voters who used the scanners. Are these systems software-independent? We would argue that the answer is no for the electronic pollbook, as the design of this system has prevented an audit to determine if the number of optical scan records is correct, i.e., its software must be trusted to be correct. A contemporaneous paper record would have made the electronic pollbook software-independent.

5. Discussion

(*a*) *Are other* software-dependent *alternatives sufficient?*

Other alternatives to software-independence raised thus far have confused the main motivator for software-independence with other security issues, including:

1. *Parallel testing* (Jones) which is often cited as an efficient and accurate gauge of the correct operation of a voting system and, by implication, the correctness of its software. However, relying upon it to detect errors in an election outcome would require that it be done in a very comprehensive manner for each use of the voting system, which is impractical. Parallel testing can be at best an approximate gauge of software accuracy; it is also problematic against malicious code, since it relies on an assumption (not supported in our software-independence framework) that voters are not signalling to the malicious code as to whether it is in "test" mode or running a real election.

2. *Software verification of certified voting system software*, which can be used to determine that the voting system is running the software that it is supposed to, i.e., the correct version of federally-certified software (it must be emphasized that this is not possible in today's voting systems). It cannot ensure that a software-dependent approach is using error-free software.

3. *Use of independent multiple voting systems operating synchronously*, which has been proposed as a method for producing multiple sets of cast ballot records that could be trusted to be correct if the systems are produced by different vendors and connected via standard interfaces. Practically speaking, requiring that different vendors produce the systems would be difficult at best and not likely to counter the software-dependent approaches of both systems.

(*b*) *Implications for testing and certification*

Given the exceptional difficulty of proving software to be correct, it is a reasonable proposal to disallow voting systems that are software-dependent altogether.

If testing and certification of software-dependent voting systems are to be nonetheless contemplated, then one should expect the certification process should be very much more demanding and rigorous for a software-dependent voting system than for a software-independent voting system. The manufacturer should submit a formal proof of correctness, with perhaps an assurance level corresponding to EAL level 6 or 7 (Common Criteria, EAL), and public disclosure of the source code. Moreover, the voting system must permit proof it is running the software it is supposed to.

(*c*) *Related issues*

There may be other aspects of software misbehavior that don't quite fit our proposed notion of software-independence. For example, software may bias a voter's choices in subtle ways (say by displaying one candidate's name is slightly brighter characters on a touch-screen). These issues fall outside the scope of software-independence, since the correct "election outcome" isn't well-defined until the voter indicates her choice.

6. Conclusions and suggestions

The history of computing systems is that, given improvements and breakthroughs in technology and speed, software is able to do more and thus its complexity increases. The ability to prove the correctness of software diminishes rapidly as the software becomes more complex. It would effectively be impossible to adequately test future (and current) voting systems for flaws and introduced fraud, and thus these systems would always remain suspect in their ability to provide secure and accurate elections.

A *software-independent* approach to voting systems will provide voters with an assurance that errors or fraud in election results can reliably be detected. Testing costs to prove the correctness of the software can be held somewhat in check if, fundamentally, the correctness of the election results does not rely on the correctness of the software.

Acknowledgments

We'd like to thank John Kelsey and Ka-ping Yee for helpful comments and suggestions.

References

Adida, B. 2006. *Verifying Secret-Ballot Elections With Cryptography.* PhD thesis, MIT Department of EECS.

Aslam, J. & Popa, R. A. & Rivest, R. L. 2007 On auditing elections when precincts have different sizes, December 18, 2007. `http://people.csail.mit.edu/rivest/AslamPopaRivest-OnAuditingElectionsWhenPrecinctsHaveDifferentSizes.pdf`.

Chaum, D. 2004. Secret ballot receipts: True voter-verifiable elections. *IEEE J. Security and Privacy*, pages 38 – 47, Jan/Feb 2004.

Chaum, D. & Ryan, P. Y. A. & Schneider, S. A. 2004. A practical, voter-verifiable election scheme. Technical Report CS-TR-880, University of Newcastle upon Tyne School of Computing Science, December 2004. `http://www.cs.ncl.ac.uk/research/pubs/trs/papers/880.pdf`.

Common criteria evaluation and validation scheme. `http://niap.bahialab.com/cc-scheme/`.

Common criteria assurance levels. `http://www.cesg.gov.uk/site/iacs/index.cfm?menuSelected=1&displayPage=13`.

Hall, J. L. 2007. Post-election manual auditing of paper records: Bibliography, 2007. `http://www.josephhall.org/papers/auditing_biblio.pdf`.

Jones, D. Parallel testing during an election. `http://www.cs.uiowa.edu/~jones/voting/testing.shtml#parallel`.

Karlof, C. & Sastry, N. & Wagner, D. 2005. Cryptographic voting protocols: A system perspective. In *Proceedings 14th USENIX Security Symposium*, August 2005. `http://www.cs.berkeley.edu/~nks/papers/cryptovoting-usenix05.pdf`.

C. Andrew Neff. 2004 Practical high certainty intent verification for encrypted votes, October 2004. `http://www.votehere.com/vhti/documentation/vsv-2.0.3638.pdf`.

Ryan, P. Y. A. & Peacock, T. 2005. Prêt à Voter: A system perspective. Technical Report CS-TR-929, University of Newcastle upon Tyne School of Computing Science, September 2005. `http://www.cs.ncl.ac.uk/research/pubs/trs/papers/929.pdf`.

Ryan, P. Y. A. & Schneider, S. A. 2006. Prêt à Voter with re-encryption mixes. Technical Report CS-TR-956, University of Newcastle upon Tyne School of Computing Science, April 2006. `http://www.cs.ncl.ac.uk/research/pubs/trs/papers/956.pdf`.

Technical Guidelines Development Committee (TGDC). VVSG recommendations to the EAC, August 31 2007. `http://vote.nist.gov/vvsg-report.htm`.