Toward Clarifying Election Systems Standards

Earl Barr Matt Bishop Dimitri DeFigueiredo Mark Gondree Patrick Wheeler
Department of Computer Science
University of California at Davis
{barr,bishop,defigued,gondree,wheelerp}@cs.ucdavis.edu

September 27, 2005

1 Introduction

Throughout the world, electronic voting machines are recording and tallying votes during elections with increasing regularity. In the United States, most election jurisdictions use systems that conform to standards originally promulgated by the Federal Election Commission (FEC). Although the federal government does not require conformance, most states have passed laws that do.

The goal of the standards is to "address what a voting system should reliably do, not how system components should be configured to meet these requirement" [7, Vol I §1.1]. The standards present a certification procedure involving testing by Independent Testing Authorities (ITAs or, simply, Testing Authorities), and many jurisdictions cannot use systems that are uncertified. Virtually all vendors have this testing done. Nevertheless, there have been many instances of certified election systems being "broken" or performing unreliably. Why?

One goal of this work is to answer the question: if systems that meet the standards can be induced to provide inaccurate or unreliable results in an election, is the problem that the standards are not adequate or is the problem that the testing is inadequate? If the standards are inadequate, or the testers fail to test the systems adequately, the problems that we see now will continue.

A complementary goal of this paper is to show how threat modeling can lead to clearer standards and help structure the testing and review of an automated election system. The lack of a detailed threat model leads to an inability to determine if the required security mechanisms provide adequate protection against attempts to compromise the electronic voting systems—or, indeed, what "compromise" means. The current certification process—involving the standards, vendors, and ITA—does not include threat modeling or threat identification. It is not immediately apparent how these processes from commercial software development can be integrated into the certification process.

This paper proposes a division of labor designed to integrate these critical phases of secure software development into the certification procedure. In summary, the standards should require that vendors provide both threat and system models, and suggest partial models when appropriate. For example, certain threats are common to all jurisdictions and the standards should require vendors to include them in their threat model. The vendors should provide models for their specific systems, so that others can see what threats their systems are designed to counter and how those systems counter those threats. Vendor documentation should note those threats they

deem unrealistic or unthwartable. In the former case, the vendor should explain why the threat is unrealistic; in the latter case, the vendor should discuss ameliorations. The ITA should validate the vendor's models against those of the standards, to ensure the vendor handles the threats identified in the standard. It should also ensure that the vendor's models are coherent, correct, and a fair abstraction of the system. The ITA should then try to identify vulnerabilities in the system and validate that the implementation of the system, when combined with the vendor's recommended policies and procedures, counters the threats (or minimizes them, if the threat cannot be countered).

These practices, combined with code review, low-level threat inspection, and penetration testing, would cause the voting systems development process to follow the widely-accepted stages of secure system design and deployment (see [2, Ch. 18–19] or [31, Ch.1]).

This increases the assurance that the system is secure with respect to the threats identified in the standards and in the vendor's documentation. This contrasts to the current state of the standards, which ensure that only a minimal set of features are present in the end product.

The major contributions of this work are to provide

1. a precise, measured critique of existing standards
2. an answer to the previously posed question: are the standards inadequate or is testing insufficient?
3. documentation requirements that, in effect, force system development to include a design phase
4. a presentation of threat modeling that can be integrated into the certification process
5. clear examples of process models and high-level threats relevant to election systems

In Section 2, we provide background relating to voting requirement research and various voting standards. In Section 3, we review previous research about DRE security and the adequacy of the voting standards. In Section 4, we look at the current voting standards in depth, with respect to real problems identified in certified DREs. In Sections 5 and 6 we describe a process of systems modeling and then threat modeling that can be integrated into the standards and the certification process. By providing specific examples, we show how this process might have prevented existent DRE problems, like those examined in Section 4. We provide some brief comments on how to extend the work provided in this paper to address other existent and potential DRE problems, to model other election subprocesses, and to address other threats. In Section 7, we summarize and conclude.

2 Background

Developing electronic voting machines is an exercise in assurance. The goal is to convince a target audience that the machines are at least as accurate as non-electronic methods.

Of the many types of voting machines, this paper considers two. The first are systems that record votes electronically on the system. These votes may also be recorded on external media, such as flash cards, and may be recorded in multiple locations. This type of system is called a "Direct Recording Electronic" machine, or DRE. The system may, or may not, also create an external representation of the vote that the voter can verify. The

second type is the server that accepts vote records from the DREs and counts them. This server may connect directly to the DREs, or may count the totals from external media generated by the DRE. Vendors making these types of voting machines include Diebold and Sequoia.

The problem of convincing a target audience is one of *assurance*, defined as "confidence that an entity meets its security requirements, based upon specific evidence provided by the application of assurance techniques" [2, p. 478]. From this, we can see the two key concepts in defining assurance: *assurance techniques* and *requirements*.

2.1 Assurance and the System Life Cycle

The *system life cycle* models the development of systems. There are many such models; all have certain elements in common. The archetypal model is called the *waterfall life cycle model*, and we discuss it because it lends itself readily to the development of high assurance systems.

The waterfall life cycle consists of several phases. Each phase feeds back to earlier phases, as problems arising in the later phases require changes to work done in earlier phases.

The five phases are:

1. Requirements definition and analysis;
2. System and software design;
3. Implementation and component testing;
4. Integration and system testing; and
5. System deployment, operation, and maintenance.

At each step, evidence is gathered so that external evaluators can determine the level of assurance. The evidence may be minimal, for example a simple argument that the seems to satisfy the requirements, the design was done with the code in mind, the implementation seems correct, the testing that was done showed the system worked, and the documentation shows how to operate the system. For more critical systems, the evidence of assurance would need to be *much* more rigorous. For example, the developers would have to show how the requirements could be traced through the design, the implementation, the testing, and the operation so the evaluators could see that all requirements are met at all stages of development and testing. Formal specification of the requirements, and mathematical proofs that the design satisfies the requirements, are examples of evidence for high assurance systems. Testing, such as penetration ("red team") testing, provides evidence of assurance in the last three steps; the degree of assurance provided depends on the thoroughness of the testing and the conditions under which the testing is done.

2.2 Requirements

A requirement is a specific property that the system must satisfy. Requirements are derived both from the desired function of the system and from the environment in which the system will function. For example, in electronic voting, a requirement that the votes be counted correctly comes from the desired function of the

system. A requirement that the system be usable by voters with disabilities comes from the environment, because some voters will be handicapped, and the handicap must not prevent a voter from voting.

The relationship among requirements is affected by the environment in which the system is to work. For example, in voting, which is more important: knowing accurate results or never knowing how a particular individual voted? Either of these requirements can be met with a very high degree of assurance. To ensure that the ballots reflect the voters' desires and are counted accurately, each voter could publicly commit to their ballot's accuracy by announcing its contents before a notary, and all legitimate, signed and notarized ballots could be publicly tallied. This meets the first requirement, but violates the second. To ensure that no-one (except the voter) can determine how a particular individual voted, the voter can deposit her ballot in a shredder immediately after filling it out. This meets the second requirement, but violates the first.

Complicating the derivation of requirements is the multiplicity of laws in various jurisdictions. Within California, for example, some jurisdictions use majority election while others use choice voting; among those that use choice voting, jurisdictions may even differ in their specific version of choice voting. Thus, requirements for electronic voting machines vary among different jurisdictions, even within the same state.

Some requirements, however, are common among all jurisdictions. There has been work devoted to determining these common voting requirements [2, 16, 23, 26]. It is important to note, however, that all of these common requirements are necessary but insufficient: any correct, secure election system will meet these requirements, but there may be insecure systems that also meet these requirements. A list of necessary and testable requirements, though insufficient, is the most useful thing the standards can hope to attain. This limitation is a consequence of Rice's Theorem from computer science. This theorem states that it is impossible to decide (i.e., test in a principled way) if a machine, like those voting systems allowed by the standards which run arbitrary software on general purpose processors, has a non-trivial property, such as being both correct and secure. So, theoretically, there is no useful list of requirements to decide if any voting system is both correct and secure.

Models reflect requirements. A model provides a basis for asserting that something (in this instance, a voting system) meets the necessary requirements. For our purposes, we must consider two different types of models: a *system model* (described in Section 5) and a *threat model* (described in Section 6).

2.3 E-Voting Standards

A *standard* is a model developed to satisfy a specific set of requirements. If a system meets a standard, it will meet the set of requirements that the standard deals with, *provided* the standard is correct (that is, the standard meets the requirements). Some well-known standards are the (now defunct) TCSEC (also called the Orange Book), which was used to rate system security by the U.S. Department of Defense; FIPS 140-2, which provides the U.S. government security requirements for cryptographic modules; and the Common Criteria, which provides a broad and extensible set of security functionality and assurance requirements for commercial and government systems.

Mercuri [20] has argued that general standards such as the Common Criteria cannot be applied to electronic voting systems. For example, such a standard must enable one not only to assess system dependencies (which the Common Criteria allows), but also to assess conflicts among the requirements (which the Common Criteria does not provide).

Three standards relevant to DREs exist. The first [8], developed in the late 1980s for "electronic voting systems," including optical scan devices, was promulgated by the FEC in 1990 and presented performance and

testing standards for DREs. Recognizing that these standards were becoming dated, the FEC revised these in a set of standards promulgated in 2002 [9]. The goal of both sets of standards was to "assure the public of the integrity of computer-based machines." In 2005, the Election Assistance Commission (EAC), aided by the Technical Guidelines Development Committee (TGDC) and the National Institute for Science and Technology (NIST), circulated a third standards document [7], which is currently in the phase of receiving public review. It is essentially a minor revision of the 2002 FEC standards.

3 Related Work

Vora *et al.* [32] note that the standards "fail to precisely define the properties that should be required of a voting system." In a note, Mercuri pointed out that the standards do not "effectively address the varying and sometimes inconsistent or incompatible state and municipal regulations" and that the testing recommendations "primarily consist of functional testing and documentation reviews," with little review of assurance [21]. Our work supports their points by tying the lack of a model directly to ambiguities in the standards, and problems with the testing.

In mid-2003, a group of researchers from Johns Hopkins University examined source code obtained from an FTP site that appeared to be source code for Diebold's AccuVote-TS DRE system. Rubin *et al.* found the software deficient in the realm of security [18]. Their threat model assumed malevolent insiders, and they did not consider mitigations from procedural mechanisms.

The vendor, Diebold, responded that the code represented an old version of their software, and several issues, such as the passwords hard-coded in the source, "had been resolved in subsequent versions of the software" [6, p. 11]. Further, they pointed out that the Hopkins researchers did not test the software in the environment in which it was to be used, and that many of the attacks proposed were infeasible. Finally, they claimed to have developed the software using standard software engineering techniques, and that the software had passed rigorous certification checks.

The state of Maryland had purchased 16,000 DREs from Diebold to be used in the March 2004 primary election. To determine the severity of the problems highlighted by the Hopkins report, the state commissioned a study by the SAIC firm. This firm had experience in security analyses. The SAIC report [28] made 169 recommendations, primarily with respect to policies, procedures, and management. SAIC did not review any source code, and deemed the technical requirements met based upon the integrity of the Diebold software and the integrity of the off-the-shelf Microsoft Windows CE operating system (for the DREs) and Windows 2000 (for the vote counting server).

The SAIC report agreed that many of the attacks suggested by the Hopkins report would not work. For example, when a voter appears at the precinct, she is given a voter card containing an electronic authorization. To vote, she enters a booth, inserts the card in the DRE, and votes. The DRE invalidates the card (so it cannot be used until reauthorized by a poll worker) once the vote is cast. One attack described in the Hopkins report involved a voter smuggling a voter card into the booth and voting twice. The SAIC report stated that the sound of the cards being ejected would be loud enough for poll workers to hear, and they would notice that two such sounds came from a booth with one voter. They would thus detect this attack.

The state of Maryland, unsatisfied with the SAIC report, commissioned a third study. Unlike the Hopkins and SAIC reports, the Maryland State Board of Elections had RABA Technologies set up a mock precinct, with functioning DREs and a central vote counting server (the GEMS system). RABA assembled a team of

security analysts and had them attack the systems, the goal being to compromise a mock election. The team was given access to the source code for one week, and had one day to attack (the mock election day). The RABA study [25] found that, while procedural controls could mitigate many of the problems on a short-term basis, the security was completely inadequate, and bringing the security up to acceptable levels for the long term would require a complete redesign and re-implementation of the systems.

As an example, the team found that the threat of multiple votes by a single voter was quite realistic, as the noise from the ejection of the card could not be heard across the room used as a mock precinct. The team also compromised security based on the physical construction of the systems, including removing exposed wires to disable DREs[1], picking locks protecting the compartment containing the flash cards used to hold cast votes, and using the PS2 port to connect keyboards to the DRE. Further, the team gained Administrator access to the GEMS server easily, using a widely-known attack.

Similar studies by Compuware [3] and InfoSENTRY [15], both commissioned by the state of Ohio, corroborated and extended the above reports.

Experience with DREs has proven the validity of the results of these tests. For example, in November 2003, Boone County, IN election officials reported that stored voting machine data showed around 144,000 votes cast in a county with less than 19,000 registered voters [30]. A report from the California Secretary of State described problems in San Diego County and Alameda County that prevented polls from opening on time [24]. As San Diego County had no provisional ballots available, some voters were turned away; there was no estimate of how many returned later, and so the number of disenfranchised voters is not known. Other California counties (specifically, Plumas, Kern, and Solano) also reported problems. Orange County also found problems with the systems supplying an incorrect ballot to voters; although a poll worker had to make an error for this to happen, a design problem in the DREs compounded the problem. Finally, because of a change in the law, the systems used in San Diego County miscounted provisional ballots and absentee ballots, requiring that county to recount those ballots and submit an amendment to the Secretary of State's office. Other problems were widely reported in the media.

It is worth viewing these analyses from the perspective of the waterfall model presented in Section 2.1, since this model emphasizes where evidence of assurance (or lack of assurance) has been gathered, and where more work needs to be done. Collectively these reports cover phases 3 and 4 of the waterfall model. The Hopkins report was, in essence, a code review. The SAIC, Compuware and InfoSENTRY reports contained hardware component testing, and a limited code review. These analyses fall into phase 3 of the waterfall model. The RABA, SAIC, Compuware, and InfoSENTRY reports each examined the security of the system as a whole. Notably, SAIC considered previous insecurity claims in the context of the system's procedures; RABA considered the security of the full system in its environment. These analyses all span phase 4 of the waterfall model.

The important observation here is that no previous analysis of voting machines or standards has looked at either of the first two stages of the development life cycle. Our contribution is to begin this analysis. We analyze the first phase, closely examining the requirements promulgated by the standards; we show how judicious formalism facilitates ITA testing to assure that requirements are met. For the second part of the life cycle, we suggest additional requirements for vendor documentation, such as the inclusion of detailed system models and elaborated attack trees, to the same end of improving the testability of proposed election systems.

[1] According to state law, the DRE must be taken to a central location to be repaired before it can be used again. Removing a critical wire is, thus, a very effective denial-of-service attack.

4 Standards

The requirements that the standards articulate are low-level requirements. They speak to specific properties that the systems must have, and not why the systems must meet those properties. In some cases, the relevance is apparent *assuming* the reader has the same threats in mind as the developers of the standards. In other cases, the properties appear to be related to testability.

For example, the requirement that "a maximum acceptable error rate . . . of one in 5000,000 ballot positions" [7, Vol I §3.2.1] is related to the requirement that the votes be counted accurately. But the *exact* relationship of some of the aspects of this property to imagined threats is unclear. Why is the number 5,000,000 and not (say) 500,000 or 100,000,000? To determine this requires knowing the *exact* threat and requirement the standard developers had in mind. The standard does not articulate either.

Other requirements are irrelevant to threats. Consider the coding requirements of the standards, which include that source code "has no line of code exceeding 80 columns in width (including comments and tab expansions)" [7, Vol II §5.4.2 k]. This property does not appear to deal with any threat. It appears to deal with ease of testing, because code with lines of length 80 characters or less will fit on a single line on (almost) all printers and display devices, making the code easier to read. Clearly, this property speaks to code review, an aspect of testing.

As the second example shows, the standards appear to have two goals. The first deals with the goals of electronic voting systems. The second deals with the validation that the electronic voting systems meet the first goal; the standard refers to this aspect as *testing*.

We focus on how the standards address a small sample of issues determined to be problematic by previous investigations: insufficient auditing policies in Section 4.1, insufficient access control policies in Section 4.2, voting system integrity during build and deployment in Section 4.3, and data integrity during transmission in Section 4.4.

A longer critique relevant to the 2002 FEC standards is available as a technical report [12]. Many of these comments are relevant to the 2005 EAC standards, but are omitted from the discussion here for brevity.

4.1 Auditing

The operational requirements for auditing are specified in *System Audit* [7, Vol I §2.2.5]. If a report is produced, the audit then falls subject to *Producing Reports* [7, Vol I §2.5.3], *Data Retention* [7, Vol I §2.2.11], and *Data and Document Retention* [7, Vol I §4.3]. Furthermore, audits are required [7, Vol I §2.2.6 i, §2.2.11, §2.5.3.1 f] to capture "the data indicated in Section 4.5," although these sections probably should reference *Audit Record Data* [7, Vol I §4.4], since *Vote Secrecy* [7, Vol I §4.5] applies only to DRE systems and gives no audit requirements other than to retain the voter's selections for vote counting purposes.

> [7, Vol I §2.2.5.1]: Because the actual implementation of specific characteristics may vary from system to system, it is the responsibility of the vendor to describe each system's characteristics in sufficient detail that the ITA and system users can evaluate the adequacy of the system's audit trail. This description shall be incorporated in the System Operating Manual, which is part of the Technical Data Package (TDP).

The auditing abilities must be described "in sufficient detail" to "evaluate the adequacy of the audit trail pro-

duced." Such a description, however, in no way sounds like material suitable for an operating manual, whose focus is on running (not critiquing) an audit system. As a manifestation of this inadequacy, the SAIC report has already commented on the inadequacy of documentation delineating how to configure the audit software and how to review the audits produced [28, §2.1.5].

The "System Operating Manual," in which an audit capability should be described by the vendor, is referenced only once more [7, Vol II §A.1]. "System Operations Manual," however, is referenced twice [7, Vol II §1.4 i, §2.8.7]. The fact that one reference [7, Vol II §2.8.7] appears as a subarticle of *System Operations Procedures* [7, Vol II §2.8] implies that it is the same thing as the System Operations Procedures documentation. If the "Systems Operations Procedures" documentation, "Systems Operating Manual" and "Systems Operations Manual" are indeed the same, then the audit description requirement should be covered by the following:

> [7, Vol II §2.8.5]: The vendor shall provide documentation of system operating procedures that meets the following requirements: . . . [deletia] . . .
>
> f. Provide administrative procedures and off-line operator duties (if any) if they relate to the initiation or termination of system operations, to the assessment of system status, or to the development of an audit trail.

In short, no section of the standards requires the vendor to document how to configure the audit system, when to invoke that audit system to develop an audit trail, how to protect that audit trail from being compromised, or how to use an audit trail to confirm the correct execution of the system or discover and fix errors. There is only the requirement that the vendor provide documentation showing the system can produce an adequate audit trail and explaining how to produce it. For example, in Miami-Dade County, FL in 2004, officials could produce a sufficient audit trail, but the trail overwhelmed the servers when passed to the central tabulation machines [10, p. 127]. It is not surprising that independent investigations have, among their discoveries, found databases where auditing was not configured properly [25, p. 21] and systems whose audit trail could be viewed and edited by unauthorized personnel in a manner undetectable to future auditors [3, p. 47 §3.4].

4.2 Access Control

Access control is addressed in *Access Control* [7, Vol I §6.2] and *Telecommunications and Data Transmission — Access Control* [7, Vol I §6.5.1]. The functional specifications in *Access Control* require the vendor to recommend policies and describe the mechanisms used to enforce these policies. Specifically, the vendor's recommended policy and mechanisms should "permit authorized access to the system," "prevent unauthorized access," and "provide effective voting system security." Additionally, the vendor must provide a list associating all individuals with the functions and data to which they are granted access, including how and when they are granted access [7, Vol I §6.2.2]. As part of the "System Security Specification" documentation requirements, a vendor must "specify the features and capabilities" of its recommended access control policy and "provide a detailed description" of the access control measures and procedures used to enforce the recommended policy [7, Vol II §2.6.1, §2.6.2, §2.6.5, §2.6.6].

Upon review of these sections, it is unclear how the standards have defined an "Access Control Policy" or "Access Control Measures" — neither technical term appears in the provided glossary. One section [7, Vol II §2.6.1] implies that an Access Control Policy consists of "features" and "capabilities." A different section [7, Vol I §6.2] implies that an Access Control Policy is the system's access control matrix's initial state. Briefly, an access control matrix is a table which associates each individual with a list of rights [2, Ch. 2]. As an analogy, if the right is access to a locked room, the standards require the vendor to create a list of everyone who has a copy

of the key to the lock. But the standard does *not* require that vendors disclose how the access control matrix can be modified, under what conditions it can be modified, and who can modify it. To continue our analogy, the standards do not require the vendor to describe how to handle duplication of keys or changing locks, or who can do those things, or when those things can be done.

The standards also do not require the vendor to train personnel in the administration and continued maintenance of the system's access controls. In fact, system administration training is required only if a network is used [7, Vol II §2.10.2 d]. The SAIC report's objection to the inadequacy of documentation pertaining to the process of maintaining access controls provided with the Diebold system [28, §2.1.9] is symptomatic of the inadequacy of this aspect of the standards.

Further, the standards choose language to avoid constraining the vendor's choice in access control policy and enforcement mechanisms. This potentially allows vendors to use practices that are considered poor. For example, one section [7, Vol I §6.2.1 g] requires vendors to "provide a description of recommended policies for . . . segregation of duties," while it would not have been much harder to require that the vendor's policy adhere to the principle of separation of duty wherever applicable. The incorporation of this best practice in a security policy should not be left to the vendor's judgement.

Independent analyses by researchers have exposed the results of deferring to the vendor's judgement. For instance, various types of mischief are prevented by a lock on the bay on the Accuvote-TS terminal. Each terminal comes with two keys to the lock, and the locks on all terminals are identical [25, p. 18]. So, the access control policy for any precinct implicitly gives access to that lock to anyone with a key, including poll workers in a different polling place or a different state, technicians who work for the manufacturer or the testing authority, and individuals who had access at a tradeshow. As each machine had two keys, and approximately 16,000 machines were purchased by Maryland alone, a large number of individuals may have either retained a key or obtained a copy of a key. Similarly, hard-coded passwords were used to control supervisor functions in systems made by Diebold [25, p. 16] and ES&S [3, p. 96 §1.31]. Again, the implicit access control policy effectively gave supervisor access to anyone who had ever seen these passwords, including officials from past elections, poll workers in other precincts, the manufacturer's programmers, and anyone who has read the passwords online or in print (for example, in [18, §4.4] or [3, p. 57 §1.21(b)]). These policies, implicit in the vendor's design and not constrained by the standards, do not adhere to the best practices of systems for which security is a concern.

4.3 Voting System Integrity

There is much concern that, even with good standards in place, little requires vendors to manufacture the delivered system according to practices and using the materials that have met approval under the standards' review process. The SAIC report expressed these sentiments [28, §2.2.1]. For example, the standard does not require procedural mechanisms to enforce the following requirement:

> [7, Vol I §1.6 b]: The software submitted for certification testing shall be the exact software that will be used in production units.

The specific issue of software integrity *is* addressed in *Software and Firmware Installation* [7, Vol I §6.4.1], *Distribution of Voting System Software* [7, Vol 6.4.3 §,] and *Software Distribution Methodology Requirements* [7, Vol I §6.4.4] . This section, however, does not require any demonstration or proof that the installed software is the same as that approved by the ITA. Similarly, the standards do not comment on maintaining the integrity of

the software or system throughout its lifetime. The standards do require that each ROM be validated before the run of each election [7, Vol I §6.4.1 a] but there is no requirement that the integrity be maintained throughout the election. As a symptom of this deficiency, the RABA Red Team uncovered flaws in the Diebold system which allowed an attacker to potentially change the DRE's software at any point in its lifetime [25, p. 19].

The standards should require the vendor to specify a procedure to verify the integrity of the software and system, at least throughout the manufacturing process. This procedure should be analyzed by the ITA for correctness. Such a procedure might be made part of the *Witness of System Build and Installation* [7, Vol II §1.8.2.4]. Relevant data[2] (i.e. hardware serial numbers, cryptographic checksums, signed certificates) could be provided to election officials, so they can be assured that the system received is the same as the system inspected by the ITA (e.g. the installed software is unmodified since the witnessed build). More generally, there should be a way to verify that the *complete* system from the witnessed build is the same as the system being used, at any point in its life cycle.

Evidence of the fact that the standards require insufficient procedures for assuring system (specifically, software) integrity is apparent from various elections. El Paso County, CO, used uncertified DRE software in a 2002 election [13, p. 192]. In 2004, software on a DRE was modified *during* an election in LaPorte County, IN [22]. Also in 2004, in Marion County, IN, an election was held using uncertified software provided by ES&S. ES&S later replaced this software with certified software [5, 19, 4]; this became known when an ES&S employee blew the whistle on her employer.

4.4 Data Integrity during Transmission

The standard addresses data transmission in *Data Transmissions* [7, Vol I §5.1.3] and *Telecommunications and Data Transmission* [7, Vol I §6.5]. The following subsection tries to ensure vote-data integrity during transmission:

> [7, Vol I §6.5.2]: Voting systems that use electrical or optical transmission of data shall ensure the receipt of valid vote records is verified at the receiving stations.

While this is a necessary condition, it is not sufficient. If *mutual* authentication is not used then it is possible for someone to intercept the transmitted data before it reaches its destination, alter it, and then forward it to the central election management system (a "man-in-the-middle" attack). The system inspected by the RABA Red Team passed certification because the GEMS servers did verify the authenticity of the vote records. The DRE machines, however, did not verify the authenticity of the GEMS server, so the security of the system was compromised by a man-in-the-middle attack using only a laptop computer [25, p. 21].

The sections *Data Interception Prevention* [7, Vol I §6.5.3] and *Data Interception and Disruption* [7, Vol II §6.4.2] may have been relevant in prohibiting a man-in-the-middle attack as described above, since the attacker must intercept the transmitted signal and relay a modified signal to the target. *Data Interception Prevention*, however, only requires use of some encryption standard and appropriate measures to detect intrusive devices/processes. In a previous version of these standards, this section required the use of the Advanced Encryption Standard and appropriate measures to detect physical taps (electromagnetically-coupled pickups, wiretaps) that could leak data to an unauthorized recipient. It is odd that physical taps are only considered

[2] Currently, the standard requires a unique identification number, software version number, software vendor name, etc. to be supplied with the system documentation [7, Vol I §6.4.4.1]. If this information differs from the equipment delivered to the polling center, it is a reason for concern. If this information does not differ, however, the system still may not be the one expected. It is trivial for malicious software to spoof a version number or report any anticipated vendor's name.

in the case of systems that use telecommunications; it is more odd that the current version of the standards do not consider such threats at all. *Data Interception and Disruption* requires that the ITA review and use judgement in deciding the acceptability of the manufacturer's documented solutions to handling new external threats to the system's use of a telecommunications network. Since man-in-the-middle attacks do not require intrusive actions that could be caught by an Intrusion Detection System, do not require physical tapping, and are not a "new threat," this attack is not prevented by any version of these requirements nor does it fall under the responsibility of the ITA's review.

5 Model

A model of the system is an abstract description of what the system does, and how. A system typically has several models, at different levels of detail. For example, a model may be a feature-level model that describes the inputs and outputs of each subsystem, while omitting the details of how those parts work. Implementers can then develop the subsystems as they please, subject to the constraints imposed by the feature-level model. Alternately, a model may describe how each part of the system is to work in detail, although this restricts the ability of the implementers to make choices.

For brevity and clarity, we do not attempt to model the entire election process. Instead we focus on a single aspect of the voting process — ballot data flow. We choose to model this aspect since we can use it to address the shortcomings in the standards' treatment of ballot data transmission, as discussed in Section 4.4. Since models reflect requirements and provide a basis for asserting that a system meets those requirements, we discuss and formalize those requirements relevant to the integrity of ballot data flow.

When a ballot is marked and cast, it can be thought of as data. This model seeks to describe how this data is processed, transformed, stored, and transmitted. Any data that is related to the ballot data can be considered part of this data flow. For instance, in a presidential election, the machine audit data, the tally from an individual DRE machine, the precinct's tally, the announcement of the state's final count, the decision of the electoral college, and the resulting interior design of the oval office are all part of the ballot data flow.

This exercise will demonstrate why a set of standards for electronic voting machines should require a model of the systems. This will enable validators to check that the system meets the specific requirements laid out in the standards as necessary. It also provides a basis for a vendor to assert that the product complies with the standards. The vendor can offer varying levels of assurance evidence, based upon comparing the system, and vendor model, to the reference model.

In this section, we illustrate, at a high level, what including a model entails. Section 6 augments this by showing how a straight-forward threat model can drive the writing of clear, concise, and testable standards that assure the mitigation of those threats. We start by presenting a formal modeling language.

5.1 Data Flow Diagrams

We use the language of Data Flow Diagrams (DFDs) to model ballot data flow. DFDs were introduced in the 1970s as a tool for structured analysis (see, for example, Gane and Sarson [11]). Specifically, we follow the DFD conventions of Howard and LeBlanc [14, Ch. 4], summarized in Figure 1.

A *data store* can be thought of as a hard disk. A *process* is a function that takes a flow as input and produces

(a) Role (b) Process (c) Data Flow (d) Data Store (e) Boundary

Figure 1: DFD Legend

a flow as output. A *boundary* delineates machine and privilege transitions. It may be useful to think of all processes or individuals within a boundary as being mutually trusting of one another (they are assured, from previous procedures when crossing the boundary, to not be malicious), and to think of the boundary as indicating a tamper-proof housing or physically secure room. A *role* is a collection of specific tasks and duties assumed by an individual. Each of us must perform different tasks that carry with them different duties in different contexts. For example, some of us are teachers in one context, students in another, and parents in yet another. In the context of voting, citizens can be voters, poll workers, and observers. Roles can overlap, as with teacher and parent, or be mutually exclusive at the same time, as with voter and poll worker. *Data flows* must start or end at a process. We refer to roles, processes and data stores collectively as *entities*.

5.2 Requirements

We select a subset of the requirements enumerated in [2, 16, 23, 26] that are relevant to the Ballot Data Flow:

Any voting model must

1. Ensure that the voter's intent is accurately captured;
2. Ensure that all authorized voters' ballots are in the final tally;
3. Ensure that there are no unauthorized voters' ballots in the final tally; and
4. Prevent anyone from changing or discarding a ballot, once it is cast.

Clear, testable requirements should be included in the standards to drive testing by the ITA, because the alternative is ad hoc testing that provides little assurance in the security of a "certified" system. The trouble with the above requirements is that they do not lend themselves to testing. For example, consider how to decide if a system has captured the voter's intent. Is the voter's intent what she brings to the polling station in her head, or what she leaves in the form of a ballot? Is the voter's intent the ballot cast to the ballot-box or the one submitted to the audit-box? In general, all of the requirements may make sense from a very high-level interpretation of an election, but testing must also provide assurances that the components that form an election system are individually correct as well. It is unclear how to test whether these informal requirements hold for each of these components since, for instance, "the final tally" has no meaning until the end of an election. One might try to assign meaning at each stage for testing purposes, but without direction these interpretations will differ. Without formalizing what we mean by "intent" and "to cast," what we say about these things will be meaningless, or at best mean different things to different people.

To mitigate the vagueness (and subsequent untestability) of these informal requirements, we formalize them:

1. Every process should be auditable;

2. Every storage maintains the integrity of the data written to it; and

3. The entities on either side of a flow that traverses a boundary must be mutually authenticated.

Above, a process is *auditable* if its actions are logged to an append-only audit-trail with sufficient detail that its actions can be replayed and compared to correct behavior; *data integrity* is maintained whenever data can be transformed without loss into an exact copy of the data as sent from its source; and *authentication* is the binding of an identity to a unique entity. These terms should not be confused with similar-sounding plain language notions: they are technical terms drawn from security and cryptography literature. For an extended discussion of these terms, see Bishop [2].

We stress that our invocation of auditing should not be confused with the types of audits viewed as sufficient in other domains, like accounting. Currently, the standards seem to advocate the "collect evidence that supports a claim" style audit, and not the "prove the impossibility of the claim's falsehood" style audit.

An intuitive example may be useful[3]. A guestbook can be evidence that someone was at a motel, but is insufficient evidence to prove someone was *not* at a motel. A video surveillance system, on the other hand, is involuntary and, if deployed with sufficient coverage, could capture sufficient detail that it could provide more credible evidence of someone's presence or absence.

Figure 2 depicts an audit trail well-suited for our purposes. It uses (x, σ, y, c), where σ is $\mathcal{A}$'s commitment to input x (e.g., a type of digital signature, like a group signature) and c is a certificate which can be used to check that $f(x) = y$ (e.g., the coins used by a randomized function f). In a sense, the current standards only require (x, y) to be in the audit trail; for example, all ballot images and the final tally are kept. This, however, is not sufficient: it is trivial to forge convincing-looking ballot images that, when audited, confirm any desired tally. In general, if f is invertible, then, for any desired output y, once can easily invent the appropriate input x. Thus, the commitment σ of a role to the input is essential.

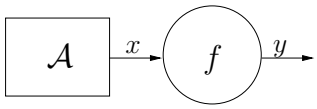


Figure 2: An example process and its inputs in a DFD.

We formalize "casting a ballot" as a process that takes both a flow originating from a "Voter" and a flow representing a ballot, and produces a flow to represent a cast ballot. The flow originating from the voter is the formalization of the "voter's intent", since there is nothing else in our formal model which could be fairly called this. "Once it is cast" is, formally, any flow after the "casting" process. We also assume that the symbols in Figure 1 are used properly. We formalize "authorized voter" as the Voter role. We formalize "unauthorized voter" as the "Adversary," and require that the model does not include the Adversary as a legitimate role.

Under these conditions, the informal requirements are satisfied.

Informal Req. 1 is satisfied. Since "casting" is now an audited process (by Formal Req. 1), the voter's intent (the incoming flow) can be compared with the cast ballot (the outgoing flow) for assurance that the cast ballot captures the intent.

[3] Actually, we have already seen an example of this in the standards. See Footnote 2.

Informal Req. 2 is satisfied. To show that all authorized voters' ballots are part of the final tally, one must prove that none of these ballots have been omitted (shown in satisfying Informal Req. 4, below) and that the tally is correct. The final tally is correct because all processes in the chain culminating in the final tally are auditable (by Formal Req. 1).

Informal Req. 3 is satisfied. Any unauthorized voter (the Adversary) is not a role in the model and, therefore, must masquerade as some other entity to interact the the system. This, however, is not possible because all incoming flows crossing a boundary have their origin authenticated (by Formal Req. 3), and those flows not crossing a boundary, if boundaries are used properly in the model, are assured to not be malicious, and thus could not come from the adversary.

Informal Req. 4 is satisfied. After "casting," a ballot cannot be changed: the flow from the cast process can only pass through storage (whose integrity is assured, by Formal Req. 2) or processes (which are auditable, by Formal Req. 1). After "casting," a ballot cannot be discarded: processes can be audited to see if they have omitted an input, processes used to access storage can be audited to see if data was left unretrieved, and data sent over a boundary is not misdirected, and potentially unreported, because of destination authentication (by Formal Req. 3).

5.3 Example Model: Ballot Data Flow

To limit the scope of the model, we describe the model's *use scenario* — those situations to which the model applies. Here, we concern ourselves with ballot data flow that starts with the casting of a voter's ballot during an election in a polling station, and ends with the publishing of the precinct vote tally. Our model does not describe ballot data flow during testing, during system development, or after the precinct tally is published.

A model for ballot data flow does *not* include those processes that are independent from the data in a ballot. For example, mechanisms that prevent double voting and ensure the voter's anonymity, although important aspects of an election, do not change the ballot data flow and are better modeled elsewhere. In Section 5.4, we comment on other aspects of an election system that might be advantageous to model.

Figure 3 models the ballot data flow of a fictional, but reasonable, DRE election system. The modeled election system represents a set of DRE stations and a single central tabulation machine located in the precinct. Details of many mechanisms and processes have been abstracted away. For example, the process by which a machine loads a ballot and interacts with a voter to produce a completed ballot is described abstractly as "2.0 Fill Out Ballot." A more detailed description of this process, which might, for instance, involve an authorization token like a smart card, could be described in a Level 2 diagram.

The standards should require vendors to produce detailed models, such a detailed version of Figure 3, that reflect their actual system. We suggest the ITA assume the responsibility of gauging the how well the vendor's model represents its actual documented procedures and devices.

Although we will see much of the payoff for this modeling work in the next section, it has already, in conjunction with the requirements, brought into focus what processes and data flows the ITA needs to examine to to assure society that a certified election system is sound. In particular, it makes clear what the auditing process entails and, in so doing, addresses the current deficiency in the standard discussed in Section 4.1.

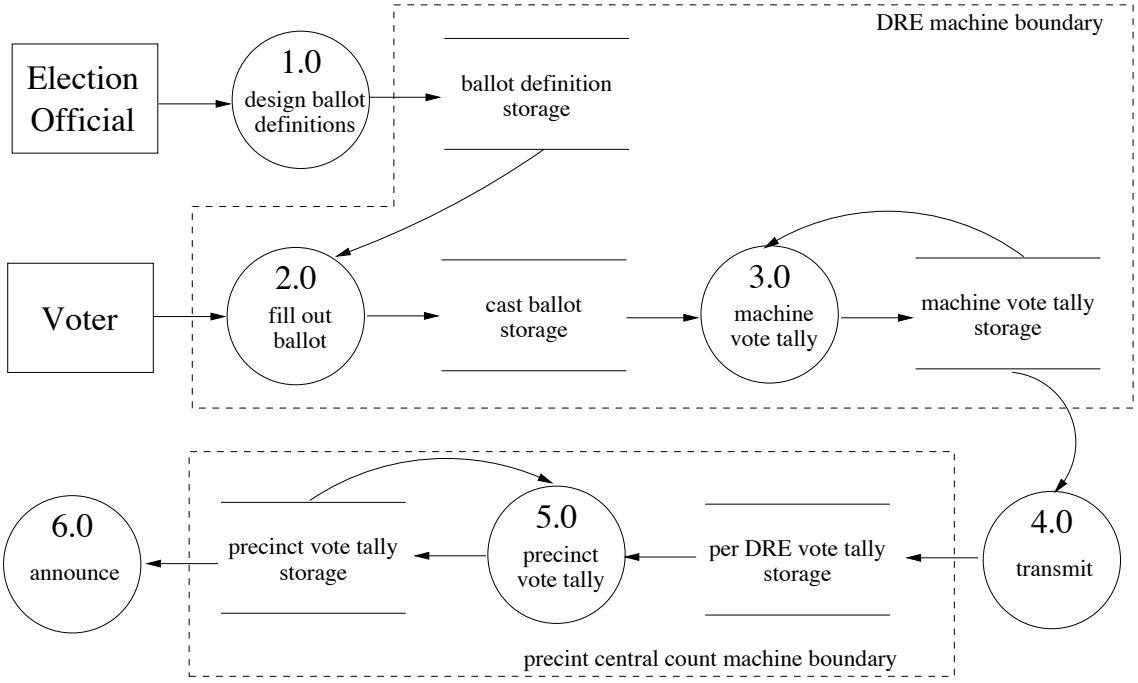


Figure 3: Level 0 DFD for a model of Ballot Data Flow in a DRE system

5.4 Summary

Other informal requirements for correct, secure voting systems are relevant to our example model. For example, the informal requirements that the anonymity of the voter should never be compromised and that election data should not be disclosed prematurely are both relevant to the ballot data flow model. Like ballot data flow, these can be formalized, e.g. in terms of destination authentication, requirements for boundaries (to make sure one cannot eavesdrop data stores or processes), requirements on data included in the flow (to make sure a ballot is not correlated with, say, a voter ID or a time stamp which could then be used to identify a voter using other records compiled either by the voting system or by an adversary).

Drafting formal requirements can benefit aspects of election systems beyond ballot data flow. Using the language of DFDs again, one can model the "flow" of the physical voting system as it moves between witnessed build, deployment, use in an election, storage, modification and re-certification. Section 4.3 comments on problems in the current standards' ability to convince election administration that the machine received is the same system that was certified. By formalizing requirements for a "Physical System Flow" model in a similar fashion to the "Ballot Data Flow" model, one would automatically gain a clear procedure for auditing the physical system to assure an election official of the delivered system's integrity.

Other simple modeling languages can be used to describe those parts of an election system that are fundamentally different from data flow. For instance, processes that require multiple entities to interact could be described using the language of sequence diagrams. One could use this language to create formal standards governing, say, the process of casting a ballot, physically transferring a document, or transferring data over a telecommunications network. The language of control flow graphs could describe those processes that, depending on the input, require one of several possible actions. The administrative process to open and close the polls and to handle potential problems during an election could be described with this language. Providing a base model for

the vendor to elaborate would assure minimal functional requirements to be met (e.g., the vendor *will* provide a procedure for what to do in the case that a machine breaks, since this contingency is in the base model provided by the standards). Using the language of the model to create formal requirements would lend assurance that these procedures meet some minimal standards (e.g., a machine is never serviced by unauthenticated repairmen). A clear standardized exposition of such procedures would also be invaluable in training administration officials. For more on modeling languages useful in describing secure systems, see Jürgens [17].

6 Threat Model

The goal of a model is to provide a system description. Given this description, one can begin to enumerate threats against the system. A *threat* to the system is a possible goal of an attacker, often called the *adversary*. Threats never go away. A threat that the adversary can realize is called a *vulnerability*. Ideally, a system's model should have no vulnerabilities, i.e. no unmitigated threats.

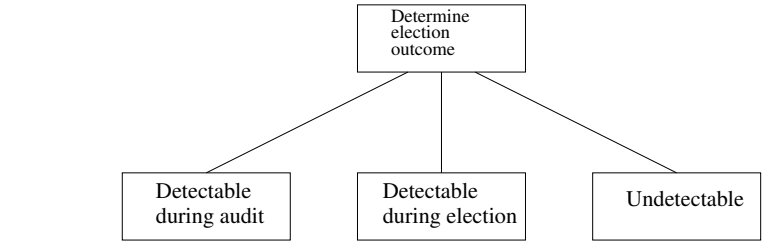
In practice, some threats simply cannot be prevented. For these, the goal is to minimize their impact. Perhaps the best-known example of a threat that cannot be defended against is communication on a multi-user system: one cannot prevent communication via *covert channels*. A covert channel is a shared resource with an attribute that can be used for communication but was not designed for that use. For example, one can use the CPU quantum to communicate in binary, releasing control of the CPU early to send a "1" and holding it to send a "0". A user waiting to use the CPU can observe this phenomena and, thus, receive messages from another user. The remediation is to design the system so that processes using the channel take a very long time to send a message.

In threat modeling (see Sellers [29] or Swiderski and Snyder [31]), threats are identified by analyzing a system's *assets* and *external dependencies*, and imagining ways to compromise these, using the system's *entry points*. For organization and management purposes, threats can be classified using conventions such as STRIDE [31]. Vulnerabilities, in turn, can be further analyzed, using such techniques as exploit cost-analysis [27] or DREAD ratings [14]. A tool used to explore threats and determine which constitute vulnerabilities is the *attack tree*.

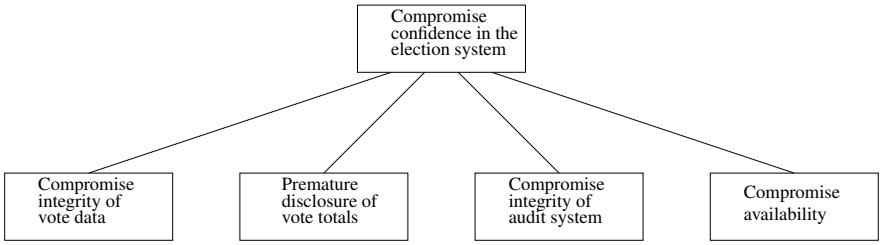
6.1 Attack Tree

Attack trees graphically depict threats. Each node in an attack tree is a threat, a goal of the adversary, and its children, if it has any, are ways to achieve that goal [14, 27]. Classically, attack tree analysis proceeds by expanding the attack tree using a detailed model of the system: thinking in terms of the model's assets and entry points, one brainstorms ways to achieve each node's goal. Each new way adds a child to that node. One *can* begin the process of constructing threat trees without knowledge of a detailed model, using abstract assets. Figure 4 shows the beginning of two such attack trees. In these trees, each root represents the compromise of an asset of the system. In Figure 4(a), the root represents the system's ability to fairly determine an election's outcome; in Figure 4(b), the root is the public's confidence in the election system. In each tree, the leaves represent a different way to achieve their parent's goal.

This process is limited by the imagination of the analyst and assumptions about the difficulty that the adversary faces achieving some threat. For example, while elaborating a tree, if one adds a node that represents the goal of factoring large composite numbers (assumed to be hard under certain conditions) then considering this node's children is unlikely to be fruitful. It is more useful to annotate the nodes of the attack-tree, bottom-up, with



(a) Determine Election Outcome



(b) Compromise Confidence in the Election System

Figure 4: Attack Trees Based on Known Election Assets

properties such as the cost of the exploit or some countermeasure. Such annotations can be denoted by circles on the tree's open nodes. Paths from a leaf to root that have no countermeasures are unmitigated threats and represent vulnerabilities.

Only vendors have sufficient information — a detailed model and a working system — to perform a good attack tree analysis. The standards should force the vendor to do so: they should require vendors to produce elaborated attack trees that contain no unmitigated paths since each path is either annotated with the countermeasures or prohibitively expensive. The standards can and should guide the vendor's analysis by providing partially expanded attack trees that contain known threats to election systems. The standards might require vendors to use these partial trees as the starting point of their own vulnerability analysis. If the standards required these things, vendors would be forced to devote resources to a vital phase of the secure systems design process, and the public would be assured that a minimal set of threats (i.e., those in the standards' partial trees) have been considered by the vendor. The models, trees, and sufficiency of the threats' mitigations can all be inspected by the ITA.

The current standards seem to have certain threats in mind, but no language to describe them. The lack of a generic description of a voting system (i.e. a model) prohibits the standards from describing the threats in a meaningful way. The standards could describe these threats in a generic way and place the burden on the vendor (to make these meaningful to their models) and on the ITA (to check the trees and models). Instead, the standards require defenses against a specific attack (for an unnamed threat) set in an implicit model, and provides no infrastructure to inspect if other attacks could result in the same goal.

For example, the common requirements for system integrity require the system to protect itself against power interruption, electromagnetic disturbances, and fluctuations in temperature and humidity [7, Vol I §2.2.4.1b–d]; similarly, the common requirements for ballot casting require that a power failure or a loss of telecommunications service do not disrupt the casting or tally of the currently active ballot [7, Vol I §2.4.3.1e–f]. The standards' developers obviously had in mind specific types of denial of service (DoS) exploits they want to defend against.

The lack of a model impedes describing these threats in a specific way. The lack of an infrastructure to inspect generic threats impedes describing these threats generally. The compromise, however, appears to be to describe specific attacks (e.g., a power failure that causes the current vote to be dropped) using an implicit model (i.e., the machine is externally powered) while ignoring the more general exploit (i.e., interruption of an election).

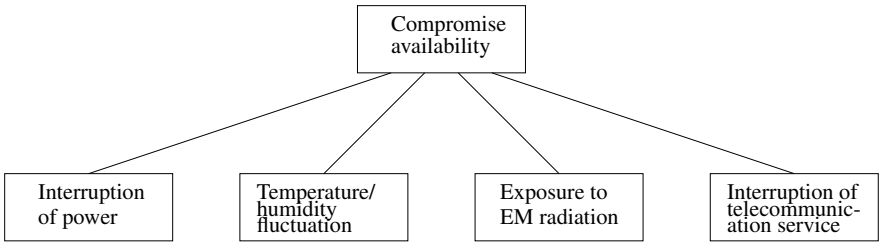


Figure 5: Attack Trees for Some DoS Exploits

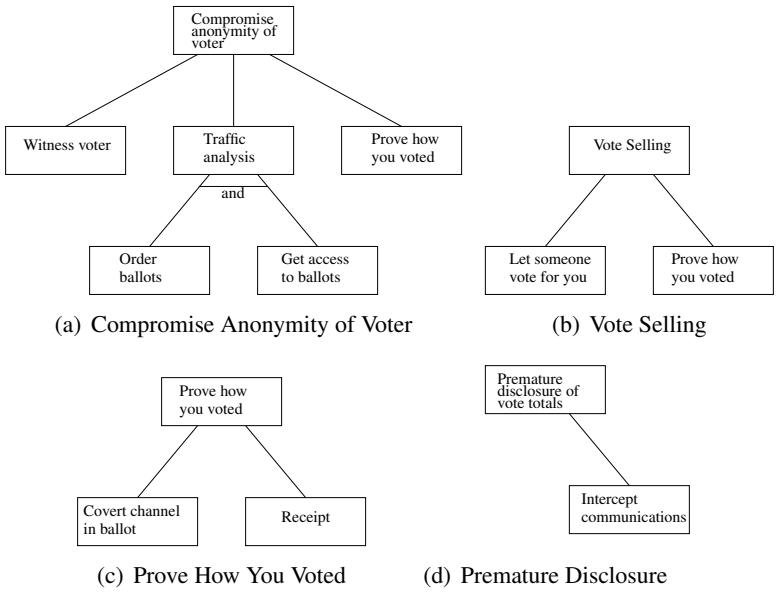
Figure 5 is, more or less, a graphical representation of the common requirements [7, Vol I §2.2.4.1b–d], making explicit the greater DoS threat. Using the above procedure — providing this partial tree, requiring modeling, requiring annotations, inspection of all these by the ITA— would force the vendor to create countermeasures for a host of threats. Using the current procedure, attack paths that are relevant to systems unimagined by the standards, and thus not explicitly mentioned, are never inspected by the ITA.

For instance, consider a system for which a power failure does not disrupt casting and other voting capabilities, but does prevent authentication across the network and halts the election. The current standards do not address such a situation. It may be tempting to revise the standards by adding an extra requirement like 'a power loss should not halt a service that is necessary for the election, even if this service has nothing to do with casting a ballot,' but the approach of incrementally patching the standards with new required features is obviously clumsy and inadequate, because unanticipated attacks will be countered only *after* a problem. Using Figure 5, however, the vendor could be required to divulge which subsystems require external power (part of modeling is listing the subsystems' external dependencies) and to annotate the tree with countermeasures relevant to each subsystem. In this way, the vendor creates countermeasures for a host of attacks that are relevant to their specific model, and not just those imagined by the standards. At the same time, this procedure assures that a minimal number of attacks are considered, i.e., those in the partial trees.

Figure 6 makes explicit some well-known threats to election systems. Some are alluded to vaguely in the standards (e.g., premature disclosure is alluded to when a system uses telecommunications devices [7, Vol I §2.2.10]), some may be the implicit motivations behind some functional requirements (that audio-output must be audible only to the voter [7, Vol I §2.2.7.4.1.2] is motivated by the threat to voter anonymity), and some are dealt with explicitly, although sometimes in odd places — a paragraph dealing with privacy, and the statement that the voter must be unable to prove how she voted, is included under an unnamed subsection [7, Vol I §2.2.7.4] of *Human Factors*. Figure 6 also demonstrates tree-reuse: Figure 6(c) is a subtree of both Figure 6(a) and Figure 6(b).

6.2 Analysis

Figure 7 is an attack tree relevant to ballot data flow. The tree has been annotated to reference discussions relevant to each attack path. Throughout this discussion, we use the ballot data flow model in Figure 3 of Section 5.3, which meets the formal requirements from Section 5.2. We interpret "legitimate ballot" as a ballot



(a) Compromise Anonymity of Voter (b) Vote Selling

(c) Prove How You Voted (d) Premature Disclosure

Figure 6: Attack Trees Representing Some Known Election Threats

cast by the Voter and unchanged since casting. "Ballot" ceases to have meaning at process "machine vote tally," thus ballots exist on the flows entering process 2.0, entering processes 3.0, and all points between. Similarly, "tally" has meaning on the flows exiting process 3.0, entering process 6.0, and all points between.

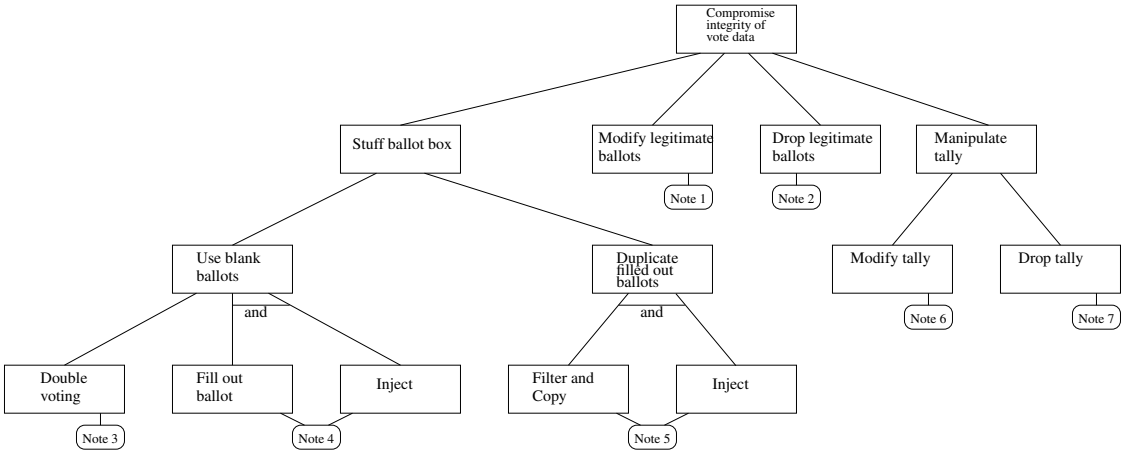


Figure 7: Compromise Integrity of Vote Data

Below, we use "outsider" to denote an adversary who cannot breach the boundaries and is restricted by entry points; we use "insider" to denote an adversary unconstrained by the boundaries; for instance, when considering a machine boundary, such an adversary might be thought of as a malicious program.

Note 1 Considering outsiders, only two entry points are available: the Voter and Election Official roles (see *Outsider Note*). For an outsider to modify legitimate ballots, this means the flow entering process 2.0

and/or the flow entering 1.0 must be modified. The formal requirements, however, guarantee that modifications on either of these flows will be caught during an audit. This immediately suggests that the ITA can test the auditing facility by inputting false or undesirable ballot definitions, etc and checking to see how immediately these problems can be caught. See *Auditing Note* below.

Considering insiders, another test is immediately clear: use the auditing facility to confirm the correctness of process 2.0 and the integrity of the cast ballot storage. This is very similar to the Readiness (Logic and Accuracy) Test currently employed [7, Vol I §2.3.4]. See *Auditing Note* below.

Note 2 This threat does not differ substantively from that considered in *Note 1*.

Note 3 Voting authorization is not part of the ballot data flow model. A control flow graph of procedures during an election that shows, among other things, voter authorization would be a starting point for analyzing this threat, as suggested in Section 5.4.

Note 4 Attacks that realize this threat require the adversary to fill out a ballot — i.e., access the ballot definition storage, or eavesdrop on the flows of process 1.0, or find out the ballot format by using public information or collude with an insider and inject the ballot into a flow that uses ballots — the flows exiting process 2.0, those entering and exiting the cast ballot storage, and those entering process 3.0[4]. The injection portion of the attack is inaccessible to outsiders, as there are no entry points that allow an outsider to manipulate these flows (see *Outsider Note*).

Considering insiders, modifying the flows interacting with any process or storage outputting, using, or holding ballots will be caught by the auditing facility. See *Auditing Note* below.

Note 5 This threat is similar to that considered by *Note 4*, with the observation that the adversary must eavesdrop on cast ballots to filter and copy them before performing the injection.

Note 6 Outsiders can only modify tallies by manipulating process 4.0 (by attacking the protocol used) or the flows entering and exiting process 4.0 (by modifying the signal sent between the parties, etc.), as all other relevant flows are inside boundaries (see *Outsider Note*). This implies the following tests: check the authentication protocol and its low-level implementation making sure it can withstand signal manipulations (using, say, checksums) and protocol attacks (using a widely-trusted protocol is good assurance against protocol attacks).

Considering insiders, modifying the flows interacting with any process or storage manipulating or holding tallies will be caught by the auditing facility. See *Auditing Note* below.

Note 7 For the reasons given in *Note 6*, outsiders can only drop tallies by interrupting process 4.0. This might be accomplished by making sure process 4.0 cannot occur (a DoS exploit) or by causing the process to occur with the wrong entity, so the ballot counts never reach their final destination and are dropped. The latter attack is made difficult by the mutual authentication requirement. Thus, an outsider must attack the authentication mechanism to perform this exploit. This implies the following tests: check the authentication protocol and its low-level implementation; check the mechanism used to detect a DoS exploit and test the back-up procedure for process 4.0, by simulating a DoS attack on process 4.0.

Considering insiders, the remarks from *Note 6* apply.

[4]Here we do not consider filling out a ballot by invoking process 2.0, as this involves masquerading as the Voter and is better described with the "double voting" attack addressed in *Note 3*.

Outsider Note One should be sure that outsiders are restricted by entry points and boundaries. That is, this assumption should be checked by making sure boundaries are secure and no extra entry points exist. This implies the following tests: test for alternate entry points using penetration testing and test the sufficiency of the boundary security (e.g., physical security procedures, tamper-prevention mechanisms, access control mechanisms). For instance, tamper-detecting tape does not prevent an outsider from crossing a boundary but provides evidence that this might have happened, forming an audit-trail prone to false positives. This does not keep outsiders from acting as insiders, but causes boundary-compromises to be audible. Thus, testing tamper-detecting tape does not tell us that our assumptions about outsiders will be good, but does tell us when our assumptions have potentially failed. See *Auditing Note*.

Auditing Note When auditing is used to detect an attack, we do not consider these attacks to be prevented in any way. Notice the goal of the tree in Figure 7 is a subgoal of Figure 4(b). If the audit trail determines that the integrity of the ballot data flow has been compromised, this may be a useful exploit. Furthermore, the integrity of the ballot data flow can be compromised in a fashion that is not caught by an audit if the auditing software is malicious or can be compromised. This is possible when considering insiders, outsiders colluding with insiders, outsiders who can modify the audit-trail, etc. Compromising the integrity of vote data, then, is still possible with auditing. In fact, it can be done in noisily, by letting auditing catch the loss of integrity, or quietly, by modifying the audit trail. The latter implies the following tests: considering outsiders, check the auditing facility's configuration and access control policies, in light of best-practices and using penetration testing; considering insiders, employ careful code review.

6.3 Summary

In the absence of a threat model, the ITA has no other option but to perform ad-hoc testing which produces no real standard, minimal assurances. In the absence of a system description (i.e., a model), threat modeling will at least describe the abstract threats the standards wish to communicate but, again, will result in ad-hoc testing against these threats. Combined, a threat model and a system model immediately yield tests to assure against pre-determined attack paths: given an attack, exhaustively list the relevant parts of the model and the relevant assumptions of the model; given this list, it becomes apparent which parts of the model should be tested and which countermeasures should be tested. With the knowledge of a goal (i.e., threat), the sufficiency of a countermeasure can be ascertained; a list of required countermeasures unbacked by threats is meaningless. For example, standards currently require tamper-evident seals [7, Vol I §6.4.6.3.4, §6.8.7.2.7]. Divorced from threat modeling, this requirement might seem a reasonable countermeasure. When considered in light of the goal of visibly compromising the integrity of the system, it is not. Threat modeling makes this obvious. Adding and checking new attack paths becomes a structured process. Documentation explaining testing becomes organized and, thus, more easily reviewable. It is even possible to check that important design principles have been observed by the vendor. We claim, for example, that defense-in-depth exists when an attack path includes a node with multiple countermeasures and a single-point-of-failure exists when multiple nodes employ the same countermeasure. Formalized using an attack tree, checking to be sure defense-in-depth is employed and there are no single-points-of-failure becomes straight-forward; whereas, currently, there is no clear way to enforce these principles.

7 Conclusion

The current electronic voting standards emphasize those functional capabilities that a voting system should provide. The certification process, accordingly, emphasizes checking that these functional capabilities are available and appear to operate successfully. However, as we have shown, the voting standards provide

1. no description of the election (models);
2. no description of threats or attacks, despite the fact that known threats and attacks motivate many of the functional requirements, e.g. speaker volume.
3. no minimal design criteria (e.g. the vendor is free to implement access control however they see fit)
4. no infrastructure for reviewing the vendor's design choices, despite providing one for reviewing implementation choices, i.e. ITA code review; and
5. no description of how testing demonstrates that an election system has satisfied the standards.

As a result, attackers have found security flaws in certified systems and failures on Election Day have demonstrated that the systems, although certified, do not meet the needs of the jurisdictions using them. The answer to the question we posed in the introduction — whether these problems were caused by inadequate standards or insufficient testing — is both.

Code review alone will not address this problem. When the auditing policies for the system's life-cycle are inadequate, an adversary could easily install different software on the system before its deployment in an election; when access control policies and other design choices are never reviewed critically, an adversary could authenticate as a trusted, privileged user and act maliciously during an election.

We recommend that the standards

- require the vendor provide models for the system proposed;
- articulate threats by beginning the task of formalizing them using attack trees; and
- provide minimal formal requirements related to threats

and that the ITA

- ensures the models provided by the vendor are reasonable and correspond to the vendor's implementation;
- inspects critically whether requirements have been met and countermeasures are sufficient; and
- interrogates the model with model-specific attacks to known threats and with attacks to other reasonable threats which have not been accounted for in the standards.

Enacting these recommendations will make the standards clearer, more easily tested and extended. In other words, they would integrate a verifiable secure design process into the standards' certification process.

These recommendations increase the responsibilities of the ITA. Informally, 'the ITA should be sure the vendor's access control policy is good' runs the risk of being as unaccountable as 'the vendor is on its honor to provide a good access control policy.' Currently, ITAs are not required to publish the results of their testing. For the certification process to provide any assurance to the states (as purchasers of the system) and to the public (as users of the system), this must change. Also, the composition of the ITA is critical and has not been addressed. If the tester is not also the end user of a system, a perverse incentive may arise: vendors may shop around for sympathetic testers who do not rigorously test their systems [1]. The ITA must be independent of the standards writers and especially vendors, but not end-users. Thus, we recommend that the ITA include election officials on loan from their home counties for a period of time. The ITA's task is so important that we believe such tours of duty are warranted.

References

[1] R. Anderson. Why information security is hard-an economic perspective. In *ACSAC '01: Proceedings of the 17th Annual Computer Security Applications Conference*, page 358, Washington, DC, USA, 2001. IEEE Computer Society.

[2] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley, 2003.

[3] Compuware Corporation. Direct Recording Electronic (DRE) Technical Security Assessment Report, November 2003. `http://www.sos.state.oh.us/sos/hava/compuware112103.pdf`.

[4] Rick Dawson and Loni Smith McKown. Marion county election board demands answers from ES&S. WISH-TV Indianapolis, Indiana, April 22 2004. `http://www.wishtv.com/Global/story.asp?S=1808590`.

[5] Rick Dawson and Loni Smith McKown. Voting machine company takes heat over illegal software. WISH-TV Indianapolis, Indiana, March 11 2004. `http://www.wishtv.com/Global/story.asp?S=1704709`.

[6] Diebold Election Systems. Summary technical analysis of recent voting system report, July 29 2003.

[7] Election Assistance Commission. Voluntary voting system guidelines, 2005.

[8] Federal Election Commission. Performance and test standards for punchcard, marksense, and direct recording electronic voting systems, January 1990.

[9] Federal Election Commission. Voting systems performance and test standards, 2002.

[10] John H. Fund. *Stealing Elections*. Encounter Books, 2004.

[11] Chris Gane and Trish Sarson. *Structured Systems Analysis: tools and techniques*. 1986.

[12] Mark Gondree, Patrick Wheeler, and Dimitri DeFigueiredo. A critique of the 2002 FEC VSPT e-voting standards. Technical Report CSE-2005-20, UC Davis, 2005. `http://www.cs.ucdavis.edu/research/tech-reports/`.

[13] Bev Harris. *Black Box Voting*. Plan Nine Publishing, 2004.

[14] Michael Howard and David LeBlanc. *Writing Secure Code*. Microsoft Press, 2nd edition, 2003.

[15] InfoSENTRY. Volume 1 computerized voting systems security assessment: Summary of findings and recommendations, November 2003. http://www.sos.state.oh.us/sos/hava/infoSentry112103.pdf.

[16] D. Jefferson, 2004. private communication.

[17] Jan Jürgens. *Secure Systems Development with UML*. Springer-Verlag, 2005.

[18] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting system. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 27–40. IEEE Computer Society, May 2004. Appeared previously as Johns Hopkins University Information Security Institute Technical Report TR-2003-19, July 23, 2003.

[19] Mary McDermott and Loni Smith McKown. Marion county clerk accuses ES&S of lying. WISH-TV Indianapolis, Indiana, April 20 2004. http://www.wishtv.com/Global/story.asp?S=1799902.

[20] Rebecca Mercuri. Inside risks: Voting automation (early and often?). *Communications of the ACM*, 43(11):176, November 2000.

[21] Rebecca Mercuri. The FEC proposed voting systems standard update: A detailed comment by dr. rebecca mercuri, September 2001. available at http://www.notablesoftware.com/Papers/FECRM.html.

[22] Kristin Miller. Computer glitch still baffles county clerk. Michigan City News Dispatch Online, November 2004. http://www.michigancityin.com/articles/2004/11/04/news/news02.txt.

[23] Peter Neumann. Security criteria for electronic voting. In *Proceedings of the 16th National Computer Security Conference*, 1993.

[24] Office of the California Secretary of State. Report on march 2, 2004 statewide primary election, 2004.

[25] RABA Innovative Solution Cell. Trusted Agent Report: Diebold AccuVote-TS Voting System, January 2004. http://www.raba.com/press/TA_Report_AccuVote.pdf.

[26] Roy G. Saltman. Accuracy, integrity, and security in computerized vote-tallying, August 1988.

[27] Bruce Schneier. Attack trees: Modeling security threats. *Dr. Dobb's Journal*, December 1999.

[28] Science Applications International Corporation. Risk Assessment Report: Diebold AccuVote-TS Voting System and Processes, September 2003. http://www.dbm.maryland.gov/SBE.

[29] Dan Sellers. Threat modeling. PPDM Calgary Data Management Conference presentation, November 2004. available at http://www.ppdm.org/events/past_events/fall_conference_2004.html.

[30] Star Report. Computer glitch was a puzzler. Indianapolis Star, November 6 2003.

[31] Frank Swiderski and Window Snyder. *Threat Modeling*. Microsoft Press, 2004.

[32] P.L. Vora, B. Adida, R. Bucholz, D. Chaum, D.L. Dill, D. Jefferson, D.W. Jones, W. Lattin, A.D. Rubin, M.I. Shamos, and M. Yung. Evaluation of voting systems. *Communications of the ACM*, 47(11):144, November 2004.