USENIX Journal of Election Technology and Systems (JETS)
Volume 2, Number 2 • April 2014
www.usenix.org/jets/issues/0202

A Systematic Approach to Analyzing Voting Terminal Event Logs*

Laurent D. Michel Alexander A. Shvartsman Nikolaj Volgushev

Center for Voting Technology Research and Computer Science & Engineering Department
University of Connecticut, Unit 4155, 371 Fairfield Way, Storrs, CT 06268, USA.

Abstract

This paper presents a systematic approach to automating the analysis of event logs recorded by the electronic voting tabulators in the course of an election. An attribute context-free grammar is used to specify the language of the event logs, and to distinguish compliant event logs (those that adhere to the defined proper conduct of an election) and non-compliant logs (those that deviate from the expected sequence of events). The attributes provide additional means for semantic analysis of the event logs by enforcing constraints on the timing of events and repetitions of events. The system is implemented with the help of commodity tools for lexical analysis and parsing of the logs. The system was rigorously tested against several thousand event logs collected in real elections in the State of Connecticut. The approach based on an attribute grammar proved to be superior to a previous approach that used state machine specifications. The new system is substantially easier to refine and maintain due to the very intuitive top-down specification. An unexpected benefit is the discovery of revealing and previously unknown deficiencies and defects in the event log recording systems of a widely used optical scan tabulator.

1 Introduction

Auditability of electronic voting equipment used in elections emerged in recent years as an important requirement in ensuring the integrity of the electoral process. Election audits may encompass a wide set of activities ranging from the physical or electronic examination of the voting equipment to independent retabulation of election results in the jurisdictions that use paper ballots. While no single auditing activity may be able to answer all questions regarding the proper conduct of an election, it is desirable to have a portfolio of tools and methods for auditing various aspects of an election. Among the important questions that should be posed in any audit is whether a particular voting terminal was used in accordance with established procedures and whether its behavior deviated from the expected. To answer these questions it is instrumental to analyze the event logs generated by most electronic voting terminals.

An earlier work [1] pursued a similar goal for optical scan voting terminals, resulting in a tool that was used in several elections in the State of Connecticut. An interesting

*Research funded in part by the Office of the Secretary of the State of Connecticut and a grant from the U.S. Election Assistance Commission.

by-product of that development was the identification of several defects and deficiencies in the event logging system embedded in the voting terminal. The tool was based on a state transition diagram representation of the way that the voting terminal generates and records events. The state diagram was enriched by rules that captured the expected timed sequences of events that would be consistent with the proper conduct of an election. The system was successfully used to analyze event logs collected from voting terminals used in real elections. This led to several findings surrounding the electoral processes in which actual behaviors did not fully adhere to the prescribed rules and procedures, and incidentally this helped improve the training of election workers. (We note that no deviations from the expected processes could be attributed to malicious actions or severe errors in the voting terminal implementation.)

Yet, the tool [1] has its Achilles' heel as it proved difficult to extend and maintain. Over time, the state transition diagram became very complex due to the sequence of extensions needed to model various aspects of the electoral process, and to identify commonly occurring deviations from the process. This complexity made it difficult to revise the implementation as the necessarily low-level modeling of events and state transitions obscured the high-level model of the election process, resulting in a system where localized changes caused unexpected and broad side-effects. Finally, a clear and meaningful reporting of the analysis results was difficult precisely because of the obfuscation of the overall electoral process. The approach outlined in this paper is offered as a solution to these difficulties and is focused on flexibility, extensibility, and maintainability.

The paper focuses on a specific voting terminal: the Premier's Accu-Vote Optical Scan (AV-OS) terminal that is used in a large number of counties nationwide. Specifically, we present a part of the post-election audit process that deals with the analysis of the event logs. Nonetheless, it is worth pointing out that the results directly generalize to any other voting terminals that produce event logs as well as variations around the electoral process. The event log of the voting terminal is stored in a removable memory card [7]. While the voting terminal hardware may be identical across many precincts deploying AV-OS systems, the contents of the memory cards differ depending on the specific elections in individual precincts. The event log on a memory card contains a history of the actions performed by the voting terminal(s) using the card since it was programmed for the specific precinct.

Event logs are analyzed to determine whether the prescribed election procedure was followed by the poll workers and whether the tabulator operated as expected. (The election procedures for the State of Connecticut are found in [14, 12].) Since it is impossible to verify the actual proceedings, we analyze their traces as recorded in the event logs. A precondition for an accurate log analysis is that the event log is a true, complete, and unambiguous reflection of the events that have occurred during the election process. In the case of the AV-OS event logging system, this is unfortunately not achievable as the event logs may be incomplete or ambiguous. For instance, when several totals reports are printed consecutively, only the first one is logged. This logging deficiency, and others, are discussed in [1]. Additionally, there are several defects in the event logging system that further reduce the accuracy of the logs. Accordingly, we distinguish the probable causes of a given event log deviation. The system detects two different types of deviations: those caused by system malfunction and those resulting from operator errors, e.g., a failure of the poll workers to follow the prescribed procedure.

Note that the deficiency of the AV-OS logging module makes it possible for an operator to conceal invalid behaviors as system malfunctions thereby reducing the usefulness of the logs. Detecting such involved manipulation goes beyond the scope of this paper.

Our Contributions. The contributions presented in this paper are as follows.

(1) We specify an event log language in terms of an annotated context-free grammar. The language models the event sequences that are compliant with the proper electoral procedures as well as deviant sequences.

(2) We present a multi-layered compliance analysis that enhances the reporting of deviations. By defining different levels of non-compliance we can classify event sequences with more precision. Rather than simply reporting that a deviation has been observed, we report the class and severity of the deviation.

(3) We use a familiar and accessible modeling medium—context-free grammars—to describe and implement the forensics tool. Context-free grammars are ubiquitous, transparent, easily modifiable and completely extensible, making it a convenient tool for advanced forensics in the hands of professionals. We follow a standard implementation design based on `Bison` [5], an established parser generator tool.

(4) We implement a detailed notification system recognizing a wider variety of deviations, and capable of issuing concise, appropriate, ranked notifications, thus reducing the required amount of manual analysis. These notifications are understandable by anyone with a knowledge of the election process.

(5) The proposed analysis tool features multiple layers of usage. While refining the underlying grammar requires a technical background (context-free grammars and `Bison`), using the diagnostic component does not. In particular, one can imagine a Political Science researcher using our tool to data-mine event logs of past elections. We also note that the meaning of the grammar can be understood by an election official with just a brief introduction by a specialist.

(6) An inventory of previously unknown deficiencies and defects in the AV-OS logging system surfaced as a result of the systematic formalization of the compliant and non-compliant event traces. In particular we observe *a*) several deficiencies in the logging of specific events, *b*) a defect that results in a failure to clear the election counters, and *c*) a defect that creates an ambiguity as whether the first cast ballot was counted.

(7) We performed an analysis on a large archive of log files (over 1,000 files). We used the system to analyze the same collection of log files as in [1]; our new analysis compares favorably with respect to [1].

We emphasize the importance of including the event log analysis as a part of a post-election audit of technology [2]. A careful examination of timestamped events reveals information about the procedures followed during an election process, including the information suggesting improper conduct of an election or malfunction of voting terminals.

Lastly, we observe that while the presented tool is geared specifically towards AV-OS, our methodology emphasizes modifiability. We chose a context-free grammar representation, as opposed to regular expressions or grammars, or a finite state machine model, because its superior expressive power enables us to maintain the desired level of modifiability at a sufficiently high level of abstraction.

It is perhaps valuable to point out that, at least in the case of the AV-OS, the language of traces could be and indeed was modeled with a (counting) finite-state machine. Yet, as the set of deviant behaviors grows, the finite-state machine specification quickly becomes large and cumbersome which significantly hampers the ability to maintain it. Trying to model the language by means of a regular grammar is also problematic: the (left or right) linearity of regular grammars do not allow one to model the language in a hierarchical

top-down fashion, and to model deviant behaviors requires the introduction of numerous "transitional" non-terminals that obfuscate the structure of logs. On the contrary, a context-free grammar specification provides a natural top-down definition that is understandable to non-specialists (upon a brief introduction), and is robust in the sense that it elegantly adapts to new deviant traces and remains highly maintainable.

The election processes are fundamentally similar across states; we anticipate that the grammar can be readily modified and/or extended to capture the procedural specifics of a given state election and a given specific voting equipment, provided that the equipment incorporates a temporal event logging system.

Disclaimer. In this work, the AV-OS terminal was treated as a "black box." All the results are obtained through testing using the functions provided by the machine, and through observations of how the terminals' use and behavior is recorded in its event logs.

Related Work. Numerous research papers have been published on the topic of log analysis, both in the area of election audits, as well as in other digital systems areas [1, 3, 4, 18, 15, 8, 16]. In [16] Wallach et al. propose the generalized log analysis tool Querifier used to identify tampering in secure logs. A high-level framework for log systems is proposed in [9], emphasizing the importance of the completeness and exactness of the log files that determine the effectiveness of any potential log analysis system. This concern is shared across several papers dealing with election log analysis [1, 3, 18]. Baxter et al. [3] developed an automated analysis of audit log files produced by ES&S iVotronic to detect vote miscounts, machines with hardware problems, and polling locations with long lines. Audit logs have also been used to analyse elections for operator misconduct or process deviation in [1] with the focus on AV-OS. Further, Wagner gave an extensive study on the audit logs produced by six voting terminals approved for use in elections in California in [18]. The study examined the support provided by the various terminals for collecting, managing, and analysing the audit log files. The study established several deficiencies across all six systems, in particular in providing third party access to the files. The works [1, 3, 18] underlined that vendors ought to provide a rigorous documentation of the audit log features, including the structure of the logs.

Document Organization. Section 2 describes the election and audit process. Section 3 discusses the language of event logs. Section 4 contains the models and definitions. Section 5 presents the modeling of the election traces. Section 6 details the approach, its implementation and discusses our findings. Section 7 discusses the logging system deficiencies. Section 8 presents the results of using our system. We conclude with a discussion in Section 9.

2 The Election Process and Audits

Here we describes the election process and election audits. The model given here is specific to the State of Connecticut, but it is easy to see that it is generalizable to other jurisdictions.

Before Election Day: Preparations begin at least 30 days prior to the election day. The memory cards of AV-OS terminals are programmed for each precinct. The voting terminals also undergo maintenance and testing to detect any malfunctions and to help prevent failures during the election. The memory cards are programmed by a service company contracted by the State. The programming normally starts three weeks before the election and completes in under 10 days in most cases. Four programmed cards then are securely transported to each polling location. When the cards arrive, usually one to two weeks before the election, officials conduct pre-election tests on all the voting terminals with all the cards. Then the

officials randomly select two cards, one to be used in the election and and the other in the back-up terminal. The two selected cards are sealed in their respective voting terminals and are set for election; no further actions must be performed until the election day.

On Election Day: The election day imposes strict time constraints on when and what actions are performed. The summary of the activities is as follows.

Before The Polls Open: On the morning of the election day, from 5:00 to 6:00 AM before the polls open, the election officials verify the seals on each AV-OS terminal, turn it on, and confirm that the machines are properly initialized. This includes making sure all candidate counters are set to zero, by printing a Zero Totals Report.

While The Polls Are Open: Each eligible voter is entitled to a single ballot that s/he receives once they are verified against the voter registration database. Once the voter fills the ballot s/he feeds the ballot to the optical scanner of AV-OS.

After The Polls Close: After the polls close at 8:00 PM the officials print the totals report directly from the AV-OS terminal (the event log can also be printed at this point). The results are delivered to the central tabulation process where the totals are computed and reported to the Secretary of the State Office (SOTS) for certification. In jurisdictions that use automated central tabulation, the results can be electronically transferred to the central server either by uploaded directly from AV-OS terminals, or by transferring the cards to the central location for uploading to the server (this is not used in Connecticut).

Audits: Three independent audits are performed for each election (in Connecticut).

The *Pre-Election Technical Audit* involves the examination of one randomly chosen memory card from the four cards supplied to each district. Depending on the election, the audit typically covers at least 25% of the districts.

The *Hand Count Audit* is a post-election audit that consists of complete manual counting of a subset of races in 10% of precincts randomly selected after each election.

The *Post-Election Technical Audit* involves the examination of the memory cards used in the election, typically covering up to 30% of the districts.

The technical audits include examining the cards for proper programming and absence of extraneous or unexpected executable code [7]. As the audit results become available, SOTS Office follows up with the districts in cases that raise questions about malfunctions or potential deviations from the proper election process. The results of the follow up are also used to refine the audit process. Our current work deals with a detailed analysis of the event logs collected from the voting terminals.

3 The Language of Event Logs and the Previous System

The AV-OS voting terminal incorporates an event logging feature that records selected events and associated timestamps in an internal log. The log can be printed or extracted for analysis. The AV-OS documentation focuses on the meaning of individual events that are logged, but it does not provide a definition of the structure of the event sequences that are to be considered correct with respect to a properly executed electoral process (this is in part sensible, given that jurisdictions may have different expectations of what constitutes a proper process). For these reasons there is a need to reconcile the event logging features of the voting terminal with an externally-specified definition of a proper electoral process. Thus any model, including the finite state machine model in [1] and the model in the current work, that aims to describe the language of valid log sequences is *a*) an approximation, and *b*) designed empirically. The construction of such models requires that event log sequences

are manually examined and interpreted. Furthermore, the language of the model needs to be routinely extended and revised to include all observed event log sequences (whether they are representative of expected proper sequences or not). Even with a model that is nearly complete, it is still expected that certain event log sequences may be flagged as invalid without necessarily deviating from the prescribed election procedures. If such a log sequence is encountered, the language of the model needs to be refined to accommodate the newly discovered behavior. Consequently, refining of the model is crucial.

The previous analysis tool [1] classifies log files as normal, i.e., showing a sequence of events expected within a normal course of an election, or as irregular, i.e., containing unexpected events, unusual sequences of events, and unanticipated timing of events. The system is modeled as a finite state machine that simulates the states that the AV-OS terminal may enter during an election process.

Over the course of several audits, two major drawbacks have become apparent. Firstly, the finite state model proved to be difficult to extend and modify. Secondly, invalid log sequences that feature more involved deviations from the norm challenge low-level state machine implementations which cannot produce appropriate and concise notifications (we provide a concrete example in Section 7). Instead of recognizing the main cause of the irregularity, the tool often reports only on its secondary manifestations. This increases the need for substantial additional manual analysis.

To provide a better system that issues precise and meaningful notifications, a mapping between a sizable known set of irregularities and notifications needs to be established. Extending the state machine underlying [1] would lead to a substantial increase in its size. The inherently low-level abstraction of the state machine obscures the higher-level, logical view of the expected event sequencing. Finally, the previous tool does not provide an assessment of the severity of deviations. A high number of notifications (e.g., the several hundred event logs examined after the 2012 election generated over 1,000 notifications), diminishes the benefit of automation when the system is unable to identify event sequences that truly deserve to be manually examined. To be effective, the systems must be able to assign to each notification a severity ranking and meaningfully identify the source of the deviations.

4 Models and Definitions

The section details the models underlying the analysis tool and the terminology for the event log analysis. It defines the notions of an election trace and election trace compliance captured through an attributed grammar.

Event Log Terminology. A *log file* is the data recorded on a memory card during the election process and it contains a sequence of entries of the types *event*, *time*, and *date*. Event entries are logged when certain events occur. A restart of the machine, for example, leads to a corresponding event entry. Time and date entries are associated with event entries and specify the time at which the event is logged. A *trace* refers to the sequence of entries contained in an event log.

Attributed Grammar for Traces. The trace language is defined in terms of a context-free grammar (CFG) [6] given as a quadruple $G = (V, \Sigma, P, Start)$. V is a finite set called the *non-terminals* or *variables*; (Capitalized below). Σ is a finite alphabet, disjoint from V, called the *terminals*; (*italicized strings* below). P is a finite set of *productions* or *rules*, with each production written as $B \to \alpha$, where B is a variable, and α is a string from $(V \cup \Sigma)^*$; Lastly, $Start \in V$ is the start symbol.

A non-terminal B *derives* a string $w \in \Sigma^*$, written $B \stackrel{*}{\Rightarrow} w$ if there is a sequence of productions in P by which B can be transformed into w, i.e., $B \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow w$ where each α_i is a string of grammatical symbol and $\Rightarrow$ denotes the one-step expansion in the grammar. The language defined by G is the set of all strings (in our case traces) that can be derived in G. An attributed grammar [11] M is a CFG G augmented with a finite set A of *typed attributes* that represent semantic values associated to the corresponding grammatical symbol. Semantic rules are used to define the semantic attribute of a non-terminal as a function of the semantic values of the symbols appearing in its defining productions [17]. A string in the language yields a parse tree, where non-terminals are internal nodes, terminals are the leaf-nodes, and productions determine the parse tree topology.

The attributed grammar presented in this paper models the language of the election traces represented by event logs. The non-terminals of the grammar models the stages of the election. The terminals of the grammar (the alphabet) are the event entries. Finally, the time and date entries are used to initialize attributes associated with the terminals at the leaf-nodes, with the rest of the attributes derived by traversing parse trees.

Election Traces and Compliance. Let the language L_{all} be the set of all traces that are known to be produced by the event logging system. This language is defined on the basis of our prior work [1], and by consulting the AV-OS documentation and by hands-on experimentation with the AV-OS voting terminal. The attributed grammar must be able to generate all traces in L_{all}.

We partition L_{all} into two subsets, $L_{sc} \cup L_{snc}$. L_{sc} contains all syntactically compliant traces. A trace is *syntactically compliant*, if its event entries occur in the expected order, and at an appropriate stage in the election process. The determination of compliance is made on the basis of the process reviewed in Section 2, and specifically by adhering to the official requirements [14, 12] that impose timing and sequencing constraints. L_{snc} contains all *syntactically non-compliant* traces. These traces are known syntactic deviations (based on [1] and experimentation) that need to trigger notifications when recognized by the parser.

Apart from syntactic compliance, we also consider semantic compliance. A trace is *semantically compliant*, if its event entries occur a permissible number of times, and within the correct time frame. To establish the *semantic compliance* of a trace, semantic analysis is performed by imposing additional constraints whose truth values are expressed by semantic attributes and their defining equations over attributes in A. We present the constraints and attributes in Section 6.1. Semantic analysis is performed on all traces in L_{all}.

We define a trace as *compliant*, iff it is syntactically and semantically compliant. We let the language L_c be the language of all compliant election traces. A trace $s \in L_c$ iff $s \in L_{sc}$ and s is semantically compliant. We define all traces not in L_c to be *deviations*.

Finally, it is possible to encounter traces that are not in L_{all} when (1) an unrecognized trace may be caused by the voting terminal malfunction (or an error in its software), (2) memory card corruption may cause errors in the stored trace, and (3) a previously unknown trace pattern is encountered (e.g., because voting terminal documentation is incomplete or because this pattern was never seen before), in which case the definition of L_{all} needs to be extended to account for such trace patterns. We define L_u to be the set of all traces that are not in L_{all}, or in other words, L_u is the complement of L_{all} with respect to the (unknown) set of traces that can ever be encountered.

Analysis Flow. Given a grammar G and its attribute system A that generates L_{all}, the analysis of a trace s is as follows. If $Start \stackrel{*}{\not\Rightarrow} s$ (trace s cannot be derived), then $s \in L_u$, and it is neither a compliant trace, nor a known invalid deviation. No further automated analysis

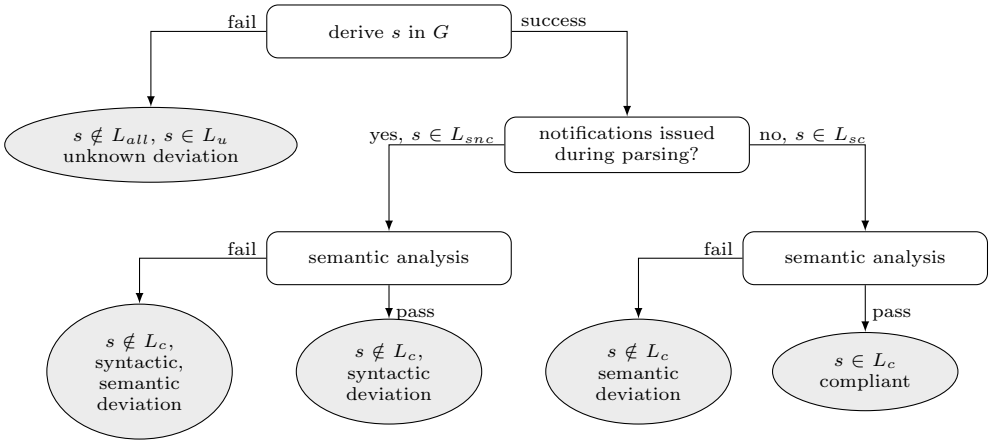

Figure 1: Analysis Flow

is performed. Such a trace needs to be analyzed manually (in the sequel we observe that this happens very infrequently). If $Start \stackrel{*}{\Rightarrow} s$, two cases are possible: (i) during the derivation one or more notifications are issued, indicating that $s \in L_{snc}$, or (ii) no notifications are issued, indicating that $s \in L_{sc}$ and s denotes a validly ordered sequence of events. Finally, the trace s undergoes semantic analysis. If any deviations in the timing, or the number of occurrences of events are observed, then $s \notin L_c$, and appropriate notifications are issued. If s does not induce any notifications, it is a compliant election trace, i.e., $s \in L_c$.

5 Defining the Grammar

The grammar for the language of event traces was developed through an iterative process. Based on our analysis, we include all traces that the voting terminal is able to produce, while delineating between syntactically compliant and non-compliant traces. Regression testing is performed against a known collection of traces. When a trace is found that cannot be generated by the grammar, it is examined manually. If the trace adheres to the requirements [14, 12], the grammar is extended to allow it. If it does not, analysis is performed on whether it is a valid trace per tabulator documentation. If it is, the grammar is again appropriately extended. Otherwise analysis is performed to understand the circumstances that lead to this kind of trace. If the behavior cannot be reproduced, we suspect a tabulator malfunction, else we conclude that there is a defect in the tabulator logging system. Decisions on whether to extend the grammar is made on a case-by-case basis.

5.1 Events Recorded in the Log: the Terminals of the Grammar

Table 1 conveys the event names as they are recorded by AV-OS in the log file, with an explanation. Recall that the log file contains *event*, *date* and *time* entries. All event entries in the log are associated with the time entry indicating when the event occurs. Some event entries, i.e., INITIALIZED and SESSION START are also followed by a date entry. Each event entry in Table 1 corresponds to a terminal symbol in the alphabet of the grammar given in Figure 2. In the grammar specification we represent each terminal symbol as the lower-case italicized string corresponding to the event name.

The temporal information associated with the events, i.e., time and date, is not represented in the (non-attributed) grammar. This information is filtered out during the lexical analysis prior to parsing a trace, and is preserved and associated with each event in the trace as its time attribute. We discuss the processing of these attributes in Section 6.1.

Event Name	Event Description
AUDIT REPORT	Appears when an Audit Report is printed.
BAL COUNT END	After the ender card is inserted in an election, this action appears.
BAL COUNT START	Appears when the first ballot is cast in an election.
BAL TEST START	Records the beginning of a test election.
CLEAR COUNTERS	Appears when the counters are set to zero.
COM ERROR	A communication error between the machine and the GEMS system.
COUNT RESTARTED	The machine is reset during an election, after at least one ballot is cast.
DOWNLOAD END	Records the end of data load to the card when using GEMS.
DOWNLOAD START	Records the start of data load to the card when using GEMS.
DUPLICATE CARD	Records that a card duplication occurs (in the master card and the copy).
ENDER CARD	Records when an ender card is inserted, signifying the end of an election.
INITIALIZED	The 1st action in the Event Log; this action records date.
MEM CARD RESET	The card is returned to 'not set' status, if it was set for election.
OVERRIDE	Records an override by a poll worker. Used for overvoted ballots in CT.
POWER FAIL	Appears if the machine is unplugged or a power failure occurs.
PREP FOR ELECT	Recorded when the card is set for election.
SESSION START	Date action. Appears every time you reset the machine.
TOTALS REPORT	Appears when a Totals Report is printed.
UNVOTED BAL TST	Appears when an unvoted ballot test is performed.
UPLOAD END	When an upload is completed, this action is recorded.
UPLOAD ERROR	Appears when an upload error is detected.
UPLOAD STARTED	Marks the beginning of an upload.
VOTED BAL TEST	Appears when an voted ballot test is performed.
ZERO TOT REPORT	Appears when a Zero Totals Report is printed.

Table 1: Action Types

5.2 Election Trace Grammar: Non-terminals and Productions

The context-free grammar in Figure 2 defines the syntactic structure of all election traces that can be produced by the AV-OS terminal during the election process, from the time a card is programmed, until the election is closed and the results are printed. The grammar is designed to generate the language L_{all}, that is, all election traces that, given the current understanding, can be produced by interacting with the AV-OS terminal. Recall that $L_{all} = L_{sc} \cup L_{snc}$, where L_{sc} are the syntactically compliant traces and L_{snc} are the syntactically non-compliant traces. The productions that are involved in generating any trace in L_{snc} account for known, non-compliant behavior and issue notifications whenever they are triggered during the parsing process.

The productions in the grammar are given in extended Backus-Naur Form (readers familiar with grammars should be able to read it). Reiterating briefly, the non-terminals are given as Capitalized strings and the terminals are the *italicized* strings. Each production has the form "NonTerminal = right-hand-side", where the right-hand-side is a sequence of terminals and non-terminals. The notation "A → rhs1 | rhs2" is a shorthand for two production, "A → rhs1" and "A → rhs2". Zero or one repetition of an expression is denoted

```
1:  Start →  Init, TestPrepElect?
2:         | AbortedInit;

3:  Init →  Initialized, Downloadstart+, Clearcounters, Downloadend
4:        | Initialized, Downloadstart+, Clearcounters, Prepforelect;

5:  AbortedInit →  Initialized, Downloadstart*;

6:  TestPrepElect →  (Test | Test, Prep | Test, Prep, Elect), CardReset?;

7:  CardReset → Memreset, TestPrepElect;

8:  Test →  (Votedbaltest | Unvotedbaltest | Counttestbal)*;

9:  Counttestbal →  Clearcounters, Zerototsreport?, Balteststart?, Override*,
                    Endercard, Balteststart?, Printtotals*;

10: Prep →   Prepforelect, Clearcounters
11:        | Prepforelect;

12: Elect →  BeforeValBalCast
13:        | BeforeValBalCast, (Balcountstart, Countrestart)*
14:        | BeforeValBalCast, (Balcountstart, Countrestart)*, Endercard,
             Balcountstart, Balcountend, Printtotals*
15:        | BeforeValBalCast, (Balcountstart | Override), AcceptingMoreBallots,
             Endercard, Balcountend, Printtotals*;

16: BeforeValBalCast → Zerototsreport, (Zerototsreport | Balcountstart Zerototsreport)*
17: AcceptingMoreBallots →  (Countrestart | Countrestart Balcountstart | Override)*;
18: Global → audit report | com error | duplicate card | power fail | mem overflow | session start;

19: Balcountend →  Global*, bal count end;
20: Balcountstart →  Global*, bal count start
                   | Global*, bal count start, bal count start;
21: Balteststart → Global*, bal test start;
22: Clearcounters →  Global*, clear counters;
23: Countrestart →  Global*, count restarted;
24: Downloadend →  Global*, download end;
25: Downloadstart →  Global*, download start;
26: Endercard →  Global*, ender card;
27: Initialized →  Global*, initialized;
28: Memreset →  Global*, mem card reset;
29: Override →  Global*, override;
30: Prepforelect →  Global*, prep for elect;
31: Printtotals →  Global*, totals report;
32: Unvotedbaltest →  Global*, unvoted bal test;
33: Votedbaltest →  Global*, voted bal test;
34: Zerototsreport →  Global*, zero tot report;
```

Figure 2: Context-free grammar for AV-OS event traces

by the post-fix "?", zero or more repetitions is denoted by the post-fix "*", and one or more repetitions is denoted by the post-fix "+".

To implement the notification of deviant traces, we rely on `Bison` [5] actions that execute an arbitrary piece of C++ code when a grammatical production characterizing a deviation is reduced. Each paragraph below describes a non-terminal, the productions where it appears as the left-hand-side, and the relationship to the election traces. Rule numbers refer to the line numbers in Figure 2.

Start. This is the start symbol of the grammar. An election trace can either have a completed Initialization stage followed by an optional sequence of Test Elections, Preparing for an Election, and Elections (rule 1), or an aborted Initialization stage, in which case no further events can be recorded in the log file (rule 2).

Init. During the Initialization stage a memory card is programmed. Rule 3 represents the expected sequence of events: an INITIALIZED event (dated), one or more DOWNLOAD START events, a CLEAR COUNTERS indicating that all counters have been zeroed, and finally a DOWNLOAD END event, indicating that the card has been successfully programmed with the data for the selected election. Additionally, it has been observed in at least one trace that a PREP FOR ELECT was substituted for the DOWNLOAD END event. This invalid behavior is accounted for by rule 4. A corresponding notification is issued. (This newly discovered logging deficiencies is discussed later.)

AbortedInit. Traces have been encountered where an INITIALIZED event is followed by zero or more DOWNLOAD START events. This sequence occurs when a card is not successfully programmed. E.g., if a card's memory is cleared through the supervisor function at the precinct and reprogrammed (without connecting to an election management system), the download process is aborted and no DOWNLOAD END event is recorded. A notification is issued if rule 5 is triggered since this sequence indicates an incorrect procedure.

TestPrepElect. Production rule 6 encompasses all traces that can be generated after the Initialization stage, that is, after the card has been successfully programmed. The expected course of events here is a sequence of test-election related events accounted for by the non-terminal Test, followed by the card being prepared for the election mode, accounted for by the non-terminal Prep, and lastly events related to the actual election process, accounted for by the non-terminal Elect. Note that the production rule includes the following disjunction: (Test | Test, Prep | Test, Prep, Elect). This disjunction accounts for the cards that have not been used in the actual election process. These cards only show events related to test elections and possibly preparing for the election, but no election events. In this case a notification is issued, but only for the post-election audit because in this case it is expected that the trace shows an election. Lastly, it is possible to reset a memory card to the pre-election stage through the supervisor functions at any point after the card has been programmed. This is accounted for by the optional CardReset non-terminal. However, resetting the card is not a compliant behavior and thus it is associated with a notification.

CardReset. In case a card is reset during the election stage a MEM CARD RESET event is recorded and the card returns to the pre-election stage. At this point more test elections can be run, the card can be prepared for election and subsequently elections can be run.

Test. This non-terminal derives election traces produced during the pre-election stage. At this stage, voted and unvoted ballots can be tested as well as test election performed. Events logged during a test election are accounted for by the Counttestbal non-terminal. All of these events are optional: it is admissible to skip this stage entirely and simply proceed to preparing the card to the election.

Counttestbal. This corresponds to the events produced if a test election is run. The procedure and traces recorded in the event log are similar to those of an actual election. A test election starts with the counters being cleared, in which case a CLEAR COUNTERS event is recorded. The user then has the option to print a zero totals report. If the user chooses to do so, a ZERO TOT REPORT appears in the log. Further a BAL TEST START is recorded once the first valid ballot is cast. Overrides can be performed, with each Override recorded in the log. The test election can be ended by inserting an enders card, which is recorded, or by turning the machine off, in which case only a SESSION START event is recorded. After an ender card is cast the user has the option to print a totals report. If an ender card is inserted before any other ballots are cast the system records an ENDER CARD event followed by a BAL TEST START event. (This is a known defect in the AV-OS logging system [1].)

Prep. Before an election can be run the card needs to be set into election mode. In the pre-election stage the user has the option to prepare for election. If that option is chosen, a PREP FOR ELECT event is logged. All the counters are supposed to be zeroed and a CLEAR COUNTERS event recorded. A notification is issued if a CLEAR COUNTERS event is missing, which is accounted for by rule 11.

Elect. This non-terminal accounts for the events logged during the actual election stage. Rule 15 describes the expected sequence of events. After a card is prepared for election, the machine needs to be restarted. Upon restarting a zero totals report is printed. In this stage, before any valid ballot is cast, a restart of the system always yields another ZERO TOT REPORT event. If the first ballot cast is not a valid ballot, e.g., a ballot not recognized by the machine, a BAL COUNT START event is logged. However, since the counters aren't updated, upon restarting the machine another zero totals report is printed and logged. This behavior is accounted for by the BeforeValBalCast non-terminal. After the first valid ballot is cast, or an override is issued, the system continues to accept ballots until the ender card is inserted. This stage is accounted for by the AcceptingMoreBallots non-terminal. If the machine is restarted in this stage, a COUNT RESTARTED event is recorded. Inserting the ender card ends the election. An ENDER CARD event followed by a BAL COUNT END event is recorded. A totals report is printed and the user is asked if the election should be closed. Shutting off the machine without selecting an option at this point leads to a known error [1]: the TOTALS REPORT event is not logged even though a totals report has in fact been printed. Restarting the machine at this stage results in another totals report printed and logged. Rules 12 and 13 account for cards on which an election has been started but not successfully completed. Notification are issued when these rules are triggered. Rule 14 accounts for the scenario in which the ender card is inserted before any other valid ballots are cast.

BeforeValBalCast. After a card has been prepared for election but no valid ballot has been cast yet, restarting the machine results in a zero totals report. If any invalid ballot is cast at this stage a BAL COUNT START event is logged.

AcceptingMoreBallots. After the first valid ballot has been cast but the ender card has not yet been inserted, the machine accepts ballots. A restart in this stage results in a COUNT RESTARTED event. If a new ballot is cast, a BAL COUNT START event is logged.

Global. This non-terminal can yield any global event terminal, i.e., an event that can occur at any stage in the election process. Note that production rules 19 through 34 simply account for the fact that a global event can occur at any stage of the election process by prefixing any non-global event with a sequence of zero to many global events.

6 Event Log Analysis

Now we present the semantic analysis and the notification system.

6.1 Attributes and Semantic Analysis

The analysis is performed on the abstract syntax tree (AST) that is constructed during the parsing of a trace. The leaf nodes of the tree correspond to the terminals of the grammar in Figure 2, which in turn correspond to the event entries in election traces. The internal nodes of the tree correspond to stages in the election process, such as the test election stage, prepare for election stage, etc. Each node in the AST has a set of attributes. The semantic analysis consists of verifying that specified events lie within a permissible time frame, and occurred a

Attribute	Event	Scope
#S	session start	All
#MR	mem card reset	All
#PF	power fail	All
#CE	com error	All
#DS	download start	Init
#BT	bal test start	Test
#PT	totals report	Test, Elect
#ZR	zero tot report	Elect
#CR	count restarted	Elect

Figure 3: Multiplicity Attributes

Attribute	Event	Scope
TI	initialized	Init
TBT	bal test start	Test
TPR	prep for elect	Prep
TZR	zero tot report	Elect
TBE	bal count end	Elect

Figure 4: Time Stamp Attributes

permissible number of times. These constraints are expressed through the attributes of the tree nodes. Attributes are synthesized throughout the tree, i.e., derived by a parent node from its children in a bottom-up fashion. There are three types of attributes: *time-stamp*, *multiplicity*, and *constraint* attributes. A time-stamp attribute of some node D in the AST corresponds to the time-stamp of a specified leaf node in the subtree of D. A multiplicity attribute of node D in the AST corresponds to the number of times some specified node E occurs in the subtree of D. A constraint attribute of node D in the AST corresponds to a multiplicity or time stamp constraint defined over the attributes of the subtree of D. A constraint attribute can be defined over the attributes of a specific node in the tree, or over the attributes of several nodes in the tree.

Three components contribute to the semantic analysis: *nodes*, *attributes*, and *constraints*.

Nodes. There are seven different types of nodes in the AST. The leaf nodes of the AST are defined as *event nodes*, and as mentioned previously, correspond to the terminals of the grammar. The AST expresses the grouping of events into election stages through node hierarchy. The internal nodes $Init$, $Test$, $Prep$, and $Elect$ correspond to the four different stages of the election process, namely programming the card, the pre-election test mode, preparing for an election, and the actual election stage. Their descendant nodes correspond to the events that are recorded during a given stage in the election process. The $TestPrepElect$ node expresses the fact that the Test, Prep, and Elect nodes are a unit. Every time a memory card is reset, the card returns to the pre-election mode. From there the pre-election, prepare for election, and election stages can be repeated. Each such repetition corresponds to a $TestPrepElect$ node. Finally, the All node is the root node of the tree.

Attributes. Figure 3 defines multiplicity attributes, and Figure 4 defines the time-stamp attributes. The Attribute column gives the names of the attributes, the Event column shows the event with which the attribute is associated, and the Scope column shows over which subtree the attribute is defined, e.g., the scope of the attribute. If, for example, the Scope column has the value Test, the attribute is defined over each subtree with root node Test.

Attributes are defined inductively over the nodes of the tree. A multiplicity attribute is a single integer. Given a multiplicity attribute $\#M$ that tracks the multiplicity of event e in the subtree with root node R, attribute $\#M$ is defined as follows.

$$R.\#M = \begin{cases} \sum_{c \in children(R)} c.\#M & \text{if } R \text{ is an internal node,} \\ 1 & \text{if } R \text{ is a leaf, and corresponds to } e, \\ 0 & \text{otherwise} \end{cases}$$

A time-stamp attribute is a set of time-stamps. Given a time-stamp attribute TI that tracks the time stamps of all occurrences of event e in the subtree with root node R, attribute TI has the following definition.

$$R.TI = \begin{cases} \bigcup_{c \in children(R)} c.TI & \text{if } R \text{ is an internal node,} \\ \{e.timestamp\} & \text{if } R \text{ is a leaf, and corresponds to } e, \\ \emptyset & \text{otherwise} \end{cases}$$

Constraints. A constraint attribute is a boolean predicate defined over the time stamp and multiplicity attributes of the nodes. Currently we use two types of constraint attributes: multiplicity constraint attributes and time-stamp constraint attributes. A multiplicity constraint attribute PM of node N defined over multiplicity attribute $\#X$ of node N has the following general form:

$$N.PM = (r_1 < N.\#X < r_2)$$

where r_1 is the lower bound on the number of occurrences, and r_2 is the upper bound.

A time-stamp constraint attribute PT of node N is defined over its time-stamp attribute TX as follows:

$$N.PT = (\forall t \in N.TX,\, ts_1 < t < ts_2)$$

where ts_1 is the earliest date and time on which the tracked event can occur, and ts_2 is the latest date and time. Should a constraint attribute evaluate to false, a notification is issued stating that a semantic constraint has been violated.

We note that the time-stamp constraints are crucial to the analysis due to the time-sensitive nature of the election. We build on the foundation for the temporal analysis of election traces given in [1]; following that work we next provide a detailed example of the temporal restrictions on the election process.

The election day order imposes time limitations on when and what actions can be performed. According to the election process we expect to see the following: from 5:00 to 6:00 AM election officials should turn on the AV-OS terminal to be used in the election and produce a zero totals report. Thus we expect to see a session start (SESSION START) event about an hour before the polls open, followed by a zero totals report (ZERO TOT REPORT) event. We expect to see the ballot count start (BAL COUNT START) event after the time the polls open. This can be followed by a sequence of override (OVERRIDE) events. Finally, by the time the polls close, we expect to see the ender card (ENDER CARD) event, followed by a ballot count end (BAL COUNT END) and a totals report (TOTALS REPORT) events.

Example AST. Figure 5 shows an example trace, and the resultant AST is given in Figure 6. In the interest of space we abbreviate the names of the leaf nodes; the abbreviations are given next to the event names in Figure 5. The tree is augmented with four attributes: $\#S$, TI, PM, and PT. The attributes are written in italic.

For the sake of clarity we have only included four attributes in the tree. Attribute $\#S$ is a multiplicity attribute which tracks the number of occurrences of the SESSION START event throughout the entire election process. Attribute TI is a time-stamp attribute that tracks the time-stamps of all INITIALIZED events that are logged during the Initialization stage. PM and PT are constraint attributes. PM is defined for the All node of the tree, and restricts the SESSION START event to occur between 0 and 3 times: $All.PM = 0 \leq All.\#S \leq 3$. PT is defined for the Init node of the tree, and restricts the time for all INITIALIZED events in the Initialization stage to occur between 0:00 on 10-20-12 and 11:59 on 10-30-12: $Init.PT$ = ($\forall t \in Init.TI$, 0:00 10-20-12 $\leq t \leq$ 11:59 10-30-12). The attributes are synthesized throughout the tree as described in the previous section.

Remark. The semantic analysis presented above relies on semantic attributes and constraints to recognize non-compliant traces. Programming languages do rely on type systems

Event	**Abbreviation**	**Time**	**Date**
initialized	init	12:00	10-20-12
download start	dlstart	12:00	10-20-12
clear counters	clrc	12:00	10-20-12
download end	dlend	12:00	10-20-12
session start	sstart	11:00	10-24-12
bal test start	btest	11:05	10-24-12
ender card	ender	11:07	10-24-12
totals report	trep	11:10	10-24-12
prep for elect	prep	11:20	10-24-12
clear counters	clrc	11:20	10-24-12
session start	sstart	05:30	11-06-12
zero tots report	zrep	05:31	11-06-12
bal count start	bcstart	08:01	11-06-12
ender card	ender	20:14	11-06-12
bal count end	bcend	20:14	11-06-12
totals report	trep	20:14	11-06-12

Figure 5: Example Election Trace

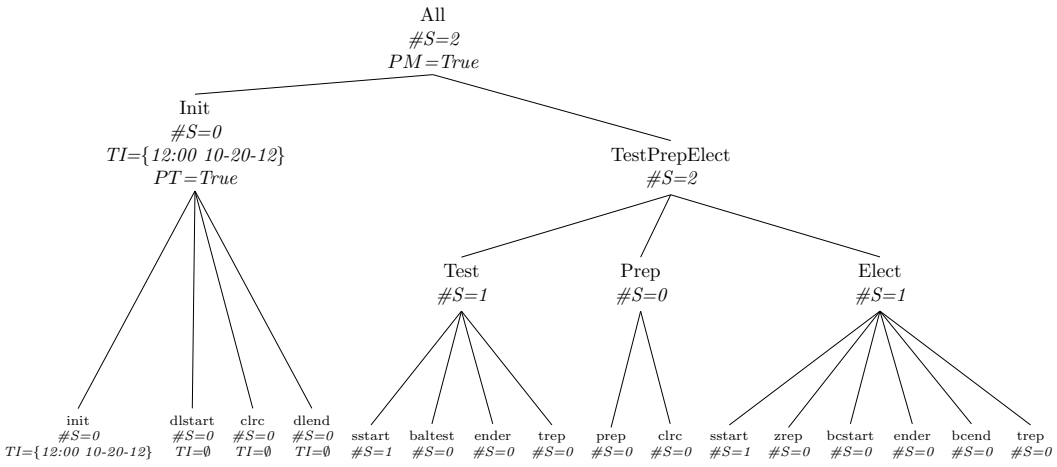

Figure 6: Abstract Syntax Tree (AST)

to annotate and analyze parse trees, hence, it it tempting to assess their potential for our analysis. Given the key roles of multiplicities and timestamps in such an analysis, one must consider so-called *dependent type systems* [13] to capture, within types, the critical inputs used by the analysis. While this is certainly a feasible avenue, it is not clear that it would significantly simplify the specification currently captured through constraints.

6.2 Notification System

The outputs of the parsing process and the semantic analysis manifest themselves through the notification system. It includes four types of notifications: *syntax errors* that are issued when an election trace cannot be parsed, *grammar notifications*, issued when deviant

productions are triggered during parsing, *multiplicity notifications*, issued when a specified event occurs too few, or too many times, and *time notifications*, issued in case a specified event occurs outside of its permissible time range. Both repeat and time notifications are generated during the semantic analysis, after the input file has been successfully parsed.

Syntax errors are rare and signify that the offending log file shows major deviations from the expected format (suggesting an unknown software or hardware malfunction in the voting terminal, or a failure of the memory card that stores the log). In a recent analysis of over 500 memory cards, we have encountered only one syntax error. Once a syntax error is encountered and the manual examination suggests that it is not due to equipment failure, the grammar can be amended with additional productions so that similar future errors are recognized and flagged accordingly without raising another syntax error. Analysis of syntax errors is crucial in identifying equipment failures and/or exposing new deficiencies in the logging system. (Instances of this are discussed in Section 7.)

Grammar notifications are issued to account for known, erroneous behavior where multiplicity or timing semantic constraints are violated. E.g., if the last event recorded in the log is a BAL COUNT START event, with all the subsequent, expected events missing, a notification is issued stating that the election has been aborted midway. All notifications carry custom messages to concisely explain a specific, known deviation from the expected behavior.

7 New Identified Event Logging Deficiencies

The implementation of the event logging in the AV-OS terminal is known to contain several defects and deficiencies [1]. Our systematic approach helped reveal additional issues with the AV-OS logging. Although doing so was not among the goals of our development, we note that it is the rigor of our approach and the comprehensiveness of our analysis that enabled the identification of new defects and deficiencies.

Manifestations of Logging Deficiencies. Here we elaborate on the deficiencies in the AV-OS logging system exposed by our analysis, provide examples, and discuss the ramifications. Consider the three deviating traces shown in Figure 7. For brevity, we only show the relevant section of a given trace. The three event traces are real—these have been recorded during the official voting terminal use.

Event log A in Table 7 shows an election run, followed immediately by a PREP FOR ELECT event. It is possible, after an election, to reset a card to the pre-election stage. Resetting a card is done through the supervisor functions and is recorded in the event log as a MEM CARD RESET event entry. In Log A, however, no MEM CARD RESET is logged between the concluded election and the PREP FOR ELECT event. We have not been able to reproduce such a log sequence using plausible scenarios, leading us to suspect that this is either an intermittent error or unintended behavior due to a race condition. The time-stamp of the first election indicates it was run before the official election date, during the test stage. We have encountered several event logs suggesting that in testing the voting terminal an election was run instead of a test election (while it can be beneficial to run an election instead of test election during pre-election test, the current official rules require that test election is run). This incorrect usage appears to be harmless. However, log A suggests a machine malfunction. Not only is the MEM CARD RESET event absent from the log, we also see that the PREP FOR ELECT event is not followed by a CLEAR COUNTERS event, thus the next election starts with a COUNT RESTARTED event. This indicates that the counters from the previous election were not zeroed, and thus the totals of this subsequent election

are incorrect. Fortunately, in this case, the subsequent election is merely another election run as a test election, so no election data was compromised. However, the same machine malfunction could affect actual election counts.

Event Log A	**Event Log B**	**Event Log C**
10-31-12	11-03-12	11-04-08
15:38 ZERO TOT REPORT	11:05 PREP FOR ELECT	06:41 ZERO TOT REPORT
09:29 BAL COUNT START	11:05 CLEAR COUNTERS	07:00 BAL COUNT START
09:31 ENDER CARD	14:11 SESSION START	07:05 SESSION START
09:31 BAL COUNT END	11-06-12	11-04-08
10:32 TOTALS REPORT	14:26 ZERO TOT REPORT	07:06 ZERO TOT REPORT
10:33 SESSION START	15:18 BAL COUNT START	07:07 BAL COUNT START
11-01-12	15:34 BAL COUNT START	20:58 ENDER CARD
10:33 PREP FOR ELECT		20:58 BAL COUNT END
10:33 SESSION START		21:02 TOTALS REPORT
11-01-12		
10:33 COUNT RESTARTED		
10:34 ENDER CARD		
10:34 BAL COUNT START		
10:34 BAL COUNT END		
10:52 TOTALS REPORT		

Figure 7: Event logs causing syntax notification (A) and grammar notifications (B and C)

Event Log	**Type**	**Severity**	**Message**
Log A	Syntax	10	Syntax Error at line 33.
Log B	Grammar	2	Two consecutive BAL COUNT START events.
Log B	Grammar	2	Election aborted before Balcountend.
Log B	Date	1	PREP FOR ELECT occurred at 11:05 11-03-12.
Log B	Date	2	ZERO TOT REPORT occurred at 14:26 11-06-12.
Log C	Grammar	1	ZERO TOTALS REPORT after BAL COUNT START.

Figure 8: Notifications issued for event logs A, B, and C

This situation was identified because our system produced a syntax error notification, as shown in Figure 8. We note that the analysis tool in [1] failed to report this log file as non-compliant.

The analysis of event log B (Figure 7) results in two grammar and two time-stamp notifications (Figure 8). The grammar notifications indicate that two consecutive BAL COUNT START events occur and that the election has not been properly concluded. The BAL COUNT START event should only occur when the first ballot is cast in an election, or if a ballot is cast after the machine has been restarted. The latter should be preceded by a SESSION START and a COUNT RESTARTED. Log B, however does not display this sequence of events. The consecutive BAL COUNT START events indicate another deficiency. Log B initially resulted in a syntax error, however, the addition of an error-handling production (production 19, Figure 2) accounts for this behavior and issues an appropriate grammar notification. Time stamp notifications indicate that two events have occurred outside of their allotted time-frame.

Log C exposes another logging system deficiency. A BAL COUNT START at 7:00 is logged, followed by a SESSION START and a ZERO TOTALS REPORT. The BAL COUNT START event indicates that a ballot has been cast, while the ZERO TOTALS REPORT indicates that the counters are

at zero. This means that casting the first ballot did not increment the counters. This event trace occurs when the first ballot cast in an election is not recognized by the machine. The ballot is cast and the BAL COUNT START event is logged, however, since the ballot cannot be read, the counters are not incremented.

Discussion of Mitigations. In light of the above problems we strongly recommend that a rigorous analysis of the AV-OS hardware and software is performed by the vendor to determine the cause of the machine malfunction captured in log A, and the logging deficiency in log B. Further we recommend that the training of poll officials be improved.

There is also an enhancement to the AV-OS logging system that would greatly reduce the ambiguity in the logs. Currently it is impossible to discern whether a memory card was used in one terminal only, or in several different terminals throughout the election process. In fact, the established procedures implicitly require that every memory card that was used in an election are used in at least two different terminals. First, each card is programmed on one terminal (using the election management system), then it used with a different terminal at the precinct during the actual election. Each precinct has two voting terminals, and any card can be switched at any time from one terminal to the other. This causes ambiguity in the log analysis assessment and presents the potential for masking incorrect tabulation or improper use of the voting terminals. One way to approach this is to augment the SESSION START action type. The SESSION START event is logged whenever a voting terminal with a memory card already inserted is turned on, or when a memory card is inserted into a terminal. Currently it is impossible to distinguish between these two scenarios because the terminal is never identified inside the event log. Supplementing the SESSION START event with a paired event that records a unique voting terminal identifier would resolve the ambiguity.

8 Running the Event Trace Analysis System

The analysis tool is implemented in C++, using `Bison` [5], a GNU parser generator, and using `Flex` [10] for lexical analysis. The grammar is an unambiguous, zero-conflict, deterministic LR(1) grammar. In the course of the development, refinement, and regression testing of our system we used it to analyze several thousand event logs collected in recent years from actual elections. The performance of the system is quite good, certainly making it feasible to rapidly analyze large numbers of event logs. In particular, running the analysis tool on a conventional laptop, the system is able to process over 150 log files per second on average.

Case study. We review the results produced by the analyzer on a data set of 421 event logs from the 2008 Presidential Election (these represent a total of 30% of all districts: 10% randomly selected districts and 20% of districts that chose to participate in the audit); 279 logs were collected from AV-OS voting terminals used in the election, and 142 logs were collected from the back-up terminals. The date and repeat notifications produced by our tool are consistent with those reported in [1]. We now summarize selected observations.

In event logs from 19 terminals (four of these were used in the election) we observed a MEM CARD RESET event. Resetting a memory card places it in the pre-election state. However, the election protocol states that memory cards should not be reset. Fortunately, all 19 cards show an election run during the test election stage. Thus this is not problematic, since preparing for election zero the counters anyway. Nevertheless, the rules disallowing resetting are there for a purpose. If a card were to be reset in the middle of an election, after ballots have been cast, the counters would be zeroed and votes lost. For example, one event log containing MEM CARD RESET events was reset on and used on the election day. The

MEM CARD RESET event is logged less than twenty minutes before the first ballot is cast in the election. Resetting a card on the election day should not occur under any circumstance.

In two event logs we found incomplete election sequences. Both logs end on a BAL COUNT START event logged on the election day. All further expected event entries are missing. This indicates that either the election was not completed properly, or that the memory card was removed before the election was concluded. However, the cards have zero counters, thus no ballots were cast.

Additionally, we discovered events that were not supposed to occur in post-election event logs. The AUDIT REPORT events are recorded in two logs. Two logs contained UPLOAD START events. Neither the printing of an audit report nor the upload of any data should occur during the election process. This indicates the election procedure was not followed.

Logging system deficiencies. We have also identified new deficiencies in the event logging system. As we already discussed earlier (and in [1]), these deficiencies result in ambiguity and could in principle be used to mask invalid operator behavior.

Our analysis detected three event logs showing a BAL COUNT START event, followed by a ZERO TOTALS EVENT. This exposes a deficiency in the logging system and suggests that the first ballot cast in the election was not read correctly by the machine. It is impossible to establish whether the ballot cast was indeed invalid, or whether the machine simply failed to read it.

Another logging deficiency related to ballot casting was observed in one log where an election is run during the test election stage. The (out-of-order) sequence of events for the election is as follows: ENDER CARD, BAL COUNT START, BAL COUNT END. We have been able to reproduce this log trace by inserting the ender card before any other ballots are cast. Such an event sequence occurs if the first ballot cast in an election is the ender card.

While the results of our analysis confirm that in several instances election procedures were not followed, we have found no indication of security problems or malicious intent. However in some cases the analysis suggests software/hardware malfunction.

9 Conclusions and Future Work

Election audits are a critical procedural component of the electoral process to guarantee the proper conduct of an election. Our work demonstrates yet again how audits can be valuable in the forensic analysis of data collected from voting terminals used during the election. Indeed, the audit process reveals several classes of problems ranging from voting terminal malfunctions and defects to deviations in the recommended behaviors for system operators. Our contributions encompass a new formalization of voting machine event logs to systematize a multi-layered compliance analysis that delivers detailed notifications characterizing election traces. The event log analysis uses attributed context-free grammars, making the system highly extensible and maintainable, and readily available for refinements that reflect requirements for a correct conduct of an election. Additionally, our methodology led to the identification of previously unknown deficiencies and defects in the AV-OS logging system, further emphasizing the value of comprehensive audits.

We are currently preparing recommendations on implementing event logging systems for voting terminals that would enable even more comprehensive audit analyses. In our future work we will continue refining our approach and we intend to adapt the language definition for use in other jurisdictions using similar equipment. Other research directions prompted by our work include the exploration of machine learning as a means for automating

the generation of grammars, and the development of an automated approach for aligning grammars with the requirements and expectations of other jurisdictions. Finally, we intend to make our system available for educational and research use.

Acknowledgments. The authors thank the referees for detailed and insightful comments that resulted in substantial improvements to our presentation.

References

[1] Antonyan, T., Davtyan, S., Kentros, S., Kiayias, A., Michel, L., Nicolaou, N., Russell, A., and Shvartsman, A. Automating voting terminal event log analysis. In *Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections* (Berkeley, CA, USA, 2009), EVT/WOTE'09, USENIX Association, pp. 15–15.

[2] Antonyan, T., Davtyan, S., Kentros, S., Kiayias, A., Michel, L., Nicolaou, N. C., Russell, A., and Shvartsman, A. A. State-wide elections, optical scan voting systems, and the pursuit of integrity. *IEEE Transactions on Information Forensics and Security 4*, 4 (2009), 597–610.

[3] Baxter, P., Edmundson, A., Ortiz, K., Quevedo, A. M., Rodríguez, S., Sturton, C., and Wagner, D. Automated analysis of election audit logs. In *Proceedings of the 2012 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections* (Berkeley, CA, USA, 2012), EVT/WOTE'12, USENIX Association, pp. 9–9.

[4] Bing, M., and Erickson, C. Extending unix system logging with sharp. In *Proceedings of the 14th USENIX Conference on System Administration* (Berkeley, CA, USA, 2000), LISA '00, USENIX Association, pp. 101–108.

[5] Bison gnu parser generator manual. Free Software Foundation http://http://www.gnu.org/software/bison/manual/bison.html, 2013.

[6] Chomsky, N. Three models for the description of language. *Information Theory, IRE Transactions on 2*, 3 (1956), 113–124.

[7] Davtyan, S., Kentros, S., Kiayias, A., Michel, L., Nicolaou, N. C., Russell, A., See, A., Shashidhar, N., and Shvartsman, A. A. Pre-election testing and post-election audit of optical scan voting terminal memory cards. In *Proceedings of the 2008 USENIX/ACCURATE Electronic Voting Workshop (EVT 08), July 28-29, 2008, San Jose, CA, USA* (2008).

[8] Detecting security incidents using windows workstation event logs. SANS Information, Network, Computer Security Training, Research, Resources https://www.sans.org/reading-room/whitepapers/logging/detecting-security-incidents-windows-workstation-event-logs-34262, 2013.

[9] Etalle, S., Massacci, F., and Yautsiukhin, A. The meaning of logs. In *Proceedings of the 4th International Conference on Trust, Privacy and Security in Digital Business* (Berlin, Heidelberg, 2007), TrustBus'07, Springer-Verlag, pp. 145–154.

[10] Flex (the fast lexical analyzer). Sourceforge http://flex.sourceforge.net, 2008.

[11] Knuth, D. E. Semantics of context-free languages. *Theory of Computing Systems 2*, 2 (June 1968), 127–145.

[12] Marksense voting tabulator. Connecticut. Secretary of the State. http://www.ct.gov/sots/lib/sots/legislativeservices/regulations/12_opscanusereg.pdf, 2008.

[13] Martin-Löf, P. *Intuitionistic Type Theory*. Bibliopolis, Naples, 1984.

[14] Connecticut. Secretary of the State. Print.

[15] Rouillard, J. P. Refereed papers: Real-time log file analysis using the simple event correlator (sec). In *Proceedings of the 18th USENIX Conference on System Administration* (Berkeley, CA, USA, 2004), LISA '04, USENIX Association, pp. 133–150.

[16] Sandler, D., Derr, K., Crosby, S., and Wallach, D. S. Finding the evidence in tamper-evident logs. In *Proceedings of the 2008 Third International Workshop on Systematic Approaches to Digital Forensic Engineering* (Washington, DC, USA, 2008), SADFE '08, IEEE Computer Society, pp. 69–75.

[17] Slonneger, K., and Kurtz, B. *Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach*, 1st ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[18] Voting system audit log study. California Secretary of State, Debra Bowen http://www.sos.ca.gov/voting-systems/oversight/directives/audit-log-report.pdf, 2010.