Formal Analysis of Attacks for E-voting System

Komninet Weldemariam\(^1,3\)

\(^1\)Dept. of Eng. and Comp. Science
University of Trento
Trento, Italy
Email: sisai@fbk.eu

Richard A. Kemmerer\(^2\)

\(^2\)Dept. of Comp. Science
University of California
Santa Barbara, USA
Email: kemm@cs.ucsb.edu

Adolfo Villafiorita\(^3\)

\(^3\)eDemocracy Group
Fondazione Bruno Kessler
(fbk-irst) Trento, Italy
Email: adolfo@fbk.eu

Abstract—Recently, the use of formal methods to specify and verify properties of electronic voting (e-voting) systems, with particular interest in security, verifiability, and anonymity, is getting much attention. Formal specification and verification of such systems can greatly help to better understand the system requirements by thoroughly specifying and analyzing the underlying assumptions and security specific properties. Unfortunately, even though these systems have been formally verified to satisfy the desired system security requirements, they are still vulnerable to attack.

In this paper we extend a formal specification of the ES&S voting system by specifying attacks that have been shown to successfully compromise the system. We believe that performing such analysis is important for two reasons: first, it allows us to discover some missing critical requirements for the specification and/or assumptions that were not met. Second, it allows us to derive mitigation or counter-measure strategies when the system behaves differently than it should. We used the ASTRAL language for the specification, and the verification is performed using the PVS tool.

Keywords: astral, attack scenario, e-voting, formal methods, pvs, specification and verification.

I. INTRODUCTION

In spite of the potential advantages that electronic voting (e-voting) might bring to the polling station, such as accessibility for impaired people, improved accuracy, and faster operations, its adoption in various countries has been slow and/or the cause of great debates and controversies. One of the reasons is that e-voting machines are complex real-time embedded systems that are required to operate in a (possibly) hostile environment. Another and more relevant reason is the poor design and poor implementation of (some of) the systems currently deployed for elections in various countries. See, for instance, [1], [2], [3], [4], for related matters.

There are a number of ways in which the integrity and assurance of a complex and safety-critical system’s correct behavior with respect to a specification can be achieved. Among them, formal verification has been shown to improve the security and quality of complex systems (e.g., [5], [6], [7], [8]). These approaches allow designers to prove, test, or otherwise examine interesting properties of a complex process whose behavior is specified abstractly, and then interactively refine the behavioral specification to be as close to an implementation as appropriate for a given assurance level.

The use of such techniques for e-voting systems is discussed in [9], [10], [11], [12], [13]. This paper complements [13] where the authors formally specified the ES&S voting system using the ASTRAL [14] language. A number of critical requirements, as they are documented in the ES&S manuals, have also been specified and analyzed using PVS [15]. This paper focuses on the formal analysis of the attacks that were discussed in [3], [16]. We do so, by extending the original formal specification with a set of transition specifications that represent those attacks. Each transition corresponds to a particular threat action for the ES&S voting system. The ASTRAL Software Development Environment (SDE) was used to process the extended specification and to automatically generate proof obligations for the PVS analysis tool. If proved, these proof obligations would verify that the extended system meets the stated critical security requirements.

If all of the proof obligations were to be proved, then the system specification must be missing some critical security requirements, since the modeled attacks were already demonstrated to be successful [3], [16]. Therefore, it would be necessary to see what additional critical requirements are needed to disallow the threat actions and keep the extended specification from being proved. In contrast, not being able to prove the extended specification would indicate that one, or more, of the threat actions violates at least one critical security requirement. However, since we know that attacks composed of these threat actions have been used to successfully compromise the system, it also indicates that there must be an implementation error or an unsatisfied procedural assumption that results in the actual system or the environment not satisfying their respective formal specification.

This paper reports the formal modeling and analysis of attacks for the ES&S voting system. In Section II, we recall the main components of the ES&S system, outline the voting process for using this system, and introduce the attack scenarios that were successfully implemented to compromise the system. In Section III we discuss the formal specification of the system and some critical requirements.
that specify the desired security properties. In Section IV
the formal specification of the threat actions that extend the
original specification are presented. We discuss related work
in Section V and draw our conclusions in Section VI.

II. ELECTRONIC VOTING SYSTEMS

A. ES&S Voting System Components

For the purposes of this work (see [16] for a more detailed
and complete view), the ES&S voting system is composed of:

- **DRE**: Direct Recording Electronic voting machine,
called the iVotronic. It is equipped with a touch-screen
where the voter casts his/her votes. The information
shown by the touch-screen changes in real-time to
match the voter’s choices. The iVotronic also stores the
audit data.

- **RTAL**: Real-Time Audit Log Printer, which performs
the function of a VVPAT (Voter-Verified Paper Audit
Trail) for the ES&S system. It produces a paper-based
record of the choices selected by the voter. The RTAL
is plugged into the DRE and the paper record is
viewable by the voter. The voter’s choices are under a
transparent cover so that they cannot be modified other
than through the normal voting process.

- **PEB**: Personalized Electronic Ballot. This is a device
used by the pollworker to load a ballot, initialize the
next ballot, and collect tabulated data and audit informa-
tion. Each time a PEB is inserted, its authenticity is
checked by the DRE using a four-digit code (election
qualification code, EQC), which is assigned prior to
election day.

- **CFC**: Compact Flash Card. This device holds files too
large to fit in the PEB and also audit data. The card
must be present to open and close the polls. At poll
closing, the audit data is automatically dumped into
the card.

B. Voting Process for a DRE based System

In this section we present a high-level overview of running
an election using the ES&S system. We skip the description
of the election preparation phase (e.g., the pre-election day
operations checklist, obtaining an EQC code, qualifying
PEBs, and clearing and testing voter terminals). Instead,
we assume that these operations have been carefully and
correctly done at the central location.

Prior to opening the polls, a pollworker unpacks and sets
up the DRE and plugs in the RTAL printer and power cables.
Poll workers must also ensure that a properly programmed
CF card is installed before powering on the DRE.

A Master PEB is inserted into the terminal to load the
ballot and later to open the terminal for voting. The same
master PEB must be used to close the terminal after the
polls have closed. Removing the PEB changes the terminal’s
current mode to sleep mode.

Once the polls are opened, a pollworker initializes the
ballot for a qualified voter by inserting a supervisor PEB,
which can be the same Master PEB used to open the polls,
into the machine. The terminal mode changes from sleep
to pollworker mode, the EQC code of the PEB is checked,
and the ballot is initialized, provided that the EQC of the
PEB matches with the one the terminal is configured for.
The pollworker removes the supervisor PEB and leaves the
terminal for the voter.

After the ballot is activated, the machine takes the voter
through each contest. DRE machines automatically forbid
overvoting, but not undervoting. When a voter selects or
cancels a candidate for a particular contest, an appropriate
indication is displayed on the screen and is printed on the
RTAL record. If the voter selects a candidate, the RTAL
record is marked as “Selected” and scrolled out of sight;
otherwise, it is marked as “Canceled” and scrolled out of
sight. The voter is eventually given the opportunity to review
his ballot, and if the voter confirms it, the votes are recorded
to local storage. The process continues in this way for all
qualified voters.

C. Example: Selected Attack Scenarios

In this section we give a short overview of the attack
scenarios that were discussed for the ES&S system; a
more detailed description is given [16]. For the purpose of
this work we also sketched a sequence diagram for each
the corresponding attack scenario (see Figures 1 and 2).

1) **changing the vote for an unattentive voter**. In this
scenario, the voter proceeds with the normal voting
process and the attacker intercepts the process just
before the review ballot is displayed. The attacker
steals votes by assigning them to the candidate who
s/he desires to win. The modified vote is displayed
on the DRE review screen and the change is printed
on the RTAL tape. Because the voter is unattentive,
he/she will not look at the screen nor the RTAL.
However, if the voter does check the screen or the
printed output and discovers that an error has been
made, s/he can recast the vote and the attacker will
stop stealing votes for a period of time. Otherwise,
the attacker’s modification is stored locally upon the
voter’s confirmation (See Figure 1).

2) **changing the vote for a careful voter**. This scenario
assumes the voters carefully cast, confirm, and check
the corresponding screen and printout. However, they
are not familiar with all the details of how their
votes are printed on the RTAL tape. The attacker
does not intercept the normal voting process until
after the cast ballot and confirm screens have been
shown to the voter. At this point the attacker changes
the voter’s electronic ballot, and the RTAL prints the
modified selection. The RTAL then immediately prints
the summary information and the barcode.
3) **canceling/completing the vote for a fleeing voter.** In this scenario the attacker takes advantage of a fleeing voter, which is a voter who does not complete the voting procedure, by intercepting the call to the routine that enables a chirping sound, which alerts the pollworker that a voter has fled. There are two possible scenarios depending on the voter’s vote: i) If the fleeing voter voted against the attacker’s candidate, then the attacker does nothing and lets the chirping routine perform as it should. The pollworker then discards the ballot and there will be one less vote for the undesired candidate. ii) If the fleeing voter voted for the attacker’s candidate but s/he did not complete the voting process then the attacker completes the voting process. This results in another vote being cast for the attacker’s candidate.

4) **faking a fleeing voter to cancel a vote.** This attack scenario is similar to #3. However, in this case the attacker cancels the vote by making it look like the voter fled. In particular, if the voter did not choose the candidate that the attacker wants, the attacker intercepts the confirmation process and pretends to cast the ballot: the normal “thank you” screen is displayed, but nothing is printed on the RTAL tape. After some amount of time elapses (during which the voter most likely leaves the voting booth) the attacker directs the system to display the confirmation screen. Then after another reasonable amount of time has passed the attacker calls the chirping sound routine and the machine immediately starts chirping. A pollworker will think the voter was a fleeing voter, and the ballot will be discarded (see Figure 2).

---

In Ohio the votes of fleeing voters are discarded. Whereas, in California the pollworker casts these votes.

---

**III. SPECIFICATION OF THE ES&S SYSTEM**

**A. Modeling the ES&S System**

In [13] we treated the ES&S system as a complex, real-time embedded system, consisting of four components, which were discussed in Section II-A. We mapped each of these components to ASTRAL [14] process instances and we also specified security critical requirements to prove the correctness and integrity of the election results.

The complete ASTRAL specification is approximately 30 pages long. In this paper we discuss only the parts of the specification that are relevant to this work. In particular, the part that allows us to extend the specification in order to incorporate the attack scenarios that we discussed previously. Below is an example of a process declaration in the ASTRAL language:

```
PROCESSES
  the_DRE: array [1..Number_Of_DRE] of DRE_Process [...],
```

which expresses that there are Number_Of_DRE of DRE process instances. While, the following shows a snippet of a global type declaration.

```
TYPE
  Races IS SET OF Race,
  Candidates IS SET OF Candidate,
  Ballot IS SET OF Race_Candidate_Pair,
  Decision IS (Selected, Cancelled), [...]
```

The total number of votes for a particular candidate in a race are modeled with the following variable within the DRE process:

```
VARIABLE
  TotalTallyCount(Candidate, Race) : Non_Negative
```
The communication between the DRE and the RTAL processes is modeled by the variables:

VARIABLE
Signal_Enabled: Boolean,
Which_Signal: SignalType

where the first variable signals that the DRE is sending information to the printer and Which_Signal carries the kind of information to be printed (e.g., is the print information a start vote session message or a vote selection).

When a voter casts a vote, s/he is actually interacting with the system by navigating from one screen to another using an appropriate button (such as, NEXT or BACK). We model such interaction by assigning an integer number to each screen shown to the voter and by defining specification functions that take as input a screen number and return the information to be displayed and the buttons available.

The variable Display(Screen_Number) of type screen is used to hold the state of the screen as it is to be shown to the voter while s/he is voting. For example, if the voter is in one of the race screens then the value of the Display contains the candidates of that race with appropriate button(s) displayed on it.

An ES&S DRE requires a pollworker to insert a qualified PEB device in order to allow various operations to run the election. These operations include loading the appropriate ballot, opening or closing polls, initializing the ballot, collecting election results, and performing various administrative tasks. These aspects are modeled by ASTRAL transition specifications. Note that in ASTRAL, a transition is modeled by entry and exit conditions, and a non zero duration is assigned to each entry/exit pair. Transitions are executed as soon as the entry conditions are satisfied assuming no other transition for that process instance is executing.

The following snippet specification encodes the ballot loading operation prior to start election.

TRANSITION Insert_PEB ( p: PEB_ID )
ENTRY [ TIME : I_P_Dur1 ]
MachineTurnedOn & *PEB_Inserted
& Stored_EQC = p.Secret_EQC
& p.Kind = Master
& Which_Phase = Pre_Voting
& DRE_State = Initial_State
& *Ballot_Loaded [...] EXIT
FORALL R: Race {
  Race_Candidates { R } = P.Candidates_Of_Race { R } }
& Ballot_Loaded
& DRE_State = Opening [...]

A voter navigates from one screen to another and performs a selection by executing the transitions Push_Button and Make_Selection, respectively. The transitions for each of the processes are specified similarly to Insert_PEB (see [13] for more details).

B. Critical Security Requirements

Once we specify the relevant information of the system model we need to specify what security requirements the system should meet given the assumptions about the behavior of the system and the external environment that interacts with the system.

In particular, we specified the following concerns:

1) Environmental/Procedural Assumptions. We have specified a number of behaviors about the external environment that the e-voting system relies on. For instance, the behavior of the people (voters, pollworkers, and election officials) who interact with the system. This is outside the ES&S system, but it influences how the system operates.

2) Security Requirements. We have specified approximately 25 critical requirements (expressed as invariants, constraints, and schedules) that must be satisfied by the system given all the possible assumptions about the environment. In the ES&S system, for instance, the DRE should correctly handle vote selection and the RTAL should update the paper tape after the voter pushes the start button, makes a selection, confirms a vote, or when the poll worker rejects the ballot of a fleeing voter.

Below we show examples of critical requirements expressed as invariant and constraint specifications using ASTRAL.

Specifying the integrity of the election results: In this example, we want to guarantee that, after the election is closed, the results downloaded into the master PEB must be equal to the sum of the results collected from each DRE. The property is specified in the global invariant clause as

EXISTS p: PEB_Number
{ the_PEB[p].Kind = Master
  & FORALL d: DRE_Number
  { the_PEB[p].ResultDownload_Completed
    & the_DRE[d].Which_Phase = Post_Voting
    & the_DRE[d].DRE_State = Closed
    & FORALL C: Candidate, R: Race (C ISIN
      the_DRE [d].Race_Candidates { R }
      + the_PEB[p].tabulatedData
        (C, R, the_DRE [d].Self )
      = the_DRE [d].TotalTallyCount(C,R) )))

Specifying that all voter selects/cancels are displayed on the DRE screen: The following constraint expresses the fact that when a voter selects or cancels a candidate C for a given race R, the DRE screen must be updated accordingly:

FORALL C: Candidate, R: Race
  { Fill ( Picked’ ( Candidate_Name { R } , Race_Title { R } )) = UnMarked
    & Fill ( Picked ( Candidate_Name { C } , Race_Title { R } )) = Marked

29
Table I

<table>
<thead>
<tr>
<th>Requirements (obligations)</th>
<th>After Splitting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invar, Const, Sched</td>
<td>Invar, Const, Sched</td>
</tr>
<tr>
<td>DRE</td>
<td>4, 6, 1</td>
</tr>
<tr>
<td>RTAL</td>
<td>1, 1, 3</td>
</tr>
<tr>
<td>PEB</td>
<td>1, 0, 1</td>
</tr>
<tr>
<td>CF Card</td>
<td>0, 0, 1</td>
</tr>
<tr>
<td>Global</td>
<td>6, 0, NA</td>
</tr>
<tr>
<td>Total</td>
<td>12, 7, 6</td>
</tr>
</tbody>
</table>

& Display (scrNumber’) -=
  - Display’ (scrNumber’)
  -> Display (scrNumber’) =
  Update (Display’ (scrNumber’),
    Candidate_Name (C), Marked)
& FORALL C: Candidate, R: Race
  { Fill (Picked {Candidate_Name (C),
              Race_Title (R)}) } = UnMarked
& Display’ (Candidate_Name (C),
    Race_Title (R)) = Marked
& Display (scrNumber’) -=
  - Display’ (scrNumber’)
  -> Display (scrNumber’) =
  Update (Display’ (scrNumber’),
    Candidate_Name (C), UnMarked)

C. PVS Verification

The specification was first constructed and type-checked using the ASTRAL SDE [17]. We validated the specification and generated the corresponding proof-obligations for the critical requirements. Moreover, the specification was automatically translated into its PVS [15] counterpart using the ASTRAL SDE, which enabled the specification to be passed to the PVS theorem prover for verification.

Before invoking the theorem prover, the ASTRAL split engine was used to split and classify the ASTRAL specification into collections of simpler properties that infer the whole clause so that the proof of each property could be tackled separately. Table I shows the number of invariants, schedules, and constraints for each of the four processes and the global invariants. It also shows the number after they are split by the ASLAN SDE.

Using the PVS interactive theorem prover and the techniques discussed in [18], we have proved many critical requirements for the system (mostly local invariants and constraints). More specifically, we have proved 13 of the 22 invariants, 3 of the 8 schedules, and 7 of the 10 constraints. We expect that the other global and local properties can be proved using the same or similar proof techniques and strategies.

IV. Extending the System Specification by Modeling Attack Scenarios

We model the attack scenarios presented in Section II-C in terms of threat actions expressed as ASTRAL transition specifications. The system model is extended by augmenting the specification with new possible states that are the result of the execution of the threat actions.

In particular, the model extension is performed as follows:

- we define new types, variables, and constants. There are two kinds of variables that we declare: those that provide additional information about the state of the system (e.g., the system is now about to display the review ballot) and those that hold information about the successful execution of a threat action (e.g., a fleeing voter has been faked).
- a transition is defined for each threat action, which is part of a given attack scenario. Note that an attack scenario can be implemented using one or more threat actions.
- a transition may be split into two or more transitions, or a transition may be extended with more information (e.g., the Print_Selection transition in the Appendix is extended to print the effect of threat actions) to specify the attack scenario.

We assume that the attacker can intercept the normal voting process at any point. For instance, if s/he intercepts the process before the review screen is displayed and the attack is successful, then the tempVoteRecord variable should include the maliciously modified candidate and the Display variable should update the screen accordingly. It is, in fact, the voter’s task to correctly verify that what is displayed exactly matches her/his preferences.

To represent the various kinds of voters (unattentive, careful, and fleeing), we introduced the following global type

\[
\text{VoterType} : (\text{unattentive, careful, fleeing})
\]

In addition, the variables

\[
\text{vote_changed, Fleeting_Faked: Boolean, attPickedName : Name}
\]

are declared, to, respectively, hold information about whether the voter’s vote is changed, whether the fleeing voter is faked, and the name of the attacker’s candidate. In addition, information about where the attacker intercepts the process to start the threat action is encoded by the variables

\[
\text{review_displayed and summary_sent2RTAL}
\]

When an attacker changes or cancels a vote, it is actually performing a sequence of interactions with the DRE process in order to fulfill the threat action. The successful completion of such an action eventually assigns new values to some of the exported variables above.

The following transition specifies the change vote threat action, which appears in the sequence diagram depicted in Figure 1.
TRANSITION Attack_Change_Vote(
    vc, ac: Candidate, vType: VoterType)
ENTRY [TIME ACV_Dur]
Which_Phase = During_Voting
& Terminal_Mode = voter_mode
& vType = Unattentive
& EXISTS R: Race 
    vc ISIN Displayed_Candidates ( R )
    & ac ISIN Displayed_Candidates ( R )
    & vc ISIN tempVoteRecord ( R )
    & Picked(Candidate_Name(vc),
               Race_Title(R))
    & Picked(Candidate_Name(ac),
               Race_Title(R))
& vc -= ac
& EXISTS b: Button ( b = REVIEW
    & Button_Pushed ( b ) )
& scrName = REVIEW_SCREEN
& 'Review_Displayed
& 'Vote_Changed
EXIT
EXISTS R: Race 
    vc ISIN Displayed_Candidates’ ( R )
    & ac ISIN Displayed_Candidates’ ( R )
    & vc ISIN tempVoteRecord’ ( R )
    & tempVoteRecord ( R ) BECOMES
    (tempVoteRecord’ ( R ); SET_DIFF (vc )
    UNION ( ac )
    & Picked ( Candidate_Name ( vc ),
               Race_Title ( R ) ) = FALSE
    & Picked ( Candidate_Name ( ac ),
               Race_Title ( R ) ) = TRUE
    & FORALL CN:Name, R1:Title ( 
        { ( CN -= Candidate_Name(vc)
            & CN -= Candidate_Name(ac) )
        | R1 -= Race_Title ( R ) )
    ->
    Picked(CN,R1) = Picked’(CN,R1)
    & currentRace = R
    & Signal_Enabled
    & Which_Signal = Vote_Signal
    & pickedName = Candidate_Name ( vc )
    & attPickedName = Candidate_Name ( ac )
    & Vote_Changed

The enabling condition for this threat action (i.e., for the transition) specifies that the fleeing voter is voting during election period in the voter's terminal mode, that there exists a race R such that the voter's candidate vc is in the displayed candidates list for race R for which the voter already voted, that the attacker's candidate ac is also a legitimate candidate contained in the displayed list for the same race R and it is not selected by the voter, and the voter desire is different from the attacker (i.e., vc ∼= ac). In addition the voter has already requested the review screen, currently there is nothing shown on the REVIEW_SCREEN, and there is no change of vote at the moment.

After the threat action is successfully executed (i.e., after the transition is ended) the following holds: the voter's selection contained in the tempVoteRecord now contains the attacker's choice, the picked value is true for ac and is false for vc. In addition, the exported variables currentRace,pickedName,attPickedName, pickedValue and Which_Signal have new values, and the signaling variable is true. This indicates that the RTAL can now print the modification expressed in these exported variables. The RTAL process prints this information by executing the Print_Selection transition (the details of this transition can be found in the Appendix).

The above modification, which is contained in the tempVoteRecord variable, is also displayed on the review screen. It is worth noting that both the review screen and what is printed on the RTAL tape report the modified selection, rather than the original one. From an attacker's point of view, it is better to keep the display and tape consistent, because, if an abnormality is detected, then it is more likely to be attributed to a display miscalibration rather than to an attack. It is possible that the voter will detect such a change. In this case, the voter can recast his/her vote by calling the Push_Button and Make_Selection transitions.

The Faking a Fleeing Voter attack is an example of a scenario that requires several threat actions. The canceling of votes by fake a fleeing voter has three threat actions (as depicted in the sequence diagram, see Figure 2). The threat actions are specified as three transitions in the DRE process:

1) Attack_Change_Vote: this is an except/exit transition 2 that specifies the fact that an attacker fakes a fleeing voter by pretending to complete the voting process on her/his behalf. The exit assertion of this transition will set the variable Fleeing_Faked to true.

2) Attack_ReDisplay: this transition specifies the fact that after some delay (during which time the voter leaves the booth) since the voter is successfully fooled, the attacker directs the DRE to display the confirmation page again.

3) Attack_Call_ChirpingRt: this specifies that after the voter has been fooled and DelayTime has passed, the attacker resumes the normal voting process by calling the chirping routine. This results in the pollworker taking action according to the prescribed procedure. (This transition is only enabled after the first two transitions have been executed.)

Due to space limitations, we are not able to present the formal specifications for these threat actions, which are specified similarly to the ones described earlier.

After extending the system specification with the threat actions, the ASTRAL SDE is used to generate the proof obligations for the extended specification. Because there are

2In ASTRAL, specification exceptions are handled explicitly by adding except/exit pairs in addition to the normal entry/exit pairs [14].
additional transitions, there are more proofs to be done. In addition, because some of the original transitions were split and/or extended, the corresponding proof obligations must be reproved.

We have just begun the verification process for the extended specification. We started with the proof obligations that were unchanged to assure that they are still valid. So far, we have reproved 3 of the 13 invariants and 2 of the 7 constraints. When we started proving the proof obligations for the original system specification we were slowed down due to changes in the PVS theorem prover. We have now updated the ASTRAL environment to be consistent with the latest version of PVS, and we plan to have all of the properties proved in the near future.

V. RELATED WORK

Work to rigorously define e-voting properties, attack models, and languages for describing counter-measures is still preliminary. The application and usage of verification methods and tools for these systems is even more limited. In what follows we mention some attempts to improve the current design and development of e-voting systems both from procedural and system perspectives.

Related to the procedural aspects work, such as [19, [20] suggest possible improvements to existing procedures, and [21], [22], [23] introduce techniques to formally analyze what security breaches may be derived by executing the procedures in the wrong way.

Assessments of some existing e-voting systems have been done [3], [4], [24]. These primarily focus on the design and implementation flaws, which could be exploited to compromise and invalidate elections. The authors also suggest a drastic change in the way in which e-voting systems are designed, developed, and tested.

The use of formal methods in the specification and verification of e-voting systems is relatively new. We mention [9], [10], that present the formal specification and verification of an e-voting protocol using Pi calculus, [11] that presents a tool, FSMC+, that has been used to specify and verify the control logic of an e-voting machine, and [12], where the authors present a mobile implementation of an e-voting system and used formal verification techniques to validate the security property of their proposed system.

VI. CONCLUSION AND FUTURE WORK

In this paper we showed how formal verification techniques can be used to model and reason about the security of e-voting system. In particular, we presented the extension of the specification (see [13]) for the ES&S system to capture attack scenarios, as discussed [3], [24], [16].

We believe that besides analyzing the system against its requirements, it is equally important to perform an analysis under malicious circumstances where the system execution model is enriched with attack behavior. This is helpful in order to detect missing requirements or unwarranted assumptions about the specification we developed. In addition, this allows us to sketch counter-measure strategies to be used when the system behaves differently than it should and to build confidence about the system under development.

The threat actions that we specified in the extended system were those needed to model the specific scenarios presented in [16]. They are a minimal sampling of the possible threat actions, but they demonstrate the approach on a real system. In the future we would like to model more general threat actions to see if new attack scenarios or missing critical requirements can be identified.

So far we managed to formally verify that the specification satisfies many of the critical requirements that we discussed in this paper, mostly the local invariants and constraints. The proofs were achieved by following the techniques presented in [18]. For instance, we applied the try-untimed and try-untimed-con proof strategies to prove some of the local invariants and constraints of the system. In general, the proof is carried out first by splitting the critical requirements and applying the appropriate proof strategies developed to support the analysis of ASTRAL specifications (see [18]).

The success of the next generation of e-voting machines depends upon being able to capitalize from the lessons learned by using and analyzing the systems currently deployed. The work we have presented in this paper is one way in which we can get a better understanding of the strengths and the weaknesses of existing systems and thus lay the foundations for engineering and deploying a new generation of more secure and robust technologies for polling stations.

REFERENCES


VII. APPENDIX

The following transition specifies how the RTAL prints the information sent by the DRE process. Its entry condition specifies that the DRE is connected to the RTAL, the DRE has sent a signal to the RTAL for printing, the send signal should be in one of the permitted voting procedures (i.e., My_DRE.Which_Signal = NoSignal), and the RTAL is currently in waiting state. The exit specifies a complex conditional. It is related to the normal voting process as carried out by legitimate voters (careful or attentive voters) and the altered voting process due to the various attacks as discussed in the paper.

TRANSITION Print_Selection
ENTRY [ TIME : P_Dur ]
My_DRE.Plugged_In
& My_DRE.Signal_Enabled
& RTAL_State = Wait
& My_DRE.Which_Signal = NoSignal
EXIT
RTAL_State = Printed & IF My_DRE.Which_Signal = Start_Signal
| My_DRE.Which_Signal = Vote_Signal
THEN
IF My_DRE.Which_Signal = Start_Signal
THEN
tapePosition = tapePosition’ + 1
& CutLengthCounter = CutLengthCounter’+1

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on July 09,2010 at 08:30:11 UTC from IEEE Xplore. Restrictions apply.
& voterNumber = voterNumber’ + 1
& Tape (tapePosition) BECOMES
  Make_Print_Info (My_DRE.RTALMessage)
ELSE IF My_DRE.Vote_Changed
  THEN
  //Attack point. */
  tapePosition = tapePosition’ + 2
  & CutLengthCounter = CutLengthCounter’ + 2
  //This prints the cancelation of the voter’s candidate choice. */
  & Tape (tapePosition – 1) =
    Make_Print_VoteEntry{
      My_DRE.pickedName
      ,My_DRE.currentRace
      ,Cancelled
    }
  //This prints the selection of the attacker’s candidate selection. */
  & Tape (tapePosition) –
    Make_Print_VoteEntry{
      My_DRE.attPickedName
      ,My_DRE.currentRace
      ,Selected
    }
  & FORALL i: Tape_Number
    (i ≠ tapePosition & i ≠ tapePosition – 1)
    -> NOCHANGE (Tape (i))
  ELSE
    //Normal voter’s choice print. */
    tapePosition = tapePosition’ + 1
    & CutLengthCounter = CutLengthCounter’ + 1
    & Tape (tapePosition) BECOMES
      Make_Print_VoteEntry{
        My_DRE.pickedName
        ,My_DRE.currentRace
        ,My_DRE.pickedValue
      }
    FI
  ELSE IF
    My_DRE.Vote_Changed
    THEN
    //If attacked before the summary has printed. This is the case for careful voter. */
    tapePosition = tapePosition’ + 5
    & CutLengthCounter = CutLengthCounter’ + 5
    & Tape (tapePosition – 4) =
      Make_Print_VoteEntry{
        My_DRE.pickedName
        ,My_DRE.currentRace
        ,Cancelled
      }
    //This prints the selection of the attacker’s candidate selection. */
    & Tape (tapePosition) =
      Make_Print_VoteEntry{
        My_DRE.attPickedName
        ,My_DRE.currentRace
        ,Selected
      }
    //Followed by immediate print of the summary information and barcode. */
    & Tape (tapePosition – 2) =
      Make_Print_Info{
        My_DRE.RTALMessage
      }
    & Tape (tapePosition – 1) =
      Make_Print_Undervote{
        My_DRE.underVotedRaces
      }
    & Tape (tapePosition) =
      Make_Print_BallotBarcode{
        My_DRE.BallotBarcode
      }
    & FORALL i: Tape_Number
      (i ≠ tapePosition & i ≠ tapePosition – 1 &
      i ≠ tapePosition – 2)
      -> NOCHANGE (Tape (i))
    & VoteStartPosition (voterNumber) BECOMES tapePosition
    & VoteEndPosition (voterNumber) BECOMES tapePosition
    & summaryPrinted = TRUE
    ELSE
    tapePosition = tapePosition’ + 3
    & CutLengthCounter = CutLengthCounter’ + 3
    & Tape (tapePosition) =
      Make_Print_Info{
        My_DRE.RTALMessage
      }
    & Tape (tapePosition – 1) =
      Make_Print_Undervote{
        My_DRE.underVotedRaces
      }
    & Tape (tapePosition) =
      Make_Print_BallotBarcode{
        My_DRE.BallotBarcode
      }
    & FORALL i: Tape_Number
      (i ≠ tapePosition &
      i ≠ tapePosition – 1 & i ≠ tapePosition – 2)
      -> NOCHANGE (Tape (i))
    & VoteStartPosition (voterNumber) BECOMES tapePosition
    & VoteEndPosition (voterNumber) BECOMES tapePosition
    & summaryPrinted = TRUE
    FI
  FI
ENDIF