Extending the Functionality of the VoTeR Audit Stations to Perform Transitive Audits

Elizabeth Muirhead

Abstract— **To check the validity of an election outcome, the state of Connecticut performs audits. An audit examines some portion of the ballots to determine if the apparent election result is correct or if the state needs to perform a full recount. At the University of Connecticut in the Center for Voting Technology Research (VoTeR), auditors perform audits by scanning and reinterpreting every ballot from randomly selected voting precincts to produce cast voting records (CVRs). Auditors are then responsible for performing a second audit on those results called a transitive audit. A transitive audit is a type of risk-limiting audit, and it can catch more malicious threats to election security. Risk-limiting is based on the notion that you cannot know the true election outcome without a recount, but you can minimize the chance of an undetected false result. The following paper describes an application that helps auditors interpret and browse the results from the original audit and perform the transitive audit. By implementing an independent piece of software into the audit process, we add another layer of protection and safety to our elections.**

I. INTRODUCTION

To make sure that public elections in Connecticut accurately reflect the outcome, we implement practices at all stages of the process that allow us to better trust and monitor elections. The state of Connecticut requires that audits are performed on a randomly selected 5% of voting precincts [1]. This law underwent change in 2016; prior to then, it was required that 10% of precincts be audited [2]. Generally speaking, an audit is the process of manually examining some portion the ballots to compare results [3][4]. To perform an audit, auditors need a valid, reliable audit trail [3]; the audit trail is comprised of the original records of the votes, which in most states, including Connecticut, are paper ballots. Another important feature of an audit is the use of independent equipment; the independence helps to maintain faith in the quality of the audit. If the original voting equipment or software were tempered with or malfunctioned, it is reasonable to expect that, an independent check would catch it [2]. Finally, it is important that the audit produces some sort of immutable records that can be further checked by a transitive audit.

A transitive audit is the process of checking the results of an audit, often through semi-automated techniques [4][5]. At the University of Connecticut Center for Voting Technology and Research (VoTeR), they perform ballot-level comparison audits by using their equipment and software to interpret each individual ballot and commit to a record of it called a cast voting record (CVR). A transitive audit examines these records to ensure that the software from the first audit is interpreting the ballots correctly. When auditors perform the transitive audit, they are performing a type of risk-limiting audit [5][6]. The notion of risk-limiting is related to the fact that you cannot be positive your results are correct unless you perform an entire recount. When you perform a risk-limiting audit, you are just trying to minimize the risk that if there was an incorrect result, you would not catch it [3][7].

As described in [2], when a district in Connecticut is audited, VoTeR configures an audit station, which is the process of setting up the equipment to read the particular ballots from that precinct. Auditors are then responsible for dividing the ballots into batches and scanning the batches one ballot at a time; the audit station generates its own interpretation of the ballot (the CRV). The audit station generates a cover sheet for each batch, which lists the totals from that batch, including the definitive votes and the ambiguous votes. Particularly in cases where there are ambiguous votes, auditors may want to scan a single batch multiple times. The audit station may mark some of the votes differently each time it scans, and auditors should scan the batch multiple times to achieve

stable totals. Once a batch has been reviewed and committed to, the totals from that batch are added to the election totals. After all of the batches have been read as many times as necessary and committed to, the audit station produces the final CVRs and final tally.

The application described in this paper is an extension of the audit station; it uses the final CVRs produced by the first audit, the election definition provided by the state, and an image of the ballot. The election definition is an Extensible Markup Language (XML) file that contains information about the structure of the physical ballot. The strength of the application is in its independence; it is separate from the other software in the lab. It takes in three, trusted files. Since it does not use legacy code or take input from untrustworthy sources, the application can be trusted by auditors; independence propagates our confidence in the audit process.

Additionally, since the code is fairly short, this allows for the possibility of outside programmers to check it. If auditors or the public were wary of the application, an independent programmer could easily read the code and verify that it is not malicious.

In the following sections of the paper, we attempt to provide as much information as is necessary to understand the design and functionality of our application.

II. OVERVIEW OF APPLICATION

This application is written in Python. The characteristics of Python fit our goals for the project. Python is extremely readable syntactically, making it easy for programmers who are not familiar with the application to review it. Additionally, it is type-safe, making it more secure and less prone to buffer-overflows.

The application needs three pieces of input data includes the encrypted CVRs, the election definition, and an image of the blank ballot.

The goal of this application was to design something that made the transitive audit process faster and easier, offer alternate ways to view and represent the CVRs, and conduct the transitive audit with auditors still maintaining a reasonable level of control.

When the auditors run the application, the first thing that they see is the following screen, which contains three buttons:

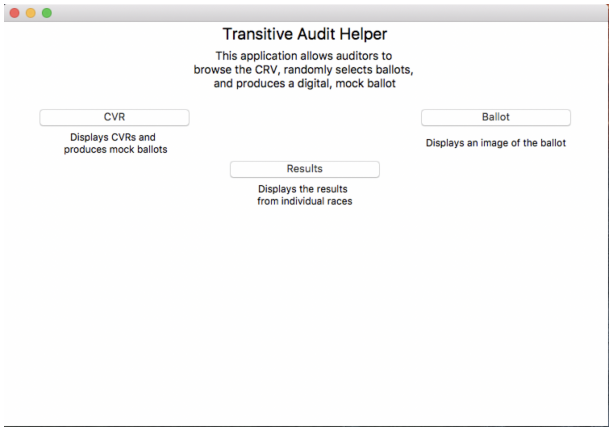


Figure 1: Home screen of application

Selecting the button that says “Ballot” displays a blank image of a ballot.

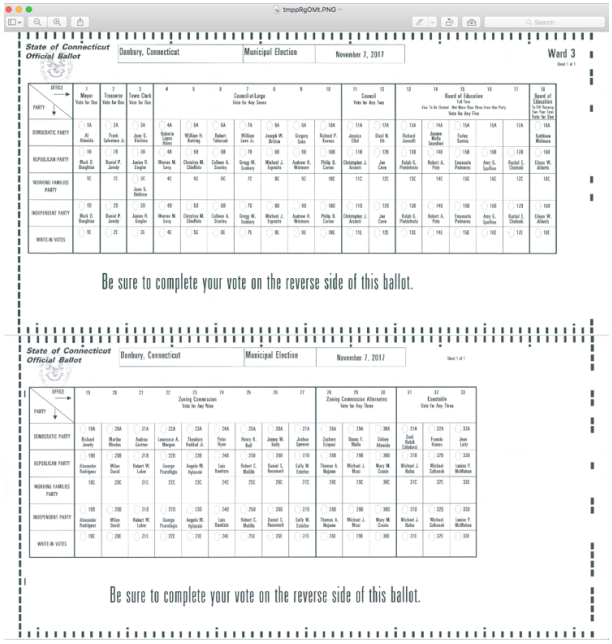


Figure 2: Blank image of a ballot

Selecting the button that says “Results” brings you to this screen:

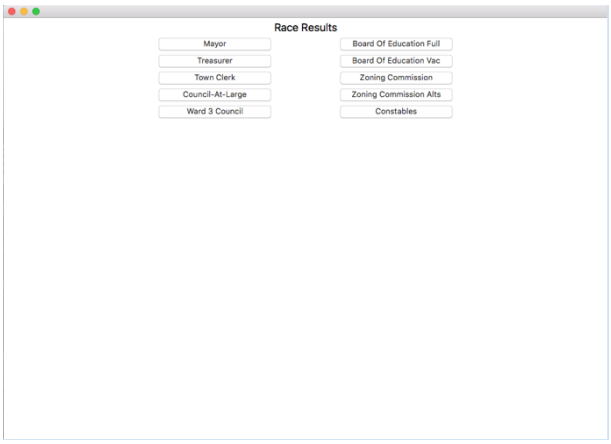
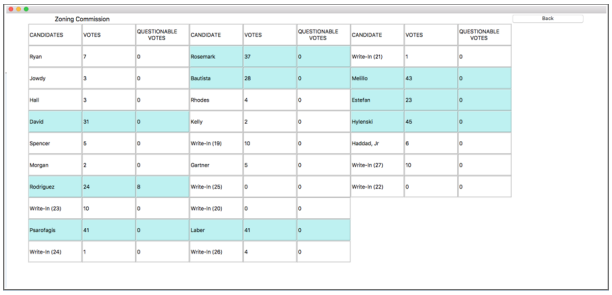


Figure 3: Results screen

From there, auditors can view the number of total votes and questionable votes that each candidate received. The application also takes into account the number of winners that a race is supposed to have and highlights them in blue. If there is a tie for a seat, it is highlighted in red.



CANDIDATES	VOTES	QUESTIONABLE VOTES	CANDIDATE	VOTES	QUESTIONABLE VOTES	CANDIDATE	VOTES	QUESTIONABLE VOTES
Ryan	7	0	Rosemark	37	0	Write-in (21)	1	0
Jowdy	3	0	Bautista	28	0	Melillo	43	0
Hall	3	0	Rhodes	4	0	Estefan	23	0
David	31	0	Kelly	2	0	Hylenski	45	0
Spencer	5	0	Write-in (19)	10	0	Haddad, Jr	6	0
Morgan	2	0	Gartner	5	0	Write-in (27)	10	0
Rodriguez	24	8	Write-in (25)	0	0	Write-in (22)	0	0
Write-in (23)	10	0	Write-in (20)	0	0			
Psarofagis	41	0	Laber	41	0			
Write-in (24)	1	0	Write-in (26)	4	0			

Figure 4: Results from a race with nine winners

The final button 'CVR' displays the CVRs in an excel-like table, making it navigable and easy for auditors to use. Our application offers two additional features: producing mock ballots and preforming transitive audits.

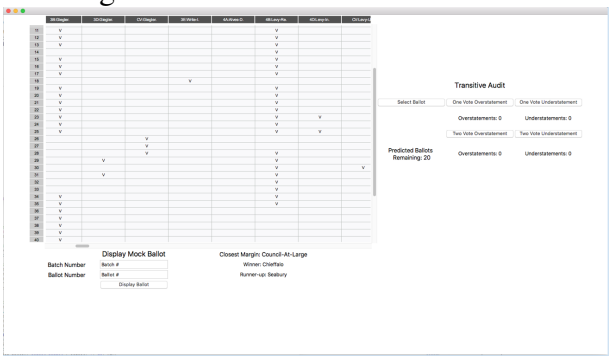


Figure 5: This screen displays the CVRs in a table and gives the auditors access to the mock-ballot and audit feature

The auditors can input the batch and ballot number of a desired ballot, and the application will access the corresponding CVR and generate a representation of what that ballot should look like. It indicates definite votes with black circles and ambiguous votes with blue semicircles. This feature utilizes the CVR data and the bubble locations in the election definition.

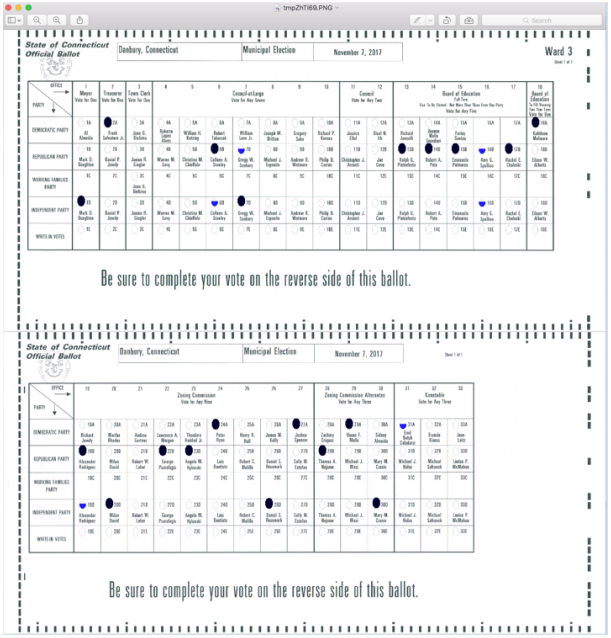


Figure 6: Mock-ballot produced by application using CRV information

There are, however, three special cases we identified that cannot be accurately represented on the mock ballot due to the limited detail in the CVRs.

Definite Over-Votes

The CRVs to not indicate when a ballot contains an over-vote for a race. If a ballot contains more votes than allowed for a particular race, none of the votes in that race are reflected in the CVR.

Possible Over-Votes

In situations where the combined total of definite votes and ambiguous votes would be an over-vote, the CVR does not reflect any of the votes.

Cross-Votes

When a candidate is endorsed by multiple parties in a single race, there is the possibility for a cross-vote. If the voter fills in multiple bubbles for a single candidate in a race, it is a cross-vote. It is

counted as a single vote for that candidate, and it is indicated in its own column on the CVR. In a situation where a candidate is endorsed by two parties, the program fills in both bubbles. If, however, there were a candidate who was endorsed by three or more parties, the CVR is not nuanced enough to show which combination of bubbles are filled in. It just shows that at least two bubbles were filled in, and every bubble for the candidate would be filled in on our mock ballot.

In addition to the mock ballot, this part of the application also has tools that help auditors conduct the transitive audit. The following image displays this portion of the application.

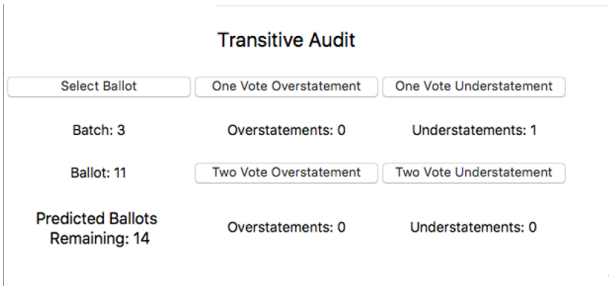


Figure 7: The transitive audit feature, which allows auditors to select ballots and report over and understatements

The application calculates the number of ballots necessary to audit using an equation for a ballot-level comparison audit with a 10% risk limit from [3].

$$n \leq 4.8 + \frac{1.4(o_1 + 5o_2 - 0.6u_1 - 4.4u_2)}{m}$$

In the equation, n represents the number of ballots audited. o_1 is the number of overstatements, and u_1 is the number of understatements. An understatement is when auditors change the voting system interpretation in a way that increases the margin(s) between the winner and the runner-up. Conversely, an overstatement is when the auditors change the interpretation to decrease the margin between the winner and runner-up. u_2 represents the number of two-vote understatements, which is when a ballot contains an error that, when corrected by auditors, increases the margin by two votes. o_2 is the number of two-vote overstatements, when auditors change the interpretation to narrow the margin by two votes.

Auditors can utilize the "Select Ballot" button to randomly select a ballot for the audit. They would then be responsible for comparing the appropriate physical ballot to either the CRV or the mock ballot. Auditors would be responsible for reporting overstatements and understatements, and the application is capable of recalculating the number of necessary ballots. This feature is advantageous compared to current methods of randomly selecting and calculating the risk limit outside of the application. Our application adds a level of ease and usability, but auditors still maintain control of the process and are responsible for spot checking the ballots.

III. STRUCTURE OF CODE

The application uses four python files; each is described in more depth below.

parse_data.py

The parse_data.py script contains two classes; one treats the csv file containing the CVRs as an object called CVR, and the other treats the election definition as an object called ElectionDef. Between the CVRs and the election definition, the application has all of the information necessary to tally and display the votes. The CVR class builds ordered lists that contain the names, parties, and number of votes for each candidate.

The ElectionDef class builds a list of trees. The election definition itself is read in as an ElementTree. The class contains methods that traverse the tree, parse out the information as needed, and build a new list of trees using the Python anytree package. Each individual race is treated as its own tree; see Figure 8. Each bubble on the ballot corresponds to a different branch. Each of these branches contains the candidates name, party, and bubble location. The reason for this is that a single candidate can be endorsed by multiple parties, and different candidates with the same name may be running in different races. The trees preserve the same amount of individuality as the actual bubbles on the ballot.

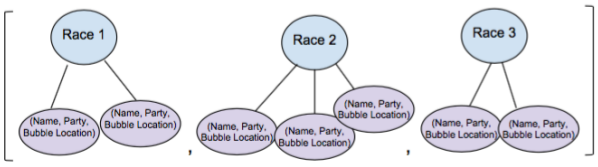


Figure 8: A list of trees where each tree contains a branch for each bubble on the ballot

window_manager.py

The second Python script is called window_manager.py. It contains two separate classes; one creates a window using the Python Imaging Library, PIL [8]. We selected this package because it has the ability to display Tagged Image File Format (TIF) files; images of the ballots are stored in TIF files.

The other class in the window_manager script creates a tkinter window. The tkinter package has features that allow us to create, buttons, labels, and tables, which is useful for achieving the features of the application [9].

audit.py

This file contains an audit class that creates an instance of an audit using the smallest race margin and the total number of ballots. The class contains methods that update the overstatement and understatement values. It also calculates the number of ballots needed for the audit and updates the value as ballots are selected for the audit.

application.py

The final python script, called application.py, is the script that runs the application and calls the desired classes and methods from the other three scripts. This script is written in a procedural style. The main function creates a tkinter window that contains buttons to perform the desired actions.

IV. AREAS FOR FURTHER DEVELOPMENT

This application is somewhat limited by the input information that it has access to. For example, the CVRs do not contain enough nuance to show over-votes or explicitly define cross-votes. If the CVRs recorded these special cases, the functionality of the application could be expanded to create mock-ballots that are accurate in all special cases.

On a more pressing note related to security, the application should put more emphasis on verifying the input data. The application uses the CVRs directly, after they have already been decrypted. To improve the security of the input data, the application should access the encrypted CVRs and run the script that decrypts them. Presumably, the application would only be able to decrypt the file, if it had been encrypted by the software from the VoTeR lab. The decryption step should be build into the software. In continuing the development of this project, more effort should be put into understanding the encryption that the audit station uses and possibly utilizing digital signatures to ensure that the input data is from the lab.

Additionally, all of the input data is hard coded into the current version of the application. In order to make it more usable, the input data should be passed in at run-time. Auditors should not be responsible for changing the source code to accommodate the input for their particular audit, nor should they be able to access the source code and change it.

V. CONCLUSIONS

Our application uses a semi-automated approach to expedite the transitive audit. Auditors will still be in control and be required to review some the physical audit trail. One of the key features that ensures the reliability of this application is its independence. Since the code is independent, short, and written in python, a programmer outside the lab should be able to easily review it and verify that it is trustworthy.

Ensuring that the audit process is as accurate and transparent as possible is critical. To continue moving in the right direction, more states need to be proactive and thorough with their post-election auditing. Connecticut only audits 5% of precincts, which is cause for criticism; while it saves time and money, it leaves gaps in the auditing process. There are still ten states (Alabama, Arkansas, Delaware, Georgia, Louisiana, Maine, Mississippi, New Hampshire, Oklahoma, and South Dakota) that do not require any form of post-election auditing [11]. Developing software to streamline the audit process pushes election security in the right direct and hopefully nudges legislators towards

more secure, cautious, and voter-centric policies. The integrity of our democracy is only as strong as our elections.

ACKNOWLEDGMENT

We would like to thank the National Science Foundation, the University of Connecticut, Dr. Bing Wang, Dr. Alexander Russell, Dr. Laurent Michel, and Della Farney and the Center for Voting Technology Research.

REFERENCES

[1] State of Connecticut Senate, Substitute Senate Bill No. 252. Hartford, CT, 2016.

[2] T. Antonyan, T. Bromley, L. Michel, A. Russell, A. Shvartsman, and S. Stark, Computer Assisted Post Election Audits, State Certification Testing of Voting Systems National Conference, 2013

[3] P. B. Stark and M. Lindeman, "A Gentle Introduction to Risk-Limiting Audits," in IEEE Security & Privacy, vol. 10, no., 2012. Available: https://www.stat.berkeley.edu/ stark/Preprints/gentle12.pdf

[4] M. Lindeman, R. L. Rivest, and P. B. Stark, Retabulations, Machine-Assisted Audits, and Election Verification, stat.berkeley.edu, Mar 2013. [Online]. Available: https://www.stat.berkeley.edu/ stark/Preprints/retabulation13.htm

[5] J. Bretschneider, S. Goodman, M. Halvorson, R. Johnston, M. Lindeman R.L. Rivest, P. Smith, P.B. Stark, Risk-Limiting Post-Election Audits: Why and How. Available: stat.berkeley, https://www.stat.berkeley.edu/ stark/Preprints/RLAwhitepaper12.pdf

[6] M. Lindeman, P.B. Stark, V.S. Yates, BRAVO: Ballot-polling Risk-limiting Audits to Verify Outcomes, Available: https://www.usenix.org/system/files/conference/evtwote12/evtwote12-final27.pdf Quantum Electron., submitted for publication.

[7] P.B. Stark, Super-Simple Simultaneous Single-Ballot Risk-Limiting Audits, Available: https://www.usenix.org/legacy/events/evt/tech/full_papers/Stark.pdf

[8] Lundh, F. and Clark, A. (2017). Pillow Pillow (PIL Fork) 4.2.1 documentation. [online] Pillow.readthedocs.io. Available at: http://pillow.readthedocs.io/en/4.2.x/ [Accessed 4 Jun. 2018].

[9] Docs.python.org. (2018). 26. Graphical User Interfaces with Tk Python 3.7.0 documentation. [online] Available at: https://docs.python.org/3/library/tk.html [Accessed 4 Jun. 2018].

[10] Stark, P. and Wagner, D. (2012). Evidence-Based Elections. IEEE Security & Privacy, [online] 10(5), pp.33-41. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6203498&tag=1.

[11] Ncsl.org. (2018). PostElection Audits. [online] Available at: http://www.ncsl.org/research/electionsandcampaigns/post-electionaudits635926066.aspx#state%20reqs [Accessed 28 Jun. 2018].