2D BARCODES AS AN ATTACK VECTOR: VULNERABILITIES, ATTACK PROCESS AND MITIGATION FOR VOTING SECURITY

Daniel LaChance
Mentored By: Greg Johnson
Center for Voting Technology Research
Univeristy of Connecticut
Storrs, Connecticut

Abstract—**QR codes are an effective way of sharing information quickly and reliably, primarily because of their error correction capabilities. This error correction feature can be harnessed for secret data sharing by changing bits that the scanners will be able to correct. This secret data can compromise voting security by violating privacy by hiding voter identification in the ballot QR code, as outlined in VVSG 2.0 guidelines. Our team developed two proof of concept schemes, tested their effectiveness, and tested codes to analyze their number of errors. We concluded that any QR code with any errors should be treated with suspicion because an attacker may have tampered with it. In order to detect errors, it is important that every voting center use a tool to scan the number of errors in a code before extracting any data. Although many voting machines' QR codes can withstand 25% error capacity, if human workers can not see what is causing a high amount of errors with their eyes, the code has most likely been tampered with. Common ink smudges and bad lighting will be recognizable to anybody but secret messages will not. It is important for voting centers to do so in order to better secure our elections.**

I. Introduction

In recent years, voting technology, and its security, has demanded increased national attention (Russell, 2014). Currently, there are many different devices such as ePollbooks, ballot marking devices and tabulators (Castillo, 2019). ePollbooks function as check-in devices for eligible voters. Ballot marking devices mark the ballot; many employ a QR code with the voter's intended choices stored inside it. Tabulators read voter choices, sometimes encoded within those QR codes, and tabulate election results (Castillo, 2019).
The use of 2D barcodes is a frequent component of the voting process and although they serve the greatest purpose, it is not only with QR codes (Russell, 2014). QR codes are used to identify ballots used to create a cast vote record during a risk limiting audit (Russell, 2014). QR codes are also used in some ballot marking devices to encode the votes cast by a voter (Merrill, 2015). ePollbook solutions are capable of scanning barcodes on government issued identification cards to check in voters (Bernhard, 2020). Due to the varied uses of 2D barcodes in the voting process, we focused our attention on vulnerabilities that could have an impact on election security. More specifically, we investigated the vulnerabilities of 2D barcodes and their impact on the security of an election.

II. Background

A quick response code is a 2-D matrix that encodes data through black and white pixels which represent binary text (Tiwari, 2016). There are forty different versions of QR codes and an increase in version corresponds to an increase in storage capacity (Tiwari, 2016). The size of these versions ranges from a 21 x 21 matrix with 152 maximum data bits to a 177 x 177 matrix with 23,648 data bits (Chow, 2020). These 2-D codes are used by the ballot marking devices to transfer voter data from the voting machine to the tabulator and into the voting database (Castillo, 2019).
The data in QR codes are encoded through the Reed-Solomon encoding method (Tiwari, 2016). This method uses four different methods of data encoding; these methods are L, M, Q, and H (Tiwari, 2016). The error correction capabilities change depending on which encoding method is being used, L has 7% capability, M has 15%, Q has 25%, and H has the greatest error margin at 30% (Tiwari, 2016). Since method H has the greatest number of capable errors, it can store the least amount of actual data for the same QR version (Huang, 2020).

This QR code feature can be taken advantage of by storing data by purposefully changing the bits on the face of the QR code (Kieseberg, 2010). If a percentage of the bits changed is less than the threshold of the encoding method, whether it be L, M, Q, or H, the scanner will automatically correct the errors and the change will go undetected by the layperson (Kieseberg, 2010).

As shown in Figure 1, Yang-Wai Chow and his research team created a graphic to visualize the data bits being changed in order to store secret information. Each of the QR codes (d), (e), and (f) have been changed significantly but they are still able to be scanned. The regions highlighted in sections (g), (h), and (i) can be extracted to reveal a secret message.

Since the bits being changed have an equal likelihood of already being the desired color or needing to be changed, the maximum threshold for secret data encoding is not equal to the method capabilities; it is actually greater (Haung, 2020). It is possible to reach a scheme where roughly 65.6% (2430 of 3706 bits) of the maximum data in QR codes using the error correction level H can be embedded with certainty (Bui, 2014). The largest QR code, version 40, using the highest level of error correction can house a secret message of up to 4860 bits (Huang, 2020). This number of bits is even enough

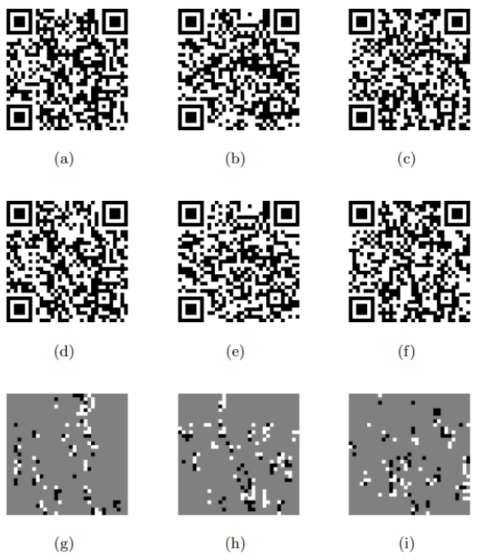
Fig. 1. Yang-Wai's secret data sharing scheme

for a small gray-scale image. Similarly, to QR codes, PDF-417 codes are also two-dimensional barcodes that house stored data using error correction algorithms (Rong, 2011). These codes are used in many government processes such as the storage of driver license personal data (Rong, 2011). Specifically, in the voting process, these codes are used to check in voters using their driver's licenses as identification (Castillo, 2019).

These 2D barcodes input data into systems that support the voting process, particularly during the check in process. These systems are vulnerable to attacks by exploiting their reliance on database technology and Structured Query Language (SQL) instructions (Shar, 2012). These SQL instructions can be hijacked through the introduction of other, malicious, SQL statements (Shar, 2012). Attackers can manipulate or access these databases through web inputs by inputting malicious SQL commands through the normal web input portal (Shar, 2012). Operators, called tautologies, allow attackers to access data without a registered account (Shar, 2012). In total, it is possible to delete information, return sensitive information, and more through the use of SQL injections on a system ill equipped to handle malformed QR codes (Halfond, 2006).

Many SQL attacks can be prevented through the implementation of input sanitizers that do not process inputs with SQL operators (or simply remove them) because most websites do not have a use for user inputs with these characters and they can serve to harm the database (Shar, 2012). It is also possible to create a man-in-the-middle web server that receives all of the inputted information and randomizes the inputs so any SQL injection would be rendered useless (Boyd, 2004).

One of the largest companies for voting machines, Dominion, has many machines with multiple vulnerabilities (Castillo, 2019). Machines such as the ImageCast X (ICX), which print out a QR code with all of the voter's information embedded within it, do not have any form of physical security, meaning it should be monitored by the voting center (Castillo, 2019). One of Dominion's most popular systems, the ImageCast Evolution (ICE) tabulator, which scans the QR codes with voter data encoded in them, does not house any data storage systems, so it must send all inputs to an outsourced database (Castillo, 2019). Additionally, if a QR code were to not match the voter intent as entered into the ballot marking device, this system would have no way of catching those errors.

Since voting machines have physical security monitored by the polling station, it is possible to use 2D barcodes as an attack vector that no human can decipher with just their eyes. For example, a QR code is not readable by any human, so it could completely bypass this physical security and transition straight to detection by the tabulator. Since the tabulator runs off a compact flash drive, it has no operating system and it immediately stores inputs and outsources them, it in turn makes that database vulnerable to possible SQL injection attacks from voter check-in using ePollBook systems. It is possible for a QR code to be used as a SQL attack vector for voting machine databases in order to jeopardize elections. Also, any possible way to make a ballot recognizable is detrimental to voting security because, if any ballot could be marked and set apart from the rest, it could facilitate candidate bribery and a breach of voting integrity. The marking of ballots in a secret manner can be done through the exploitation of QR error correcting algorithms. For these reasons, in this paper, we show our implementation of a proof of concept encoder and scanning tool that can be implemented to increase voting security by outputting the number of errors in the code face to provide an accurate judgment of whether or not the code has been tampered with. It also prevents SQL injections through basic sanitation checks.

III. Methods

1) **Preliminary Research in Error Correcting Codes**
Using Python as the main language, we devised an algorithm to change a string of binary numbers devised in the 7-4 Hamming Code process to secretly store and retrieve a message.
2) **Develop a QR Manipulation Method for Hidden Data as Proof of Concept**
In Python, a QR code library named: "qrcode" was used for the creation and eventual manipulation of data. The "qrcode" library was used due to its ease of use to prove the possibility of secret data manipulation. The process of encoding took in two strings. One served as the usable QR code data and one was secretly encoded. The error correction algorithm was exploited by purposefully changing bits encoded in error correction blocks of the QR face to store secret data.
3) **Develop a QR Manipulation Tool in Java**
The Python library "qrcode" only allows for specific bit changing but we needed to change every least significant bit in each error correction byte. In order to access each specific error correction byte, we used the library Zebra Crossing (ZXing) in Java, the same library that Apple

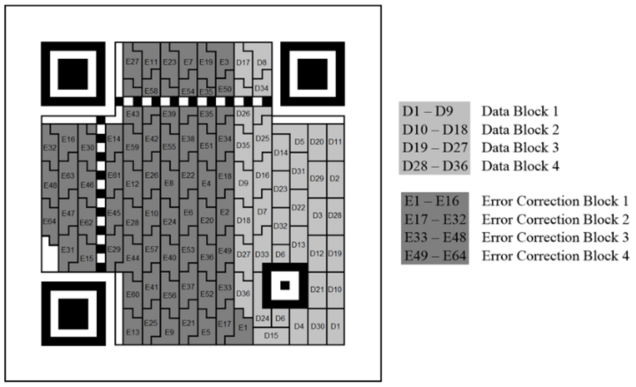
D1 – D9 Data Block 1
D10 – D18 Data Block 2
D19 – D27 Data Block 3
D28 – D36 Data Block 4
E1 – E16 Error Correction Block 1
E17 – E32 Error Correction Block 2
E33 – E48 Error Correction Block 3
E49 – E64 Error Correction Block 4

Fig. 2. Data and error correction codeword arrangement for QR code version with error correction level H. Found in Chow's article, "Exploiting the error correction mechanism in QR codes for secret sharing." All data and error correction blocks will follow a similar pattern.'

and Android QR code scanners are built off of. Using ZXing we changed every necessary least significant bit in the error correction bytes to encode the secret message.

An example of how error correction bytes in QR codes can be seen in Figure 2.

The QR code should be capable of changing $2l$ percent of its bits since every bit has an equal change of needing to be changed and not; however, we used $1.59l$ to ensure too many bits are not changed. The maximum number of bits to be changed is given by the function: $m = \frac{1}{.63} * n * e$ where $l\%$ is the error correction capacity of the code depending on the error correction level, n is the number of codewords in each block and e is the number of error correction blocks the code.

4) **Create a QR Scanner that Reports the Number of Errors**
Using the Zebra Crossing library, we adapted its Decoder class to include a method to extract every least significant bit in the error correction bytes. This allowed us to extract the message from the code face.
5) **Run an Analysis on Message Length Depending on QR Version**
Since QR code versions have different amounts of bits in their faces, we ran an analysis of the maximum character length of messages for code versions 5, 10, 15, … 40 as in Table I. After doing so, we added a few bits before the secret message to tell the scanner the character length of the secret message.
6) **Create Another QR Code Manipulation Method**
We created another algorithm that is different from the last by the location of its stored errors. The errors in the new format are centralized in one location of the face oppose to the least significant bit of every byte in the error correction bytes.
7) **Compare the Two Manipulation Methods**
We tested the locations of all of the errors and compared the two to a control. All three QR codes were encoded with the same string of randomly generated voting information. The string:
"Clarence;R.P.;Sellers;1976-03-13;586 Middle Treasure; Los Angeles;California;McCaffrey"
was encodes to mimic voting information soon to be inputted to a database along with a randomly generated secret message: "UNtVafgLMPUZcHFpmxHPYyOWIz-fppJuLutOmiSUSBiSCHSrVDiThEKO"
for both manipulation methods. QR codes encoded this secret message to varying effectiveness depending on its secret encoding scheme, version and error correction level.

TABLE I
ERROR CORRECTION CODEWORDS FOR LEVEL Q

Version	Number of EC Codewords
5	72
6	78
7	88
8	108
9	130
10	138
15	250
20	376

IV. RESULTS

Zebra Crossing (ZXing) was used to change the least significant bit in every Error Correction byte. All QR codes followed similar mapping, as pointed out in Chow's article, "Exploiting the error correction mechanism in QR codes for secret sharing," but he showed the block distribution for QR code version 5.

This scheme, shown in Figure 3 changes one bit in every byte of the error correction blocks. For error correction level Q and version 9, this scheme was capable of encoding only 19 characters.

The new scheme, shown in Figure 4 changes the entire byte of $\frac{l}{.63} * n$ bytes in each error correction block. For error correction level Q and version 9, this scheme was capable of encoding 55 characters. The rest of the max sharing data can be seen in Table II.

The rough maximum number of errors per version using the new scheme can be predicted using the equation:

$$E = 8.4958x^{1.5355}$$

The difference in error correction capacity from this scheme to previous schemes can be visualized in Figure 5.

Current Scheme Implementation:
Currently, the reader we have is capable of extracting: the version number, number of errors, secret message, and decoded text from the code and it outputs a sample of the QR code from the file.

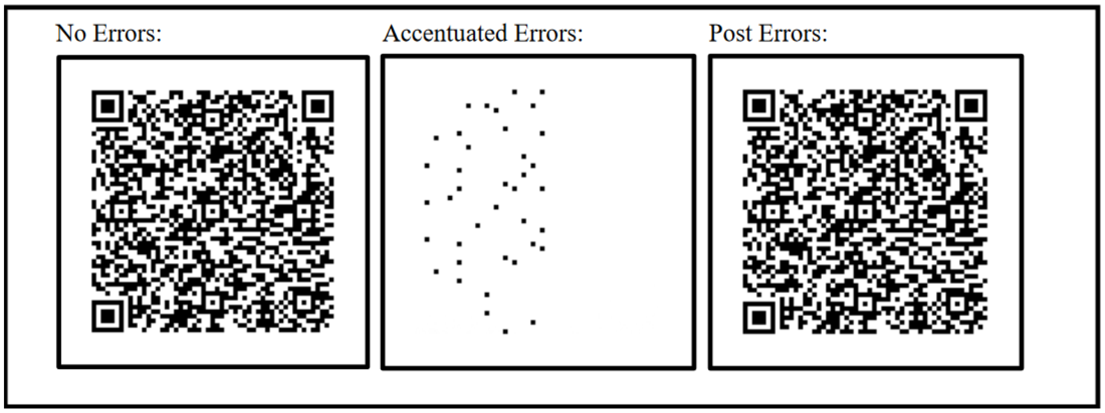

Fig. 3. Old scheme for secret data sharing

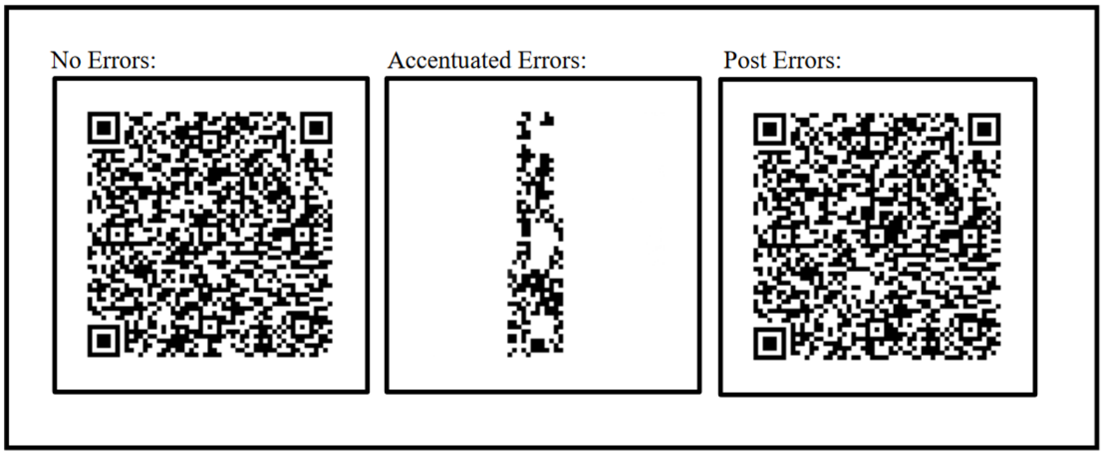

Fig. 4. New scheme for secret data sharing

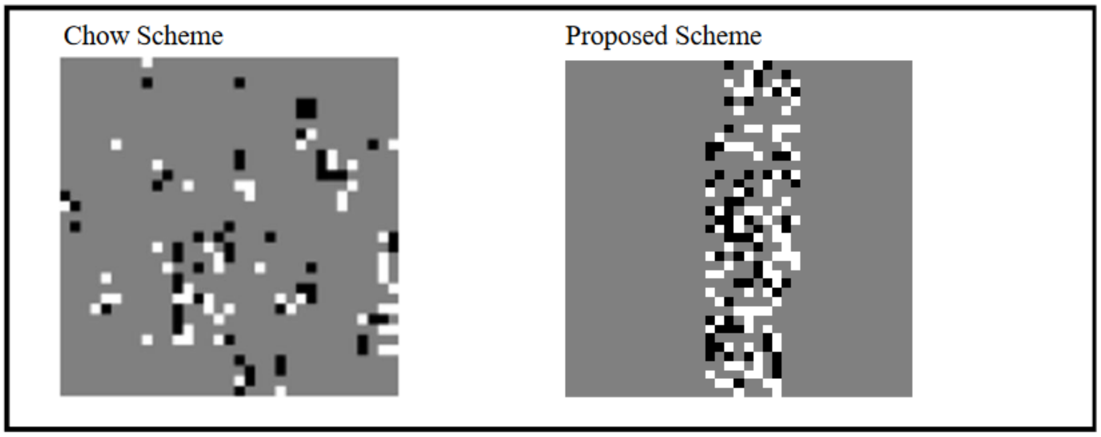

Fig. 5. Secret Errors Capacity of Huang vs. New Scheme

TABLE II
DATA CORRESPONDING TO VERSION NUMBER

Version	Number of EC Codewords	Old Length	New Length	Num Errors	Theoretical Maximum	% of Max
5	72	8	27	107	144	74.30556
6	96	11	35	143	192	74.47917
7	108	12	41	149	216	68.98148
8	132	15	47	208	264	78.78788
9	160	19	55	233	320	72.8125
10	192	23	71	290	384	75.52083
15	360	23	131	556	720	77.22222
20	600	23	219	871	1200	72.58333

Sample Output:

- **Version:** 9
- **Secret Message:** UNtVafgLMPUZcHFpmxHPYyOWIz-fppJuLutOmiSUSBiSCHSrVDiThEKO
- **Number of Errors:** 233
- **Error Correction Level:** Q
- **Decoded Text:**
  Clarence; R.P.; Sellers; 1976-03-13; 586 Middle Treasure; 86240; Las Angeles; California; President: McCaffrey; VP: Jordan

TABLE III
NUMBER OF ERRORS FOR MAX ENCODING IN VERSION 5 ERROR CORRECTION LEVEL H

Codes 1-10	Codes 11-20	Codes 21-30	Codes 31-40
164	174	175	183
162	163	179	177
164	157	177	176
166	163	153	169
176	173	175	168
171	168	158	176
165	157	152	167
167	158	173	165
172	179	163	167
165	180	165	161

In order to test the average errors for all text messages, we input 40 secret messages into a QR code of version 5 and error correction level H, seen in Table III. The secret messages were determined by a random string generator and each inputted into a QR code. The average number of errors in these codes was 168.075. Using a 95% t-interval we can conclude that the true mean number of errors per maximum message length in this case is on the interval (168.05, 168.1). Since the expected number of errors is 312 for a message of length 39, the experimental value is 7.74% higher than expected.

V. DISCUSSION

As we already knew, by purposefully changing bits in the face of the QR code, an attacker is able to store secret information and make the code distinguishable only to him or herself. This new scheme offers up the suggestion that it may be more effective, as an attacker, to centralize all of the changes being done to the face of the code. In doing so, the changes better mask themselves as a natural break or tear in the code, as in Figure 6.
It is then important for new algorithms that check for errors to not assume a malicious attacker would spread errors across the image. Both codes in figure 6 may bypass some basic security measures because they both still scan and they both appear as though the center of the code has been damaged severely, possibly by a tear or ink smudge. QR codes are meant to be able to correct for these types of damages, but an attacker is capable of manipulating the codes to his or her advantage.
In comparing the new versus the old schemes, we discovered a large difference in the error correction capabilities of each. By changing the least significant bit in every error correction byte or cycling through and changing one bit in each error correction byte, the scanner would read a checksum error. But by changing n error correction bytes, it was found that more information could be housed without resulting in a checksum error. This is presumably due to the error correction properties of the QR code as these codes are meant to correct for centralized damages.
Since this new scheme was able to house a comparable amount of errors as the Huang and Chow systems, it is possibly unnecessary to split secret messages among multiple shares of QR codes. Additionally, this scheme could theoretically be pushed even further because each QR code, with the maximum number of letters, only registered roughly 75% of the implied maximum number of correctable errors.
Although the new scheme was effective at housing secret information, the error rate was higher than is expected. 7.74% more errors were discovered in the 40 QR codes than is expected. This could have something to do with the three most significant bits in each byte being the same in each encoded letter.
The scanner is capable of outputting the number of errors in the face of a QR code, and although many Dominion ballot counting machines use QR codes of error correction level Q, we recommend that the threshold for error analysis should be set at 0. Any number of errors in the face of a QR code is a sign it has been tampered with, and it should be investigated. Simply because the code still scans does not mean it is not housing malicious information that could influence an election.

VI. FUTURE WORK

Even though the single bit conversion scheme was relatively ineffective in this paper, there can be some attempts to improve it. More than one error correction bit can be changed in each byte to house more information. Additionally, more

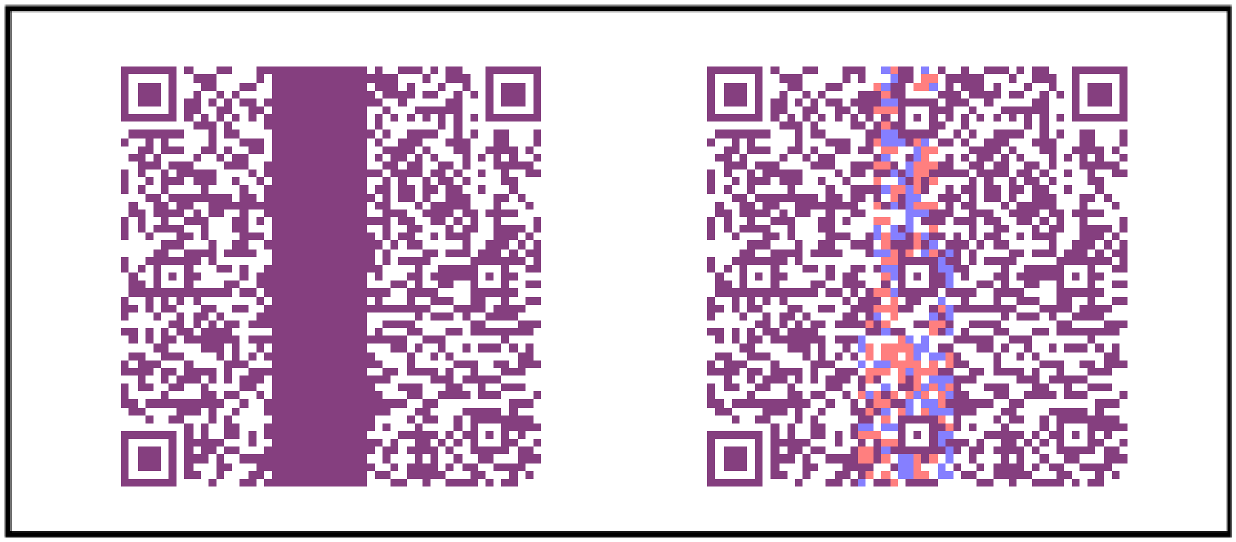

Fig. 6. Two Broken Codes

information could be housed by decreasing the number of bits used for each character. All lowercase letters in the UTF-8 encoding system do not use the three most significant bits in every byte so for pure text transfer, these bits could be removed for a 5-bit encoding system. More testing could also be done to determine the reason for the high error rate.

VII. Acknowledgements

I would like to express my gratitude to the Glastonbury Board of Education and University of Connecticut's Innovation Partnership Building for providing me with the opportunity to complete this original research.

References

[1] M. Bernhard, A. McDonald, H. Meng, J. Hwa, N. Bajaj, K. Chang, and J. A. Halderman, "Can voters detect malicious manipulation of ballot marking devices?," *In 2020 IEEE Symposium on Security and Privacy (SP)*, pp. 679-694, May 2020.

[2] S. W. Boyd and A. D. Keromytis, "SQLrand: Preventing SQL injection attacks," *In International Conference on Applied Cryptography and Network Security*, pp. 292-302, Springer, Berlin, Heidelberg, June 2004.

[3] T. V. Bui, N. K. Vu, T. T. Nguyen, I. Echizen, and T. D. Nguyen, "Robust message hiding for QR code," *In 2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 520-523, IEEE, August 2014.

[4] M. Castillo, "AP DOMINION VOTING SYSTEMS-DEMOCRACY SUITE 5.10-System Components," 2019.

[5] Y. W. Chow, W. Susilo, G. Yang, J. G. Phillips, I. Pranata, and A. M. Barmawi, "Exploiting the error correction mechanism in QR codes for secret sharing," *In Australasian Conference on Information Security and Privacy*, pp. 409-425, Springer, Cham, July 2016.

[6] W. G. Halfond, J. Viegas, and A. Orso, "A classification of SQL-injection attacks and countermeasures," *In Proceedings of the IEEE International Symposium on Secure Software Engineering*, vol. 1, pp. 13-15, March 2006.

[7] P. C. Huang, C. C. Chang, Y. H. Li, and Y. Liu, "Efficient QR code secret embedding mechanism based on hamming code," *IEEE Access*, vol. 8, pp. 86706-86714, 2020.

[8] P. Kieseberg, M. Leithner, M. Mulazzani, L. Munroe, S. Schrittwieser, M. Sinha, and E. Weippl, "QR code security," *In Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia*, pp. 430-435, November 2010.

[9] D. W. Merrill (Ed.), "Connecticut Electronic Poll Book System Requirement Specification V1.0," *Connecticut Secretary of the State*, March 6, 2015.

[10] C. Rong, L. Zhen-ya, J. Yan-hu, Z. Yi, and T. Li-yu, "Coding principle and implementation of two-dimensional PDF417 bar code," *In 2011 6th IEEE Conference on Industrial Electronics and Applications*, pp. 466-468, IEEE, June 2011.

[11] M. D. L. M. A. Russell and A. Shvartsman, "Developing requirements for electronic poll book systems: The distributed systems view and challenges," 2014.

[12] L. K. Shar and H. B. K. Tan, "Defeating SQL injection," *Computer*, vol. 46, no. 3, pp. 69-77, March 2012.

[13] S. Tiwari, "An introduction to QR code technology," *In 2016 International Conference on Information Technology (ICIT)*, pp. 39-44, IEEE, December 2016.