https://doi.org/10.2352/EI.2022.34.4.MWSF-323

A video auditing system for display-based voting machines

Scott Craver and Gurinder Bal; Binghamton University; Binghamton, NY USA

Abstract

The use of general-purpose computers as touch-screen voting machines has created several difficult auditing problems. If voting machines are compromised by malware, they can adapt their behavior to evade testing and auditing, and paper trails are achieved through printing devices under the untrusted machine's control.

In this paper we outline and exhibit a prototype of a device that audits a voting machine through screen capture, sampling the HDMI signal passed from the computer to the display. This is achieved through a standard that requires a compliant voting machine to display signal markers on the summary pages before a vote is cast; compliance is enforced via alerts to the voter with a visual and audible signal while the screen is captured and archived. This direct feedback to the voter prevents a compromised machine from failing to invoke the device. We discuss the design and prototype of this system and possible avenues for attack on it.

Introduction

It is a difficult problem to guarantee that touch-screen DRE (direct-recording electronic) voting machines, if compromised, are correctly tabulating votes as entered by voters. DRE machines input a voter's choices and save a vote directly to a memory card, rather than scanning a paper ballot; because DRE machines as general-purpose computers may be compromised or infected with malware, such a machine might record false votes, or otherwise alter its totals [2]. To this end, an auditing method is needed to verify that a DRE machine is behaving correctly. The purpose of this project is to design a reliable auditing method and technology for electronic voting machines, under the assumption that the machine is programmed to record votes unfaithfully, by a malicious actor with full knowledge of the auditing system. An ideal solution would use existing machines, with minimal additional participation by voters.

Verification methods include testing a machine before the election with mock data, isolating one of several machines during an election to test with mock data, and comparing tabulated results with a voter-verifiable printout produced by the machine. In the first two cases, a compromised machine may behave correctly if it can determine that it is being tested; in the latter case, it is necessary for a voter to verify a displayed printout, which the voter may opt not to do. It is thus challenging to verify tabulated data with voters' true choices. In [3], the authors propose a method of executing a DRE machine's software as a virtual machine from a hypervisor, and directly examining its video buffer output to confirm that a vote is displayed as recorded. The authors' use of a video frame buffer is one step closer to the method that we employ, except that the authors only use this to confirm or disconfirm that a ballot matches that which is recorded, rather than recording the buffer itself for auditing.

In this paper, we propose a novel method of auditing a DRE voting machine, by capturing the video signal sent from the DRE machine to the display. This prevents the need for the user to compare the vote to a recorded printout, and it allows an isolated device to perform the auditing capture, outside the influence of the potentially compromised DRE machine. However, to do so efficiently (and not simply record the entire video output of the DRE machine,) it is necessary to induce the DRE machine's cooperation. We achieve this by establishing a protocol that the machine must correctly participate in, and enforce this participation through aural and visual alerts to the user.

Method

In our proposed system, a DRE voting machine is assumed to be connected to a display device through an HDMI video cable. The video signal to the display is passed through a capture device that sits inline on the cable, whose signal is copied to an embedded computer. This device is programmed to identify vote summary pages, which must be displayed before a vote is cast; these frames are captured by the device, and saved to a memory card.

In order for this system to work, it is necessary for the DRE machine to display, possibly on multiple screens, a summary of the voter's ballot before the vote is cast, in such a way that the device can detect when to take a screen shot. This creates the obvious problem that a malicious machine may refuse to cooperate, either by not displaying such pages, or doing so in a way that foils the device's ability to detect a frame to capture. Our solution to this problem is to require a device to participate in a protocol that is made clearly evident to the user, with a notification if the machine is not in compliance.

Protocol description using state badges

We model the voting process at a DRE machine as a simple finite state machine, with states described in table 1. These states simply encode the phases of a voting process, which either culminates in a vote being cast, or a session being canceled. Only specific transitions are allowed between these states; the state transition diagram is shown in figure 1.

Phases (states) in DRE voting process

00	Election has not begun
01	Interim between voters or voting sessions
02	Start of a new voting session
03	Voting session is underway
4x	Summary page x is displayed on the screen for capture
05	Vote has been cast
06	Voting session is completed

In order for the machine to document that it is following this state machine, it displays *state badges* on the screen. A state badge is a small graphic that encodes the state value, which our device can recognize in the frame of video passed over the HDMI connection. Sample state badges are shown in figure 3.

These badges are designed to be easily detected on-screen in HDMI video. Because of the strictures of the format, it is not necessary to use a QR or AR code, since the badge is not subject to geometric transforms [4, 5]. Because of the nature of the displayed data, a simple pattern can be detected with a low risk of either a miss or a false detection.

Our state badge images consist of horizontal bars of pixels, 20 pixels in width, each bar consisting of a constant pixel value that is easily detectable in YUV color space. In our case we chose two colors, yellow and cyan, which will present a high Y value and clearly separated U and V values. The badge consists of two rows top and bottom as framing data, enclosing eight bits encoded as eight rows. Eight bits are almost certainly more bits than we need for our application.

In our proof of concept implementation, we observed that captured YUV values vary from source to source; the framing data allows us to fix exact U and V pixel values when decoding.

Protocol enforcement

When a suspect machine displays state badges on screen to indicate each phase in the voting process, our device identifies the state communicated through these badges. It then communicates directly to the user, both with a sound (speech) alert and a light. This naturally confirms to the user that the DRE machine is doing what it claims to do.

Table **??** below displays all of the transitions between state badges, the message announced when the transition occurs, and the light color displayed in this instance.

The instruction set includes a set of opcodes that indicate stages in the voting process. They include:

These opcodes can only be displayed in certain valid sequences, described by a finite automaton. Each state transition triggers the device to provide a visual and audio alert. The visual alert consists of a red, amber, or green light that may be solid or flashing; the audio alert consists of a voice sample describing the state to the voter. Example text messages are illustrated in the table below.

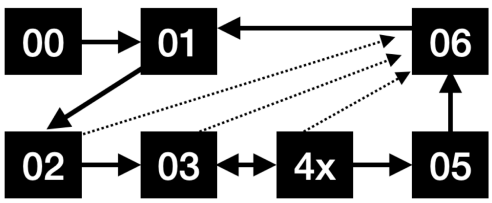

Figure 1. *State transitions allowed by the voting process.*

Figure 2. *Example state badge images.*

State transitions in DRE voting process

From	To	Meaning	Light	Message
00, 06	01	Machine ready	Yellow flashing	This machine is available.
01	02	Voting begins	Yellow solid	Your voting session has begun. Your vote is not cast until the light is green.
03, 4y	4x	Summary page X	no change	You should see page [x] of your ballot summary on the screen.
4x	05	Vote cast	Green solid	Your vote has been recorded.
05	06	Voting complete	Green solid	Thank you, and goodbye.
02, 03, 4x	06	Voting canceled	Red solid	This session has been canceled. No vote is recorded.
*	*	Wrong behavior	Red flashing	A fault has occurred. Please contact an election worker. No vote is recorded.

When a 4x opcode (summary page displayed) is received by the device, a screen capture of the summary page is taken. During such a state, each new frame is compared to a stored previous frame to ensure that the summary page is unchanged; if a sufficient change is observed, the new frame is also saved as a screen capture.

The purpose of the voice-and-light alert system is to force a machine to display a summary page with a 4x opcode, so that the auditing device can record a screen capture. A malicious machine that fails to display a 4x opcode marker will fail to reach a confirmation of the cast vote, and result in an error announced to the voter.

Implementation and testing

Our system was implemented using a laptop computer to represent our DRE machine, serving as an HDMI source. The HDMI signal was captured using an Elgato HD60S+ capture card. This device provides an HDMI video in and out signal, sitting inline on the cables between a computer and a display. The HD60S+ captures the signal on the display and exports it as a USB camera, that can then be captured by another device. This forms the video capture of our test bed, although it should be emphasized that this is an off-the-shelf technology for a proof of concept. As we will show below, the conversion of HDMI video to a USB camera output entails complexities in the color space of the video available for auditing. If implemented as a standalone device, it may be better to use a device that captures the HDMI signal directly, such as an FPGA board with direct access to HDMI through data.

Our captured data is then exported as a virtual camera to a

Raspberry Pi 4B, running a real-time utility to monitor and process frames from the camera. The Rasbperry Pi was more than fast enough to process data at speeds sufficient to monitor, detect and screen-capture a summary page displayed by our HDMI source.

For the software, we modified and forked a utility called `v4l2grab`, an open source project for screenshop capture of USB camera video. This utility is able to capture a single screenshot in JPEG format, or run in continuous mode, saving screenshots as rapidly as they can be acquired. The `v4l2grab` utility is written in C, and possesses a very simple structure: it possesses a single C callback function called `imageProcess()`, which decodes a frame from the camera into a $YUV444$ color space, and then calls a method `jpegWrite()` to write the converted buffer in JPEG format.

It was a trivial exercise to modify this callback function to scan the image looking for a state badge, and then returning immediately if there is no need to save the image.

If a state badge is identified, our modified utility takes action as follows:

- If the badge is the same as the one identified in a previous frame, the function returns.
- If the badge is different as the one identified in a previous frame, the finite state machine graph is used to determine an action taken. For the time being, the action (spoken message) is printed to standard out.
- If no state badge is identified, this is treated as an invalid state in the finite state machine, except at the beginning of the program's execution. Once a badge is seen, a missing badge is regarded as an error.
- If a new badge is specifically of the form $4x$ for some page x, the function simply falls through to the existing code for processing and saving a screen shot.

One complication in our implementation was data appearing in a YUV format that was not supported by `v4l2grab`. This YUYV format was simple to process, because the utility gives us direct byte access to the buffer; however, we had to write an additional YUV conversion function in order for `jpegWrite()` to be called.

Identifying state badges

Our initial results showed that different source devices, displaying the same purported colors (yellow `0xffff00` and cyan `0x00ffff`) produced significantly different YUV values. This we attributed in part to the processing imparted by the ElGato capture card.

Our code use a very simple thresholding method to identify yellow and cyan pixels: in either case a pixel must have a Y value exceeding a luminance threshold (in our case 120), and then U and V values one of which lies below a dark threshold and the other above a light threshold (chosen in our case as 90 and 120, respectively). This was sufficient to identify cyan and yellow pixels correctly across multiple devices that we used in our experiment.

Our algorithm proceeds by processing the image frame top to bottom, and left to right. If a cyan pixel is encountered, the subsequent interval of cyan pixels is measured in the scan line. If this interval length is within an interval of 10 to 20 pixels, bars in subsequent rows are checked for an all-cyan or all-yellow pattern, including the specific framing pattern at the top and bottom of the badge.

In our tests, badges were combined on either a white or black field, with and without text representing a ballot. We encountered no errors in identifying or decoding state badges, regardless of their placement in the image. This we attribute both to the confined nature of the digital data, and the nature of the video data in which the badges are immersed. The badges as we designed them are far more robust than is necessary to communicate a byte of information to a device snooping the HDMI signal. It is important to note that if these badges are used as a protocol standard, there is no incentive for a DRE machine to provide an unreadable badge; this will cause the auditing device to declare an error to the user.

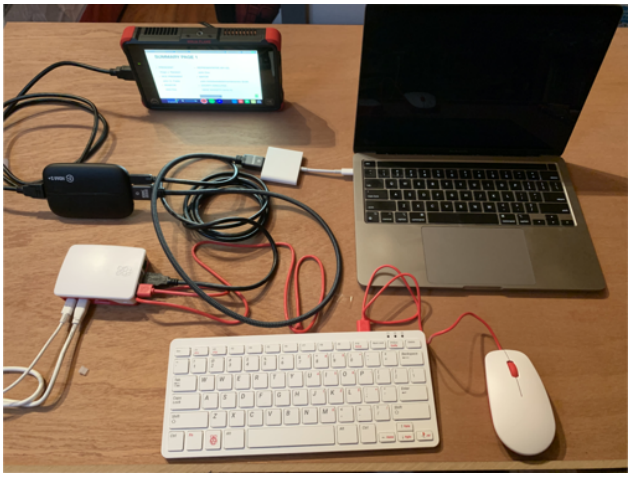

Figure 3. *Prototype of proposed system. A laptop serves as HDMI source (untrusted voting machine.) Right, an HDMI capture card, feeding a Raspberry Pi 4B auditing device (below left).*

Attacks

We describe several possible ways a malicious DRE machine could circumvent this system. We classify attacks in two categories:

1. A DRE machine displaying a summary page to the user, and triggering the recording of a screenshot of a different summary page.
2. A DRE machine displaying a summary page to the user, and triggering the the recording of a nonsensical, blank or otherwise incomplete screenshot.

An attack of the first kind is severe; an attack of the second kind identifies under audit that the machine has misbehaved, although the voter's ballot is lost.

An example of the first attack is described in [6]: it is possible to tamper with a display monitor's display controller, by updating its controller firmware, in order to direct the monitor to display pixels that are not in the HDMI signal sent to the monitor. For example, one can trigger an image to appear in a designated region on top of the HDMI signal sent by the computer. This attack is specific to certain displays, requires tampering with a

device's firmware, and is somewhat confined in what can be displayed; however, the authors demonstrate that a controller can be fed data to display, and commanded to do so, through pixel values set in an HDMI signal. This represents a fundamental limitation on the trustworthiness of a video signal as "ground truth" for what appears on a display.

Another potential attack is employing a second device, in a meet-in-the-middle attack. If the cable to a display is unplugged, and fed to an intermediate device that can manipulate HDMI signals, this device can act as a surrogate for a hacked display.

Beyond hacked displays, there are potential attacks by which a DRE machine fools the screen capture directly. A machine can briefly display a false summary page to the user, and then change it ("correct" it) after a short interval, after the screen-shot is taken. This might be blatant behavior that will alert a voter to a problem, but it represents a risk that must be addressed.

Another attack consists of displaying a blank summary screen that switches to a summary page after a brief delay, to trick the auditing device to record a blank screen. Unlike a decoy page, this may not appear suspicious to a user, rather appearing like a natural delay in the DRE machine's operation. However, it would be an attack of the second kind, that would be discovered upon auditing. Along these lines, attacks consisting of rapidly alternating a summary page and a blank screen, or alternating between images whose average appear as the desired summary page, can cause the device to record a false image.

To remedy these attacks, we recommend that in practice, an auditing device in a summary page state compare subsequent frames of video, declaring an error if a significant change is detected from frame to frame while the state badge is constant.

Discussion and conclusions

This paper outlines a method of auditing a DRE voting machine by screen-capture of a ballot summary, serving as an alternative to a voter-verifiable paper audit trail. This method allows an independent hardware auditing device, as long as a DRE machine adheres to a protocol entailing the display of markers to inform the auditing device of advancing stages in the voting process. Aural and visual alerts ensure that the machine is doing so.

One avenue for future work is that of employing screen capture using a camera positioned over a display, rather than capture of a video signal to a display. This would rule out attacks that produce a display distinct from the video signal [6]. However, such an approach would face significantly greater image processing requirements. In addition, there is a potential for robust watermarking to be used in place of an overt marker image, also with a cost of computational complexity.

References

[1] Stevan Eidson, Brett Gaines, and Paul Wolf. 30.2: HDMI: High-Definition Multimedia Interface. SID Symposium Digest of Technical Papers. Vol. 34. No. 1. (2003).

[2] Ariel Feldman, J. Alex Halderman, and Edward W. Felten. Security analysis of the Diebold AccuVote-TS voting machine. Usenix Security (2006).

[3] Sujata Garera and Aviel D. Rubin, An independent audit framework for software dependent voting systems. Proceedings of the 14th ACM conference on Computer and communications security. (2007.)

[4] Ohbuchi, Eisaku, Hiroshi Hanaizumi, and Lim Ah Hock, Barcode readers using the camera device in mobile phones. 2004 International Conference on Cyberworlds. IEEE. (2004.)

[5] KU Gökhan, and İN Gökhan. ARgent: A Web Based Augmented Reality Framework for Dynamic Content Generation. Avrupa Bilim ve Teknoloji Dergisi: 244-257. (2020.)

[6] Ang Cui, A Monitor Darkly: Reversing and Exploiting Ubiquitous On-Screen-Display Controllers in Modern Monitors. DEFCON 24, Aug. 5, 2016.

Author Biography

Scott Craver received his PhD in Electrical Engineering from Princeton University in 2004, and is an assistant professor of Electrical and Computer engineering at Binghamton University in Binghamton, NY. Gurinder Bal is a recent graduate of Binghamton University with a Master's degree in Electrical and Computer engineering.