

TOUCH SCREEN VOTING ARE THERE PROBLEMS?

Is fraud the basic danger?

No. While fraud is a worry, promoting public confidence is the primary issue, graceful recovery from the bugs in the system is a close second, and avoiding costly errors a distant third. The basic rule of thumb is that one wants to avoid any situation where one is *unnecessarily* placing trust in a person, business, or machine.

Even the maker of Australia's touch screen voting system has said as much "why should you have to trust me?". While there are in fact plenty of reasons to question the craftsmanship, security competence, and altruism of voting machines sellers this is largely irrelevant: a properly designed voting process can remove the need for blind faith.

In as few words as possible, what are the two basic problems?

Currently available touch screen voting systems all have two fatal flaws: 1) they are designed in such a way that the software, hardware and operator training must be flawless 2) the process unnecessarily makes us trust that the manufacturer is a saint. All engineers know it is simply not possible to write flawless complex software or, at least, to ever know that you did. Instead mission critical systems are designed to be fault tolerant: they fail safely and have a backup.

In voting this means voter-verifiable, recountable, hard copies of the ballots. We can eliminate the need for blind faith in the manufacture's good intentions and/or competence through the use of open source software, in much the same way we require open-meetings laws.

Logic and accuracy tests cant find subtle flaws and neither will code reviews. Open source has the advantage of not only more eyes on the code, but because there are no non-disclosure agreements, the most skillful eyes can look at it.

The voting-machine salesman says only Henny-Penny kooks think there are serious problems. Is she right?

No, organizations more qualified and less vested strongly believe there are serious outstanding issues. Up-to-date analysis by

the Library of Congress (CRS), General Accounting Office, The Ohio Attorney General, Maryland's SAIC commission, and qualified technologists have found the Federal Election Commission and NASED certification and testing process egregiously outdated, and have suggested that touch screens bring unprecedented and not yet addressed vulnerabilities into the election process.

Confirmed reports, most recently in Virginia, Florida, and North Carolina, of serious touch screen voting system malfunctions prove that certification and testing is not sufficient to eliminate design errors in complex systems. NIST has not yet revamped voting standards, and current machines have not even qualified for lesser standards used in health-care or Military hardware.

Are only a few people concerned?

About a third of United States citizens live in states that have or are considering requiring Voter-verified paper ballot systems. Other states have voted to delay implementation.

Problems and concerns motivated Secretaries of state in California, Ohio, Nevada, and Louisiana, as well as the NY state Assembly, to request a costly replacement/retrofit of their current Touch Screens to permit voter-verifiable balloting or other improved standards. Indeed California halted the use of existing systems after a vendor was caught making unauthorized changes to certified software.

Is Diebold the only bad system?

Diebold Systems get the most abuse only because, due to a computer security lapse, they accidentally made public confidential problems with their systems. There is no reason to believe other companies are any better; they just have not exposed themselves to the same degree. Indeed the Sequoia data collection software uses the notoriously insecure MS Windows.

In fact, any company that won't make their source code and hardware open should be assumed to have security problems. It would be very hard to cite a counter example.

FIVE MYTHS OF INVULNERABILITY IN TOUCH SCREEN VOTING SYSTEMS

Even though fraud is not the issue we want to rely on, it's worth countering myths about voting system invulnerability.

1) *Touchscreen proponents will tell you the scale of conspiracy required to achieve an intrusion or software backdoor is huge.*

This is not true. Most backdoors are not deliberate but simple coding errors. And most exploits are found and viruses written by a single person. Witness for example yesterday's discovery of an unintentional backdoor in Windows arising from a single line of source code missing a single word. Even if this had been deliberate, it passed all code reviews and lurked for years; it did not require any conspiracy, and was discovered by a lone hacker

2) *Touch Screen proponents will tell you that source code certification and escrow is enough to protect us.*

This is not true. All of the documented mistakes to-date were on systems that fully passed all testing and certification, exceeded all state federal standards, and many had escrowed source.

Source code escrow has proven useless since although the code is escrowed the so far states are not allowed to look. Worse, the public has no access and the most trained eyes are prohibited by non-disclosure agreements.

3) *Touchscreen proponent will tell you that their voting kiosks are invulnerable because they are never connected to an insecure Internet connection.*

This is not true. An equivalent statement has been made for the much more tightly regulated Automatic Teller Systems used by banks. These are never connected to the Internet, *in theory*. In fact, the internet worm Blaster infiltrated the ATM systems of two major banking institutions. How? Because even though it was against policy, someone somewhere did connect a computer directly to the Internet, if only for a moment.

Moreover, though any given computer might never be connected to the Internet, the software on it was almost certainly written on a computer that was connected. Indeed it

was probably written in another country.

4) *Touchscreen proponents will tell you that since future ballots may not be known at the time the vendor programs the machine they cant put anything bad in the software.*

This is not true. For example, a machine could be rigged such that when a malicious voter touches the screen in a certain combination of places then whatever ballot is cast next will become the "desired" outcome for the machine. This is exactly how slot machine software has been rigged (to change the payoff schedule following a combination of coins and button presses). Other successful slot machine hacks include using the daily "testing software" to secretly re-program the systems en-mass through a backdoor. It should be noted that the security controls for successfully attacked Nevada slot machine software obey much tighter regulation than voting machine software yet hundreds of millions are stolen.

5) *Touch Screen proponents will tell you systems cant be corrupted because Election Officials only enter data, not programmes.*

This is not true, and in fact is the single most common attack. Election officials work at a deliberately simplified level normally only entering ballot description file. In principle this is just data not a program in exactly the same sense that an e-mail message is just data, a word processor document is just data, and a web page is just data; the underlying program is unchanged by the data, *in theory*. But all of these similar cases have been compromised when bugs allowed the data to contain executable instructions.

Indeed, a newly-found Windows backdoor is entered by opening a photograph in the web browser. Theoretically, an image is pure data, but a bug placed executable instructions on the command stack. One would be truly naïve to believe that elections software won't contain analogous exploitable bugs that allow program alteration via innocuous ballot description files.

HACKING PRIMER: JUST A FEW OF MANY WAYS TO ATTACK A TOUCH SCREEN COMPUTER

Easter eggs and code bombs

At Microsoft it's a firing offense to insert "Easter eggs" (secret backdoors) in software, and code is given multiple layers of security reviews. Despite this thousands of MS Easter eggs have been documented, some of staggering proportions (A flight simulator appeared in Excel when the right spreadsheet rows were clicked in the right order.) Recently discovered security holes that let viruses propagate have been in place for almost a decade.

Maryland's SAIC commission on touch screen voting has said in regard to voting system software that its 99.9% certain that maliciously inserted back doors would never be detected via code review.

Physical access hacks

Virtually all computer experts agree that if you can get unattended physical access to a computer you "own" it. But how?

The Albuquerque Tribune reported that 47 polling machines were destroyed when the truck carrying them crashed. Only because of this serendipity was it learned that the drivers escorting these machines to the polls were 1) drunk, 2) had no driver's licenses, 3) had left the machine unattended in the parking lot of Hooters for hours, and 4) were caught in a part of Albuquerque not on the path to the polls. Despite laws, policies and good intentions our own election officials cannot guard these machines from physical access.

Nationally, machines are commonly left protected only by simple locks. Last month it was demonstrated that amateurs could pick the locks on the Diebold system in under ten seconds. The Ohio Secretary of state noted that the same Sequoia systems kiosks New Mexico has approved will go into supervisor mode if an unsecured yellow button on the back is pressed—no lock picking needed.

Binary Code hacks

Of course source code is not what runs on the machines. The compiled binary is what is actually loaded. So how do you know that the binary that was loaded

corresponds to the human-readable source that you reviewed? This is a very very tricky technical problem. Source escrow does not address this at all.

Data Transport hacks.

Voter data has to be stored and transported somehow; memory cards, disks, or Internet. And it has to be collated on some computer. It has already been proven that the encrypted "tamper-proof" Diebold memory cards are anything but tamper proof and the votes can be undetectably altered. Not only can vote collation systems modify vote totals, already have they been hacked in actual elections.

In King County (Seattle, Redmond), the election officials found it cumbersome to use Diebold's GEMS database access software due to its security features and limited capabilities. Instead they used an unauthorized tool to directly access the database, bypassing the security system. They had the capability to alter the system logs to erase any record of a change made to vote totals done by this backdoor. Did they make illegal changes and erase the logs? One hopes not, but no one can actually prove it did not happen.

Access Card hacks

Commonly, a single-use activation card is given to the voter that allows him to go into the "booth" unattended and cast only a single vote. Its been shown already on Diebold systems that these cards can be forged. Worse yet, similar, forgable, cards are commonly used by Election Supervisors to program the machines.

Embedded Processor hacks

Most "computers" actually contain several embedded slave computers you rarely think about. For example, the graphics card and CD ROMs can contain reprogrammable firmware and would be an excellent place to unobtrusively attack a touch screen system.

Another approach is \$50 device, about the size of an olive, that can inserted into any keyboard cable; its called a "key logger" and it's actually an embedded computer that can record and alter any keystrokes or screen presses.



SAMPLER: WHAT DOES A SECURITY HOLE LOOK LIKE?

Examples of actual source code bugs and hacks.

Hacks and bugs can be subtle and elude detection for decades. Certainly, the bugs in the Mars landers were overlooked till they caused a problem. But few lay people ever actually see what they look like. Here are two real and recent examples: the second one demonstrates how “data” inputs can be used to reprogram a machine. The first one shows how an operator can create an unusual condition that promotes him to supervisor status.

Attacks on source code “trees” (the master copy used in development by the manufacturer) are the most deadly point of system wide infiltration. For example, a branch of the Linux kernel tree was recently attacked successfully: a single line was changed from (approximately)

```
If ( WUSER_FLAG || setuid == 0 ) then return 1
```

To

```
If ( WUSER_FLAG || setuid = 0 ) then return 1
```

The deletion of that single “=” character changed this innocuous statement from one that checked if the user had administration privileges to one that granted the user administration privileges under certain conditions. It proved subtle: trained eyes missed this change in an initial code review. This change would have found its way permanently into nearly all copies of Linux had the attacker covered up of the computer break in.

Yesterday, after the secret Windows source code was leaked, the following bug was discovered in the web page bit map image handler in
win2k/private/inet/mshtml/src/site/download/imgbmp.cxx:

```
// Before we read the bits, seek to the correct location in the  
file
```

```
while (_bmfh.bfOffBits > (unsigned)cbRead)
{
    BYTE abDummy[1024];
    int cbSkip;

    cbSkip = _bmfh.bfOffBits - cbRead;

    if (cbSkip > 1024)
        cbSkip = 1024;

    if (!Read(abDummy, cbSkip))
        goto Cleanup;

    cbRead += cbSkip;
}
```

The bug was using a signed integer for an offset: `int cbSkip` should have been `unsigned int cbSkip`. To exploit all we have to do is create an image file with `bfOffBits > 250` Megabytes and we're in: `cbSkip` goes negative and the `Read` call pushes our data's hidden payload onto the instruction stack. Innocuous “Data” has become an executable program Microsoft had never found this bug themselves; it took thousands of eyeballs before one person discovered it.