Strategies for Software Attacks on Voting Machines

John Kelsey, NIST, September 2005

Existing attackers ***are*** *that sophisticated, and these attackers are probably not as smart as the ones that might be brought to bear against a voting system.*

1. Introduction

This white paper discusses strategies for changing the outcome of an election via software attacks on voting machines. This discussion mainly focuses on DREs, but applies as well to DRE+VVPAT and PCOS voting systems. I am going to consider a number of the operational difficulties of software attacks and point out how these may be overcome. The goal of this white paper is to explain why I think software tampering in voting systems is a practical threat. There are operational difficulties, but I am convinced that these can be overcome by a skilled and intelligent attacker.

The nature of software-based attacks on electronic voting systems is that electronic records are changed. Depending on details of the attack, paper records may also be produced, and in some voting systems, the tampered software can alter the paper records in some way, albeit usually with the possibility of the voter noticing this.

1.1. Background and Environment

In the last several years, there have been increasingly sophisticated software based attacks on real-world systems. Among the targets have been:

- US government systems, including those containing classified data;
- Financial systems, including attacks that gained perpetrators large sums of money;
- Content protection systems intended to stand up to extensive external attack;
- Special-purpose cryptographic devices intended to be resistant to both software and physical attack; and
- Cryptographic and security software, again designed specifically to resist attack.

This is today’s environment. It is important to understand that we probably hear of only a small fraction of attacks on real-world systems. For each high-profile case of someone eavesdropping on a congressman’s cell phone or the pagers of secret service agents, there must be many other cases where the attackers don’t disclose what they’ve learned. For every case where financial data is tampered with and the theft is discovered and reported, there must be many other cases where it is never detected, or is detected but never reported.

In addition, we have seen the rise of sophisticated attacks on widely-used computer systems (desktop PCs) for a variety of criminal purposes that allow the criminals to make money:

- Activities/methods such as phishing (spam intended to get users to disclose private data that allows an attacker to steal their money) and pharming (exploitation of DNS[1] to redirect legitimate web traffic to illegitimate sites to obtain private data) continue to grow.
- Extortion against some computer sites continues, with an attacker threatening to shut down the site via distributed denial of services (DDOS) attack unless he is paid off.
- Large networks of “bots”—random users’ computers which have been taken over by an attacker for use in the above kinds of attacks, are bought, sold, and rented.

The sophistication of these attackers undermines the common responses to discussions of software attacks that “attackers wouldn’t be that smart.” Existing attackers *are* that smart, and these attackers are probably not as smart as the ones that might be brought to bear against a voting system—which might include national intelligence services of foreign countries. It is very hard to make an argument that the Russian or Chinese intelligence services can’t find attackers who are more competent than the ones currently making money from spamming, phishing, pharming, DDOS, and related attacks. It is even harder to make the argument that these organizations wouldn’t be interested in changing the outcome of a US national election.

1.2. Targeted Voting Systems

In this white paper, I am focusing on voting machines which are close to the voter, including DRE voting machines with or without paper audit trails, voting machines to fill out an optical scan ballot, and the machines which scan optical scan ballots. I do not generally work out full attacks—instead, focusing on the initial step of getting tampered software onto a voting system, controlling its actions so that it can change an election outcome in a reliable way, and keeping it from being discovered while doing so. Part of this may involve interfacing with other parts of a more complex attack.

2. How the Tampering Program Works

There are many ways for the tampering program to work. Without trying to get into fine details of the various voting systems, we can describe a number of broad methods for the tampering program to alter votes.

2.1. Changing System Settings or Configuration Files

[1] Domain Name System (DNS) is a distributed database that stores mappings of Internet Protocol addresses and hostnames to facilitate user-friendly web browsing.

The first method considered in to change the system setting or configuration files. There are many ways this can work. An attack program must tamper with the system settings or configuration files after L&A testing, but has a great deal of flexibility as to when to do so. The attack program can be buried in some driver or program that is only run when the voting is started, or some timed program that decides whether to trigger at a fixed time each day. Among the attacker's options within this class of attacks are:

- Swap contestants in the ballot definition or other files, so that a vote for John Smith is counted as one for Mary Jones, and vice versa, all the time. This only makes sense if the swapping can be applied selectively, and done only at voting machines which are likely to get a majority of the "wrong" vote. (This is an attack described in the RABA report[2], but we propose doing it wholesale instead of retail.)
- Alter configuration files or system settings for the touch screen or other user interface device, to cause differential error rates for one side vs. the other.
- Alter configuration files or system settings for the scanner to introduce differential error rates for one side vs. the other.
- Alter configuration files or system settings to make it easier to accidentally skip a contest or misrecord a vote, e.g., by increasing or decreasing touchscreen sensitivity or misaligning the touchscreen.
- Alter configuration files or system settings to change the behavior of the voting machine in special cases, such as detected undervotes or overvotes, or fled voters.

The main operational problems with these attacks include:

- Leaving incorrect configuration files at the end of voting, which may reveal the attack. The attack must thus trigger twice, once to change the configuration to an incorrect state for the attack, once to change it back.
- Deciding when to trigger—many of these attacks will cause voters' intentions to be misrecorded without regard for which way they're voting. Those attacks must trigger only for voting machines which are mostly used by people voting the "wrong" way. This implies either selectively installing the attack program, or selectively triggering it.
- Some of the attacks in this category may require fine knowledge of the format of the ballot definition files, though it is not clear that this must always be true.
- Changes in system settings or configuration files are likely to leave entries in the event logs. These entries must either be prevented or deleted by the attack program if the event logs are checked.

2.2. Active tampering with user interaction or recording of votes

In this class of attack, the attack program triggers during voting and interferes in the interaction between the voter and the voting system. For example, the attack program may:

[2] http://www.raba.com/press/TA_Report_AccuVote.pdf

- Tamper with the voter interaction to occasionally introduce an “error” in favor of one contestant.
- Tamper with the voter interaction both in vote entry and verification, so that the voter sees consistent feedback that indicates his vote was cast correctly but the rest of the voting machine software sees a changed vote.
- Tamper with the electronic record written after the verification screen is accepted by the voter, e.g., by intercepting the function call to write the results and altering those results before they are written.

This class of attack seems to raise few operational difficulties once the attack program is in place. One operational difficulty is of interest in dealing with systems with paper records:

- The attack which introduces biased errors into the voter’s interaction with the voting system is especially useful for attacking DRE+VVPAT and PCOS systems where the paper record is printed or filled in by the voting machine being attacked, since the attack behavior, if detected, is indistinguishable from user error. However, the attack program can improve its rate of successfully changed votes, and minimize its chances of detection, by choosing voters who are unlikely to carefully check their paper records. Thus, voters using assistive technology are likely targets, though there are probably not enough such voters to change the outcome of most elections.

2.3. Tampering with electronic memory after the fact

An alternative approach is to change votes in electronic memory at the end of voting, but before the totals are displayed locally or sent to the tabulation center.

In this case, the attacking program need only be activated at the end of voting. This also allows the attack program considerable flexibility, as it can decide whether to tamper with votes at all based on its local totals, which can include number of votes and elapsed time voting (to avoid being caught by parallel testing in many cases).

However, this raises a few interesting operational difficulties:

- This class of attack only works on voting machines that produce the electronic records, such as scanners in PCOS systems, and DREs. It is of no use against ballot marking devices. Attacks on systems that produce a paper record as well as an electronic record require an additional attack step to avoid detection.
- DREs typically store electronic records in many locations; the attack program must change them all.
- The attack program must avoid leaving entries in the event or audit logs of its accesses to the electronic totals which would indicate an attack. (If simple file access is logged, this raises no problems for the attack program; if each record

altered yields a log entry, this requires tampering with the event log to avoid detection.)

- Depending on details of the file accesses required, the attack program may face some time constraints on making the desired number of changes. However, note that a program that is to change 5% of votes for Smith into votes for Jones can simply hop around at random in the set of votes (they can't be stored sequentially for voter privacy reasons) and process about 10% of them to accomplish its goal. There also will very likely be a reasonable span of time between the closing of polls and the display and transmission of results.

3. Attack Program Control Strategies

3.1. Overview: Attack programs, remote control, and backdoors

One practical problem confronting any attacker is how to control his attack program. If we assume that getting a usable attack program or exploitable backdoor in a voting system is expensive, in terms of effort or risk, then a sensible attacker will want to reuse the attack program if possible. However, this must be balanced against the additional risk and effort needed to get a more flexible program into the voting system.

There are a number of broad strategies an attacker may have for controlling his attack program, including:

- One-step ("fire and forget") attacks: In a one-step attack, the attacker puts the attack program into the targeted machines, and has no further contact with the machine. This class of attack minimizes conspiracy size, since the attacker need not get anyone else involved in the attack to make it widespread. However, these attacks have little flexibility, and may require attack programs sophisticated enough to determine which candidate is to be favored in the election fraud from the ballot definition files.

  - One-shot attack programs—in this case, the attacker constructs an attack targeted at a single election, which will go dormant or delete itself if possible at the end of that election.
  - Persistent bias attack programs—in this case, the attacker creates a program which will attempt to bias future elections in a specific direction (probably toward a specific party, since individual candidates and questions will not be around for that many elections, in general).

- Two-step attacks:

  - Reusable attack programs—in this case, the attacker builds a program which allows some form of "remote control" to activate and/or control its tampering. This allows the attacker to reuse the same attack program in a flexible way many times, but requires a more complex and sophisticated kind of program, and also requires some kind of additional control channel into the voting machines.

  - Reusable back doors—in this case, the attacker builds in (or leaves) an easily-exploited weakness in the voting machine software, which he knows how to use to install an attack program. The attack program may be any of the above kinds.

In terms of economics of the attack, the reusable choices are much better for the attacker. Both of the patterns of attack in these choices exist for programs used to attack real-world computers; some attackers install new backdoors to allow a later compromise of the machine as needed, while others install programs allowing them to simply send commands to the compromised machine.

3.2. One-Step Attacks

The most straightforward software attacks are one-step attacks: the attacker writes, tests, and inserts the attack program into the voting system, and has nothing more to do with the voting system.

3.2.1. One-Shot Attack Programs

A one-shot attack program is targeted at a single election. When the election is over, the attack program will either go dormant or (if possible) delete itself. Because this kind of program is targeted at a single election, it can be relatively simple. It may be developed before or after the ballot definitions are produced, but it is always targeted at a single election. It may thus use candidate names or other indications to decide how to change votes, and it may trigger on exactly one date at exactly one time.

The main limitations of this kind of attack are:

- The attacker spends all the resources to develop and insert the attack program into the voting system, and he gets to use it only once.
- There is no chance for the attacker to control the attack program's behavior. Thus, he cannot keep it from triggering in places or circumstances which may reveal its existence, except for whatever guidance he provides it in its design.

If the attack program is inserted into the voting system before the ballot definitions are specified, it must be able to determine, from the ballot definition file and other system information, which ballot question it must affect, and in which direction. We assume here that the program either can determine this from the ballot definition file or the onscreen display. It is sometimes disputed that an attacker could write an attack program of this flexibility. We find this claim unconvincing, given the remarkable sophistication of some real-world attacks. However, we will point out that:

- The attack program’s required sophistication falls rapidly as the attack grows more targeted—it is going to be much easier to design an attack program to work in one county in Maryland than to work all over the US.
- The attack program is targeted at a single election, so that the name and party affiliation of the favored candidate is probably known when the program is written.
- If the attack program is finished and inserted after the ballot definitions are made available, then it need not be sophisticated at all.

This kind of attack is probably best inserted at a local level. If it is inserted into all voting machines of a certain make all over the country, it should detect an unfamiliar ballot definition file or format, or a state or county name on the ballot, and never trigger unless these look right.

I expect that this attack is a reasonably likely one to be used on a local or statewide level. The more broadly the attack is applied, the more likely it is to be detected, due to unforeseen software bugs, interactions with other options unfamiliar to the attack program’s author, etc. The best way to control this kind of attack is to selectively install the attack program only on a small subset of voting machines, using either physical access, network access, or the ability to install invalid or tampered-with software patches.

3.2.2. Persistent Bias Attack Programs

A somewhat more efficient use of the attacker’s resources may be to build an attack program which provides a systematic small bias for the party of the attacker’s choice. The attack program must detect the political party for which someone is voting and introduce a slight bias. The two most obvious ways to do this are:

- Trigger only on straight-ticket votes; change some fraction of straight ticket votes from their original party to the different party.
- Trigger when the attacker’s preferred party is losing on a specific voting machine, increasing the error rate (for example, by messing up the alignment of the touch screen or occasionally skipping a ballot question).

Again, this class of attack suffers from a lack of flexibility, though it is so broad in impact that that flexibility is not so essential. Another problem with this attack is that an attacker motivated by greed probably cannot get paid for it; while many people would broadly like to see one party or the other do better, a broad improvement for the whole party may be hard to get a single person to pay for. (By contrast, the one-shot attack program is probably relatively easy to get paid for, in the sense that there’s a single beneficiary.)

I expect this kind of attack program is the least likely to be used in practice.

3.3. Two-Step Attacks

In this section, we describe attacks that require two stages—creating/planting the attack program or backdoor, and exploiting it. These are inherently more complex in operational terms, but they are also enormously more flexible.

In a two step attack, there are typically two different insiders involved. The first insider must insert the attack program or backdoor; the second must exploit it. To affect many polling places, counties, or states typically requires many insiders sending control information into the voting machines. On the other hand, the attacks become much more flexible, and an attacker who inserts the attack program into the system can in principle make money by selling access to a corrupt politician or campaign manager.

3.3.1. Attack Programs with Remote Control

Attack programs that provide the attacker some kind of remote control over the compromised machine are widespread. (This is basically how bot networks work; the attacker who has taken them over has some way of controlling their future actions.)

In an attack program for a voting system, the attacker wants to be able to exert control over:

- Which machines with the attack program installed trigger;
- What changes are made to the election outcome; and
- What additional conditions are checked for by the attack program prior to triggering.

Depending on when the remote control messages are sent, the attacker may already know things like the complete ballot definition file contents, making it much easier to tell the attack program how to change votes. The attacker is also likely to know any recently-announced countermeasures, such as parallel testing or hand-recounts of some paper-based machines. He can take this into account in his commands to his attack program.

There are really two broad categories of remote control messages to consider:

- Commands to the attack program. These can in principle be very low bandwidth messages, and can be hidden stenganographically in a variety of files and other messages.
- New programs to install. These are maximally flexible, but require that the attacker have a fair amount of bandwidth available to the machines being controlled. These are considered in the next subsection, on reusable back doors.

The attacker’s major operational problem with remote control of his attack program is finding an available channel over which to communicate his commands to his attack program. Unlike compromised desktop PCs, voting machines are seldom directly on the internet, waiting for an inbound connection from the attacker. Instead, they are usually not powered on at all, and when they’re on, they are likely in a somewhat restricted environment. Some broad classes of command channel include:

- Voter or poll worker interaction with the machine through its normal interface. This provides very limited bandwidth, but may be useful for a “secret knock” to activate attack behavior. Because a human must interact individually with each voting machine, attacks using this technique require reasonably large conspiracies. However, if the conspirators are simply voters, it doesn’t require especially highly-placed conspirators. Further, in some cases, the conspirators are simply told to vote a certain way; they don’t even need to know what they’re doing. A simple secret knock may not require any additional insider access after the attack program is inserted, but it has limited bandwidth; the most likely use for such a secret knock is simply to turn on the attack behavior.

  The secret knock can be almost anything that can be done through the normal user interface, so long as it is extremely unlikely to happen by chance. For example:

  - Touching several specific parts of the touchscreen simultaneously or in a specific order;
  - Voting for a specific pattern of candidates. (Note that this does not require that the attack program knows which pattern of candidates will be available ahead of time; see the discussion of steganographic techniques below!)
  - Voting for a specific write-in candidate; or
  - Mistyping a password or PIN a certain number of times when trying to log in.

- Configuration files for the election, including ballot definition files, audio ballot files, etc., can contain hidden instructions. This allows a very powerful attack, because the conspiracy size is potentially two people: one person to write and insert the attack program and one person to produce a file for the election which can be reviewed by anyone without detecting anything amiss. This also allows the attack program to simply read instructions about which ballot questions to tamper with, and in what directions, from someone who already knows the full contents of the ballot definition file.

How much information is needed to control an attack program? Let’s assume the attack program needs to know one ballot question to change and in which direction, and needs to trigger only when it’s told to.

- A checksum on the command of 20 bits leaves about a one in a million chance of incorrectly triggering.
- If there are no more than 128 ballot questions, and no more than 8 choices we might want to bias the machine towards, then an additional 10 bits are needed to specify them.
- With more bandwidth, we can embed further information. For example, a 16-bit additional command can specify an exact time to trigger, starting

at any hour, for the next seven and a half years.

How hard is it to embed such control information in a file? The fine details depend on the details of the file format, but a few parameters are easy to see:

- The creation time on a file will routinely include about 16 bits of choice for the attacker. If the attacker can choose the creation date on two such files, he can embed 32 bits, enough for our simplest attack control.
- Existing off-the-shelf steganographic programs for audio and picture files allow embedding of thousands of bits in ways that are not detectable by humans.
- If an attacker can choose many small variations in a ballot definition file, e.g., by adding an extra space character or not in each of 100 entries, he can produce a lot of different variations. For the example of the space character, he can embed about 100 bits in the ballot definition file. By using some checksum or hash function in his attack program, he can simply vary the ballot definition file in unobtrusive ways at random until he finds one with the right 32-bit CRC checksum, and use the CRC checksum as the command.

In general, blocking all possible covert channels into some program or device is extremely difficult. Countermeasures which would overcome any of the above techniques would still allow variations on them to be carried out. We thus come to the conclusion that an attacker who can provide one of more ballot definition files to the voting machine is very likely able to embed detailed commands to an already-present attack program on the voting machine, with almost no possibility of detection. Note that this requires a very specific kind of insider access—someone with the power to supply or alter at least one of the ballot definition files must be in on the attack. However, there will be nothing incriminating about the files; the insider embedding commands for the attack program will be able to give his files to uncorrupted observers without fear of discovery. The original author of the attack program will presumably give each conspirator a program to use in embedding commands to the attack program, so no great technical sophistication is assumed for the insider.

- Any network access for the voting machine during machine setup, testing, voting, or even at the end of voting, but before results have been reported, can be used to give the attack program detailed commands about how to tamper with the election results.

The most natural way for this to work is for the attack program to set up a program which is silently listening on some port for an inbound connection, and which accepts the connection and accepts commands. For this kind of command channel, the attack program might simply wait for a new uploaded attack program and then install it, or might accept a small sequence of commands as described above. This kind of command channel is widely used in bot networks today.

In this case, the second step of the attack can be done by someone with no insider access, simply by carrying a wireless-enabled PDA in his pocket while voting, or by establishing communications with the voting machines from outside the warehouse in which they are being configured. (Note that normal range limits on wireless access are for standard wireless networking hardware—larger antennas and better equipment can provide a substantial improvement in range—in some cases, 802.11 access has been achieved at a range of several miles!)

Even in the absence of such a powerful channel, however, the wireless network can be used as a signaling device. Among the obvious covert channels available are:

- Precise timings of ping or other inoffensive packets arriving on the network.
- Refused connection attempts.
- The names of networks broadcasting their presence.

Again, this doesn't require any insider access for the second step.

3.4. Reusable Back Doors

An alternative technique for attacking the voting system in software is for the attacker to insert some subtle bug into the program, which will allow a fairly easy takeover of the machine later. This has the enormous advantage that getting caught doesn't mean going to jail or getting fired, it merely means having to fix a subtle bug. A capable attacker who expects a competent code review will insert a dozen subtle bugs in the program, each allowing a silent takeover of the voting system software after the fact.

Instead of control channels, we must now consider future attack channels. The best of these are probably:

- Configuration and other files. The programs reading the files can include some kind of buffer overrun, or some unusual-looking escape sequence that gets part of an input string from a file out to a command shell, PERL interpreter, or some such thing.
- Software patches. The programs doing any verification of the software patches prior to installation can have a subtle bug by which the attacker can bypass that verification step.
- User interface. The program can have a subtle bug which makes it possible to get from a low-privilege user interface to a command shell running as root or administrator, and thus to change settings or install software.
- USB and device drivers. The COTS USB driver can have a bug which allows a tampered USB device to take the voting machine over.

Note that for all of the above, the second step of the attack requires insider access—either physical access to the voting machines to be compromised, or the ability to write configuration files or provide software patches to the machines being attacked.

- Wireless and wired networks. The programs that deal with the network access, either voting-specific programs or COTS programs, can have embedded attacks as described above.

In this case, the second step may be carried out by someone without any special access. For a known vulnerability with a known attack program to be inserted, the second step can be carried out by a conspirator carrying a wireless-enabled PDA running a program to take over as many voting machines as possible with this vulnerability. This is basically what is done in recent Bluetooth viruses, which seek out vulnerable devices via Bluetooth and attempt to infect them when they appear. Network worms also use known vulnerabilities to carry out automated attacks.

4. Attack Points

Modern voting machines typically have a lot of software and files on them, and provide complex interfaces for human users and other machines. An attacker needs to find a point at which he may insert his attacking program without detection. Depending on the attack point, we can determine a great deal about the nature of the attack; an attacker who tampers only with a few machines' software can write a very simple attack program, based on a thorough knowledge of election procedures, local ballot design, etc. On the other hand, an attacker who tampers with a whole line of voting system software by a major vendor must either make use of a two-stage attack, or must write a very sophisticated attack program to avoid detection and correctly tamper with the election in a huge variety of circumstances.

4.1. Original Voting System Software and Configuration

The original voting system software is developed by the vendor, with use of COTS software and tools. It is then provided to a testing lab for some level of checking. We may assume that the testing lab will not pass voting system software with obvious attack program behavior (e.g., a reachable menu screen asking the user how he wants the election results cooked).

However, there are a number of ways that an attack program might hide within the original voting system software: (This is by no means an exhaustive list!)

- The attack program could be part of COTS software which was purchased for use on the voting system.
- The attack program could be inserted into the executables and libraries after they have been built from reviewed code.
- The attack program could be hidden within the operating system using rootkit-like techniques, or perhaps a commercial rootkit for the underlying operating system.

- The attack program could be stored in some data file which is not reviewed, but which is read by a program with a subtle bug of some kind, allowing the program in the file take over the program reading it.

Further, we are deeply skeptical of the ability of the testing labs to review all the software in the voting system carefully enough to catch all possible attacks. Even if obvious attack behavior doesn't remain, either intentional, subtle bugs or subtle attack behavior (e.g., messing up the touch screen alignment after certain user interactions from voters) may still remain despite the testing lab review.

Finally, it's worth noting that tampering with the software in the initial voting system is not limited to programmers working for the voting system vendor. COTS software writers, who may themselves be contractors or subcontractors of the original company from whom the COTS software was purchased, are in an even better position than voting system programmers to insert an attack program. This is especially true for drivers written for devices that are mostly used for voting systems. Further, anyone able to get access to the voting system software either during design or after it has been reviewed and before it has been installed on the voting machines may install an attack program. This might include people with full access to the software during development, storage, or testing.

4.2. Software Patches and Updates

COTS software often has patches and updates which are required for security. Voting software can also require updates, either to fix bugs or to extend functionality in some way, e.g., by supporting more assistive technology or a larger set of screen characters for alternate-language voting. This is an obvious attack point if the updates are not secured. The attack program may be inserted by someone working for the COTS software vendor, or by someone working at the voting system vendor, or the election official handling the installation of patches and updates.

4.3. Configuration Files and Election Definitions

If the voting system software is vulnerable to attack, an attacker may be able to take over the machine by improperly formed files. (For reference, a very successful e-mail virus used a flaw in the WinZip engine on many people's PCs to mount an attack of this kind.)

4.4. Network Communication

Some voting systems use wireless or wired network connections. If there is a vulnerability in the configuration of the voting machines, then this can allow an attacker to insert an attack program.

4.5. Device I/O

Some voting systems involve the use of an external device such as a memory card, printer, or smart card. In some cases, access to these external devices has allowed attacks to be demonstrated in the laboratory. (The RABA report gives one example, and the Diebold optical scan attack gives another.)

This is not meant to be a complete listing, and it necessarily leaves out a lot of detail to cover so many different machines. However, it is important to recognize the large number of possible attack points. In general, we expect that two-step attacks make the most sense to apply at the top, either in the original voting system software or in patches sent around to many different voting machines. The remote control aspects of these attacks makes it possible to control the attack, so that it doesn't trigger in obviously unreasonable ways and get detected. On the other hand, the more local attacks fit nicely with a one-shot attack; the attack programs can then be very simple and focused on a single election in a single state or county.

5. Avoiding Discovery

One of the most basic problems for a software-based attack is how to avoid detection. The tampering program must avoid discovery to successfully alter the election. Also, the attacker will have a strong interest in avoiding detection, since an investigation may determine that he is responsible for the attack.

5.1. Insert, Delete, or Modify Votes?

In most cases, the most effective way to tamper with an election will be to change votes that have actually been cast; this avoids introducing a disagreement between the number of votes reported by the voting machine and the number of registered voters allowed to vote. In the case of a DRE voting system, changing votes electronically changes all the records of the voters' intent which are formally available to the voting system, and so this kind of attack cannot be directly detected by comparing the electronic totals with other records. In the case of other voting systems, such as DRE+VVPAT or PCOS, the attacker must also tamper with the paper records or prevent their being cross-checked against the electronic records.

By contrast, inserting or deleting votes introduces a disagreement between the number of registered voters allowed to vote, and the number of votes recorded electronically.

Some attack scenarios may require a predefined sequence of electronic votes to be produced by the tampered software. In this case, it's not feasible to selectively change votes. However, the attack software can start with a predefined sequence of electronic votes to be stored, and record the first *N* votes from the sequence, where *N* is the number of votes actually cast on the voting machine.

5.2. Deciding How Many Votes to Change

An attack may also be detected by too strong a disagreement between informal numbers (polling data, for example) and reported election results, though it isn't clear what procedure would be needed to invalidate an election based on this kind of evidence alone. We think one of the most likely scenarios for this kind of attack to be detected is for some event to happen which radically changes the expectations of the election, just before the election takes place. The death, indictment, or complete discrediting of one candidate a few days before the election offer an opportunity for an inflexible software attack on a voting system to be revealed.

This leads to a few natural approaches for an attack program to minimize its chances of discovery:

- Where possible, there should be some way for the attacker to control which voting machines alter votes and for which races. The attacker should use this to minimize his attack's "footprint" while still leaving the attack likely to succeed. (This is discussed in a different context below.)
- Where possible, the attack program on the voting machine should change a fixed portion of the votes, e.g., move 5% of the votes for John Smith to Mary Jones, rather than simply reporting a preordained result. This avoids the situation where a dead or recently indicted candidate mysteriously wins a few precincts, while losing badly in all others, revealing the attack.
- The attack program should notice when the tampering is hopeless (e.g., when the election appears so one-sided that the benefit of improving the favored candidate's outcome is outweighed by the cost of increased chance of detection from implausible results. In that case, it should refrain from any tampering at all, since this implies a risk of detection with no corresponding chance of success.

5.3. L&A and Parallel Testing

Tampered software must avoid detection during testing. There are a number of techniques to use to ensure that testing does not detect the attack program.

- The attack program can note the time and date, and only trigger when the time and date are consistent with an election. This prevents detection during L&A and acceptance testing, but not during parallel testing. Further, a tester may attempt to reset the machine's clock; however, a resettable machine clock may open up other vulnerabilities in the voting system
- The attack program can observe behavior which is consistent with a test vs. with real voting. For example, if L&A testing in a given place is known to never go on for more than four hours, the attack program can refuse to trigger until the 7th hour of voting. Note that this is strongly affected by the nature of the attack program's operation, the nature of testing that is ever done, and the nature of voting in a specific place.
- The attack program can activate only based on some communication with the attacker or his confederates. For example:

    - Some specific pattern of interaction between the voter or election official and the voting machine may be used to trigger the attack behavior. This is often called a “secret knock.”
    - Any of the control communications channels described later in this document may be used to turn the attack behavior on or off.
- The attack program can wait for a remote interaction with the attacker before deciding whether to tamper with stored electronic votes.

5.4. Avoiding Event and Audit Logs

Tampered software must not leave telltale signs of the attack in any event or audit logs. In principle, this could be pretty difficult. However, this depends on the nature of the attack program:

- Tampered user-interface software may simply display the wrong things to the voter, while not causing any other system events. In this case, there will be no trace of the attack in the event log.
- Tampered driver software for storage devices or tampered BIOS can alter what is written to the storage devices.
- Tampered operating system or other high-privilege-level software may be able to entirely bypass the event logging mechanisms of the operating system or tamper with the logs after entries are made.
- Tampered operating system or other software may simply provide a different log to the outside world than the one stored internally, if the log is not stored on removable or write-once media.

5.5. Coordinating with Paper Record Attacks

When the tampering program must support the attacker tampering with paper records as well, it is often going to be useful for the attacker to be able to prepare replacement paper records before the voting is completed.

There are two interesting variations on this problem: First, when a DRE+VVPAT system is using a paper roll, the electronic and paper records must be identical. Second, when the paper records or ballots are kept separate, an attacker need only produce paper records that agree, not necessarily do so in the right order.

This coordination task can be solved in a number of ways:

- The attacker may simply wait until the electronic results are ready, and then print the replacement paper records. This raises some logistical problems for the attacker.
- If the attacker is in contact with the voting machine during the voting process, for example over a wireless network or via an exposed infrared port, the attacker can print replacement paper records as the tampered records are produced on the voting machine.

- The attack program can have a predefined sequence of votes, which it produces electronically and which the attacker also prints.
    - The attack program can decide based on its observed conditions whether to alter its electronic records or not. Thus, if the attack program observes a very one-sided election, or a voting pattern more consistent with parallel testing than with normal voting, it can simply not carry out the attack. The attacker must then determine which thing has happened, and replace the paper records or not depending.
- The attacker can communicate with the voting machine after the voting has ended but before the votes have been displayed to poll workers or sent to the tabulation center. In this case, the attacker can tell the voting machine what totals to report and store.

5.6. Forensics and Postmortem Analysis

Perhaps the hardest test for an attack program is some kind of forensic analysis. In this case, I imagine that an attack is strongly suspected, and competent people are analyzing the machine at great depth to find evidence of the attack program. In the most extreme case, this would involve making bit-level copies of everything on the machine, and taking the machine apart in a lab to verify that everything looks exactly as it should.

Drawing from experience in other areas, I expect that while a reviewer who doesn't know an attack exists will probably not find a competently implemented attack program, a reviewer who knows or strongly suspects that an attack has occurred, and who is allowed to destroy the machine in the quest for evidence, will probably discover that evidence.

The attack program has a number of techniques at its disposal to avoid detection by less thorough reviews, however, including:

- Hiding from operating system utilities attempting to scan files and memory, in the manner of existing stealth viruses and rootkits.
- Hiding the active part of the attack program in obscure places, such as data or configuration files read by programs running at high privilege levels (exploiting a weakness in those programs to take them over when they read an improperly formed configuration file), or driver software stored in EEPROM or flash memory.
- For one-shot attacks (see below), simply deleting all attack programs from the system after the attack is carried out, if this is possible. (This requires the use of some kind of secure delete, and even the secure delete will likely leave evidence of something interesting happening, but it is clearly possible.)
- For attacks based on backdoors, the whole attack may take place entirely in RAM, with no altered programs.
- For attack programs embedded in original COTS software or voting system software, scanning memory contents, even in a way that resists attack programs' tampering with what is seen, will not detect anything wrong.

After all this, I return to my initial assessment: a thorough, destructive analysis of the voting machine is quite likely to find evidence of an attack, especially if the broad direction of the attack is known or suspected from other evidence.

6. Conclusions

In this white paper, I have discussed a number of operational difficulties an attacker might have in compromising an election using some kind of tampered software on the voting machine. While the difficulties are real, they are not impossible to overcome, and I've tried to show at a high level how this might be done.

Software attacks are a major potential threat to voting systems because it is possible for them to be so well hidden that no outside observer ever notices the attack behavior, and because these attacks, unlike so many other potential attacks on voting systems, may be mounted with a very small set of people.