Report on US Voting Engineering

After the negative reactions to the 2016 election results by the Democratic party and the 2020 election results by the Republican party, I decided to look into our voting systems in some depth. As a very senior computer engineer who has led and implemented major computer systems for companies including Intel, Digital Equipment Corporation (DEC), and Lockheed Martin, I thought I might be able to offer a useful perspective on the topic. Never having looked into voting systems, I approached the problem with no preconceived notions.

Unlike the efforts to prove voting fraud, this report is focused on the engineering practices used to create our current system. To be clear, the audit and regression analyses that are looking for fraud are investigating the results produced by the product of the engineering practices. This report assesses the engineering that produced the voting systems. The use of Dominion artifacts should not be taken as the sole focus. The goal is to look for improvements to the process.

There can be no doubt that our voting systems are a target for both internal and external (foreign) attack. Every target has a breaking point, and it is no stronger than its weakest link. That is true whether the system is a military defensive position (like a fort) or an Internet application. Even Masada eventually fell. The question is: How truly vulnerable are our voting systems. Any intelligent commander will review his/her defenses and strengthen the weak links before they are breached. Voting systems must be approached with the same mindset.

The software component of our voting systems is only one link in the chain from a voter to its final tabulation. Although the scope of this report is primarily focused on the software engineering practices, these same practices apply to a wider range of engineering problems. The wider scope of a total voting system is addressed by project management.

Overview

Since Dominion Voting Systems were the target of a lot of ire, the logical place for a software engineer to start this investigation was to look for all the available data on code reviews for the Dominion software. Code reviews are one of the first steps of software verification. My hope was that I could satisfy myself that the engineering was satisfactory and go on to something more productive. Unfortunately, that was not the case.

From the initial code review audit, I could work both backward and forward to get as complete a picture of engineering truth as I could, without access to the actual hardware and software source code of the voting systems. As expected, the results of the code review audit were not totally satisfying; however, it did produce some nuggets of information. In particular, it provided links to look at the engineering steps prior to the code reviews as well as pinpoint some apparent weaknesses in the code itself.

There is no question that the Dominion software could be stronger, but it is but one link in the chain and perhaps not the weakest link. The weakest link may well be the process by which ballots enter the system. This is the classic “garbage in; garbage out” (GIGO) with which all computer nerds are familiar. It is a project management issue that is caused by a lack of firm requirements for the voting system. Fortunately, the *Project Management Body of Knowledge* (PMBOK) is consistent with the Software Engineering Institute’s *Capability Maturity Model Integration* (CMMI), so we can view the whole system as a continuum.

The current *Voluntary Voting System Guidelines* (VVSG) is not a robust set of requirements that can be rigorously verified. Under the current system, rogue states can actually subvert the will of their people and also disenfranchise the citizens of other states in the process. The Election Assistance Commission (EAC) was created by the Help America Vote Act of 2002 (HAVA). HAVA was passed in response to the contentious 2000 presidential election. By commissioning the VVSG as guidelines rather than requirements, the EAC is respecting the constitutional power of the states in the 12th amendment. There is a constitutional solution to the problem.

What is needed is for Congress to call a convention of states like the “Colorado River Compact” (not an Article V convention). By so doing, experts from the several states could hammer out real requirements that could be verified. The VVSG could become the VSR (Voting System Requirements). This would elevate the certification process to a uniform level without disrespecting federalism. By recurring review, like the Colorado River Compact, exposure of weak links can be addressed, and the states can hold each other responsible. The first engineering step is always the most important one, and that’s what this would be.

Requirements

In laymen’s terms, good engineering practice starts with a set of requirements and ends with verifying that those requirements have been met. Every software development organization has its own methodology built upon development stages to satisfy themselves that they have built what was required. The degree of rigor required to satisfy customer(s) varies according to the product’s use and the nature of the customer base. Companies like Intel and Oracle compete for market share. Once a sale is made, the technical support for the products is equally important as the engineering that built them. On the other hand, the engineering for human flight or to send a satellite to another galactic body is unforgiving. National pride can suffer with a big waste of taxpayer money. Even worse, people can die – especially with human flight.

It became immediately apparent that there were not enough artifacts to make a solid conclusion about the maturity of the Dominion software, but there were enough to realize that the engineering that had been done was not adequate for a system that has the repercussions of a voting system. When satellites are launched, there is danger to perhaps tens of people on the launchpad and surroundings. Those systems are required to meet CMMI level 3 management and use IEEE 1012 standards. Human flight can threaten the lives of hundreds and must meet CMMI level 5 engineering and meet DO-178B standards. It makes sense that voting systems that can adversely affect the lives of literally all Americans, present and future, should far surpass the standards for human flight. They don't even come close.

Software Audit of Dominion Democracy Suite

The first document produced for this study was the "Audit of Code Reviews for Dominion Democracy Suite." It was discovered in the reviews from SLI Compliance that the Dominion Democracy Suite is composed of over 1.5 million lines of C/C++ and over 1.8 million lines of C#. Having written about that many lines of C/C++ in my career as an operating systems kernel developer, I had two immediate reactions to this:

1. What on earth are they doing in a voting system that requires that much code? I read the entire VVSG to get a handle on the "requirements," and there is nothing there that suggests this magnitude. The main thrust of a voting system *should* be simple tabulation that should really be at most a couple hundred thousand lines of C/C++ and perhaps the same amount of C# - even with support for a Windows GUI or CGI browser interface. This means there is a lot of unnecessary complexity that is not driven by actual requirements. That increases risk.
2. I give odds that the system has several "memory leaks." Even the best developers can fall prey to a memory leak when writing C/C++. Memory leaks are one of the prime security exposures for a hacker to exploit. One coding convention used to combat this kind of error is to use a "lint" tool that does static code analysis. The newer "splint" does a more effective job of finding potential memory leaks, but it is prone to false positives. A developer can use "tags" to override the tool, so an insider threat can easily defeat the tool. A review of Dominion's coding conventions showed no evidence of the use of such a tool. Even worse, there was no requirement that compiles must be free of warnings. Since the VVSG does not call for compiles to be free of warnings, there is no reason to believe that Dominion enforces these conventions. This is unwise.

The first artifacts found were performed by a small company (around 50 people) in Austin Texas called @Sec. Surprisingly, this report was to be the most informative document on the actual code that I would find. In all honesty, the 30ish findings in their document were very light for a code review of a code base this large. @Sec had reviewed Democracy Suite 4.14. Apparently, they had performed the code review on behalf of Wyle Laboratories, who certified the software on 18 July 2013. The real problem is that the software was certified despite the fact that it clearly violated some sound requirements in the VVSG. In particular, it violated the following:

- Encryption keys violations
  - Hard-coded
  - Unencrypted keys stored on disk
  - Weak keys and cryptographic hashes
- Use of mixed-mode arithmetic. There have been accusations of Dominion systems being configured to use fractions to weight individual votes differently. This is a reasonable implementation for an HOA, where it is fair for differences in investment to garner more or less influence; however, such a thing should be

strictly eliminated from public, political elections. The VVSG calls for the removal of what is commonly referred to as “dead code.” If a voting system depends upon its configuration to avoid executing HOA-style voting inequities, that code is resident and becomes “dead.” Such code must be eliminated in a “trusted build.”

- “Complex branching structure” and bad exception handling. This is a particularly worrisome violation because it can mask a lot of potential exploits – both internal and external. Since later code reviews by Pro V&V only look at modifications from the base, there is no reason to believe that a rewrite of this ever happened.
- Privilege escalation (this is very dangerous. Hackers look for these opportunities)

Any of these problems can have severe consequences, but @Sec graded only the poor quality of keys as high severity. Other findings that could produce severe consequences were downgraded because they coupled severity with probability. In particular, they undervalued the severity of an insider threat and downgraded problems that they thought only an expert could exploit. Considering probability for a commercial product is not unreasonable; however, for products with the serious impact of human flight and above the situation changes. Old school IBM testers had a useful slogan, “if the probability ain’t zero, it’s 100%.” This is the test that needs to apply to assessing the severity of an issue for voting systems. (Taking this to the wider focus of the entire voting chain, this means that *if fraud can happen, it will*.)

The current Dominion systems are version 5, but the version 4 artifacts revealed that certification did not mean the software was devoid of severe problems. The California test report for version 5.2 documented the potential for “SQL injection.” This was further demonstrated by Col. Philip Waldron in current versions of the Dominion software. The VVSG specifically forbids SQL injection because it is a dangerous exposure, yet the Dominion systems were certified. There were two types of documentation for version 5:

1. SLI Compliance and Pro V&V reviews
2. Individual state reviews

SLI Compliance appears to have taken @Sec’s place for code reviews for version 5. It appears that Pro V&V replaced Wyle as the certifier for Dominion. The individual state reviews were more focused on the later stages of verification – actual acceptance testing. This enables the states to ensure their individual laws are observed. Both the certifiers and the states received a Technical Data Package (TDP) from Dominion that may have included a Requirements Tracing Verification Matrix (RTVM) to ensure VVSG requirements were met before they added their own verification.

It is worthy of note that Pro V&V has less than ten employees. SLI Compliance has fewer than fifty employees in Wheat Ridge, Colorado. It is unlikely that either of these companies have the resources to review over 3.5 million lines of C/C++/C# code in anything approaching the rate of change of a system of that size. Realistically, effective code reviews need to be done as the first version of such a large piece of software is being created. It appears that both SLI and Pro V&V addressed the problem in the same way:

- Use of automated source code review tools
- Manual review of changes only

Automated source code tools are good for catching some errors and enforcing conventions, but they are not great for catching security threats. Manually reviewing changes is good, but it is only as good as the breadth of knowledge the reviewers have about the whole code base. That said, this is the typical way code reviews work. As code is developed, it is reviewed. There is no real way to assess the degree of scrutiny and talent of the reviewers without a set of findings. For version 5, neither SLI nor Pro V&V stated anything more than “no code issues were found” in the documents I could find. Call me a skeptic, but this is a completely unsatisfying statement that suggests that the reviews were little more than a box checked.

Dominion Coding Conventions

Coding conventions are not imperative for software development, but they can indicate the professionalism of the organization. The underlying theory is that any developer in the organization knows there will be certain things in and about the source code. They can be so tyrannical that they either forbid or demand a tab character as opposed to a space, or they can be so lax that it behooves a developer to run the source file through what is commonly referred to as a “pretty printer” to massage it to his/her taste. I have worked in both environments, and both extremes are annoying but have no real effect on the final product.

The Dominion Voting *Democracy Suite ImageCast C++ Coding Standards* is a 28-page document for Dominion’s coding conventions, published in February 2017. As stated above, there is no evidence in the fifteen pages of actual conventions that compiles either need to be free of warnings or checked by a lint tool. From almost forty years of writing C/C++ code, I can say unequivocally that these are the two most important conventions that a software organization can adopt. All other conventions are either designed to avoid potential mistakes from future modification or are merely cosmetic. Being free of warnings and the use of static analysis tools like lint verify the existing code as it is.

There are really only five pages covering actual coding conventions. Pages 7 through 15 talk about comment conventions. To be clear, computer processors do not read comments. People read comments. The VVSG actually calls for certain elements of this, and Pro V&V covers it in their code review. Dominion specifies the use of *doxygen*. Doxygen is a tool that extracts tagged comments to create documents from within the code. This theory of development goes back to the 60s with a tool called *troff*. The idea is that documents will stay up to date if they are extracted from comments in the code. The responsibility for creating technical documentation shifts to the developer rather than a tech writer. The implication is that Dominion does not employ tech writers to create professional documentation. This is reasonable for a small organization. It is also common for open software to use such a tool.

Development Process

I could not find any details on Dominion’s development process, but a general caution about development processes is in order. The VVSG encompasses project management in addition to software guidelines. The PMBOK is the standard. “Agile” methodologies dominate the tech world today. Most organizations follow some adaptation of the “scrum” methodology, and the PMBOK has been updated to include such a methodology. Scrum is targeted to rapidly accept changes in requirements. This may not be the wisest approach to a system that needs to be secure, because old requirements are always implemented with a set of assumptions that can create a security hole when a new requirement is introduced.

Trusted Build

The individual state acceptance documents refer to a "trusted build." The SLI Compliance test report describes a trusted build in its software compliance section. There is no doubt that the trusted build is controlled in the process of being certified. What is not clear is how updates are controlled, but it appears to be contractual. The VVSG calls for periodic updates, but it does not appear that every one of these updates goes back through VSTL certification. If not, the individual state is essentially acting as a VSTL certifier whenever a "trusted build" updates its installations. But the state acceptance tests are targeted toward the latter stages of verification, so there may be no review of source code modifications. This is a project management question.

The Larger Picture

From investigating all the artifacts I could find for code conventions, code reviews, and acceptance testing of the states, the Dominion Democracy Suite has undergone a certain level of engineering rigor. On the surface, it appears adequate for a normal commercial application; however, whether it is rigorous enough for a voting system is an open question. Talent can make up for process; however, process cannot make up for talent. When it comes to a system that will endure attacks, both process and talent need to be as strong as possible. Neither Dominion nor the VSTLs (Pro V&V and SLI Compliance) appear to have taken process to the level of human flight. It is unfair to attempt an assessment of the engineering talent of Dominion.

Voting system requirements and engineering process need to exceed that of human flight. Much more is at stake. This means that the development process needs to be CMMI level 5 compliant. Impact must be considered on its own without being watered down during certification. Rigorous requirements need to guide the entire voting system, not just software. No link in the chain can be overlook.

Recommendations

The Constitution is clear that the states control the voting process, and there is no reason to change that with a constitutional amendment. The current HAVA and EAC were a step forward in response to the 2000 election; however, there are many issues that require formal state input. Congress first called the states together to create the Colorado River Compact in 1922. The Compact was created by a convention that included technical experts as well as lawmakers. Each state was basically represented by one of each. Just as the southwestern states needed to agree to apportionment of the Colorado River, the states today need to agree on requirements to secure honest election results uniformly across all states. Congress should call a similar convention to address the shortcomings of the VVSG and create a robust requirement set. Below are some suggestions for consideration by such a body.

To be clear the VVSG addresses more than software issues. It addresses the entire voting chain. The states do take the printing of ballots seriously, but from discussions with my county clerk and recorder, there is room for improvement. Just as I point to human flight for software engineering standards, ballots should be treated like currency. There is no excuse for counterfeiting of ballots to be less controlled than dollar bills.

Weaknesses in software can best be observed when the system is in action. I have witnessed demonstrations of stacks of ballots being fed into a Dominion system multiple times and not being rejected. There is an obviously missing requirement here. No ballot should be accepted more than once. Building on the treatment of a ballot as a dollar bill, a serial number should be checked to ensure against such duplicates. The serial number itself can embed a state and precinct code to keep the check local.

Dominion supports "adjudication." If I somehow don't properly mark my ballot, I don't want anyone deciding what my intended vote should be. Personally, I would rather it not count than be changed by someone else. A desire for good government dictates this philosophy. Software engineers will always try to "improve" things. Sometimes those improvements just make a mess. This is not to ignore that ballot reader errors can occur. Those errors should be bound by a requirement. I would like to see an "election compact" address this issue. It should be good theatre.

The requirements for the election system need to be brutal. The aforementioned DO-178B standard severely restricts normal programming techniques. Efficiency and elegance are sacrificed for reliability. This philosophy is currently not in the VVSG but should be put on steroids. To pinpoint an example, consider one of the major functions we witness during elections. The poll workers know the state of the election as they are tabulating ballots. This opens up a big insider threat. Even worse, these results are broadcast to the outside world thereby opening an even bigger threat. The tabulation process should be like a piggy bank. The ballots get processed, but no one sees the results until the bank is broken open. The inner workings do need to be completely traceable. The media would hate it because they wouldn't get to spend the night torturing the American public. So what?

REFERENCES

- "Voluntary Voting System Guidelines VVSG2.0." EAC, n.d. https://downloads.regulations.gov/EAC-2020-0002-0001/content.pdf.
- "Source Code Review Dominion Democracy Suite 4.14-A Voting System." California Secretary of State, n.d. https://votingsystems.cdn.sos.ca.gov/dominion-voting/democracy-suite.pdf.
- "RESULTS OF DOMINION VOTING SYSTEMS DEMOCRACY SUITE 5.5A." Pennsylvania Secretary of State, n.d. https://www.dos.pa.gov/VotingElections/Documents/Voting%20Systems/Dominion%20Democracy%20Suite%205.5-A/Dominion%20Democracy%20Suite%20Final%20Report%20scanned%20with%20signature%20011819.pdf.

- "US Election Assistance Commission Discrepancy Report for 5.5A (Attachment C)." EAC, n.d. https://www.eac.gov/sites/default/files/voting_system/files/Attachment_C_-_Dominion_D-Suite_5.5-A_Discrepancy_Report.pdf.
- "Dominion Democracy Suite 5.10 Voting System Software Test Report ." California Secretary of State, n.d. https://votingsystems.cdn.sos.ca.gov/vendors/dominion/dvs510software-report.pdf.
- "Voting System Examination of Dominion Voting Systems Democracy Suite 5.5-A." Texas Secretary of State, n.d. https://www.sos.texas.gov/elections/forms/sysexam/oct2019-mechler.pdf.
- "Test Plan for EAC 2005 VVSG Certification Testing Dominion Voting Systems Democracy Suite (D-Suite) Version 5.5-CVoting System." Pro V&V, n.d. https://www.eac.gov/sites/default/files/voting_system/files/Dominion%20Voting%20Systems%20D-Suite%205.5-C%20Test%20Plan-Rev.%2001.pdf.
- "Colorado Requirements Matrix." Colorado Secretary of State, n.d. https://www.sos.state.co.us/pubs/elections/VotingSystems/ClearVote/ColoradoRequirementsMatrix-ClearVote1-4-1.xls.
- "Uniform Voting System Request for Proposal." Colorado Secretary of State, n.d. https://www.sos.state.co.us/pubs/elections/VotingSystems/RFI/proposals/ClearBallotColoradoUVSProposal.pdf.
- "Democracy Suite ImageCast C++ Coding Standard." Colorado Secretary of State, n.d. https://www.sos.state.co.us/pubs/elections/VotingSystems/DVS-DemocracySuite511/documentation/SD-CPlusPlus-CodingStandard-5-11-CO.pdf.
- "Dominion Voting Java Coding Standard." Colorado Secretary of State, n.d. https://www.sos.state.co.us/pubs/elections/VotingSystems/DVS-DemocracySuite/documentation/dvs_Java%20Coding%20Standards.pdf.
- "VVSG 1.0, Vol. 1: U.S. Election Assistance Commission." VVSG 1.0, Vol. 1 | U.S. Election Assistance Commission. Accessed January 4, 2021. https://www.eac.gov/documents/2017/03/15/vvsg-10-vol-1-voluntary-voting-system-guidelines-vvsg.
- "Certification Test Report Democracy Suite 5.5A." EAC, n.d. https://www.eac.gov/sites/default/files/voting_system/files/Dominion_Voting_Systems_D-Suite_5.5-A_Test_Report_v1.1.pdf.
- "Test Report for EAC 2005 VVSG Certification Testing Dominion Voting Systems Democracy Suite (D-Suite) Version 5.5-C Voting System." EAC, n.d. https://www.eac.gov/sites/default/files/voting_system/files/Dominion%20Voting%20Systems%20%20D-Suite%205.5-C%20Test%20Report-Rev.%2001.pdf.
- "Dominion Voting Systems Democracy Suite 5.5C Certificate of Conformance." EAC, n.d. https://www.eac.gov/sites/default/files/voting_system/files/DVS%20DSuite%205.5-C%20Certification%20and%20Scope%2007-09-2020.pdf.
- "Dominion Voting Systems Democracy Suite 4.14A Certificate of Conformance." EAC, n.d. https://www.eac.gov/sites/default/files/voting_system/files/Scope_of_Cert4_14-A_1_FINAL_6_16_14.pdf.
- "SECURITYASSESSMENT SUMMARY REPORTFORDOMINION VOTING SYSTEMSDEMOCRACY SUITE 4.0." EAC, n.d. https://www.eac.gov/sites/default/files/eac_assets/1/28/Security%20Assessment%20Report.pdf.
- "DEF CON 27Voting MachineHacking Village." defcon.org, n.d. https://media.defcon.org/DEF%20CON%2027/voting-village-report-defcon27.pdf.
- "Source Code Review Dominion Democracy Suite 4.14-A Voting System." verifiedvoting.org. @Sec, n.d. https://verifiedvoting.org/wp-content/uploads/2020/08/democracy-suite.pdf.
- "Democracy Suite 5.5-A Certification Test Report–Modification Version 1.1." SLI Compliance, n.d. https://www.eac.gov/sites/default/files/voting_system/files/Dominion_Voting_Systems_D-Suite_5.5-A_Test_Report_v1.1.pdf.

- "Dominion Democracy Suite 5.2 Source Code Test Report for California." California Secretary of State, n.d. https://votingsystems.cdn.sos.ca.gov/vendors/dominion/ds52-sc.pdf.
- "Certificate of Conformance." United States Election Assistance Commission. Wyle Laboratories, n.d. https://www.eac.gov/sites/default/files/voting_system/files/Certificate%20and%20Scope%20DVS414.pdf.