resources with useless tasks that keep it busy; if the web server becomes too busy, it may be unable to respond to connections from legitimate users. We will discuss both categories.

In a network flooding attack, the adversary sends huge volumes of data in the direction of the victim, saturating the victim's network connection and making it impossible for legitimate users to contact the victim. SERVE's websites are at risk for this kind of attack; if their network connection is overwhelmed by a denial-of-service attack, then eligible voters will not be able to vote using SERVE.

As a rule of thumb, the robustness of a website against network flooding attacks is determined largely by the network capacity available to that website. For instance, a website with a 1 Gbps link to the Internet would be hard-pressed to withstand a 1 Gbps DDoS attack. Large e-commerce sites typically have a 10 Gbps link at most. In comparison, researchers have observed DDoS attacks with peak traffic rates in excess of 150 Gbps [MVS01,MPSSSW03]. It seems unlikely to us that SERVE could withstand such a high-volume DDoS attack.

In the second category of denial-of-service attack, the adversary sends many valid-looking requests to the victim in an attempt to overload the victim's computer and keep it busy with useless work. There are many opportunities for attacks of this sort, and it would be hard to anticipate them all. Instead, we will describe an example of one attack in this category, to give the general idea. Similar principles may apply to many other aspects of the SERVE architecture.

SERVE uses an SSL-protected website. However, SSL is susceptible to a denial-of-service attack. An adversary could send many requests to initiate new SSL connections, and the SSL protocol requires the recipient to perform a slow cryptographic operation (typically an RSA private-key computation) when responding to each such request. The exact performance depends on the security level provided, but with the fastest and lowest-security key sizes that are today considered acceptable (i.e., 1024-bit RSA), modern computers can handle about 100 new connections per second; hardware accelerators raise this number to thousands of new connections per second. Today's largest e-commerce sites can handle up to 15,000 new SSL connections per second. In comparison, an attacker might be able to initiate about 500,000 new SSL connections per second, based on the following assumption: It is plausible that an attacker could gather a "zombie network" of 10,000 slave computers, and each computer could initiate about 50 new SSL connections per second. Consequently, an attacker could generate 10 to 100 times more SSL traffic than the SERVE website is likely to be able to handle. Thus, a DDoS attack against SERVE's SSL web servers could render SERVE unreachable to voters and disrupt an election in progress.

Unfortunately, mitigating or responding to denial-of-service attacks is very difficult. Today's technology is not up to the task. For example, no good defenses against network flooding attacks are known on today's Internet. It may be possible to defend against the particular SSL attack we describe; however, defending against all variants of this scheme is difficult. As an attacker will attack the weakest link in any system, SERVE must protect against all possible denial-of-service attacks—a very difficult task.

In summary, we are concerned that, no matter how much energy is invested into defensive countermeasures, adequate protection against denial-of-service attacks is unattainable with the technology available today. No matter how careful the designers are, SERVE is unavoidably at risk.

4.2 The implications of denial-of-service attacks for SERVE

An attacker could mount a large-scale denial of service attack that renders SERVE's voting service unavailable on the day of an election. Those voting on Election Day would be unable to vote, calling into question the validity of the election.

Alternatively, network services could be knocked out or degraded for areas where a particular demographic is known to vote for a particular party. The outcome of the election could be swayed by such an attack. Detection of such a selective disenfranchisement attack would be possible, but it is not clear how to respond; once polls close, there may be no good choices.

Though today's absentee voting process already disenfranchises some voters, we fear that SERVE could make the problem worse, not better. We recognize that UOCAVA voters are having a harder time voting than they ought to, and there are reasons to believe that a significant number (perhaps 20-30%, according to

some estimates) of military voters fail in their attempt to vote absentee. However, SERVE runs the risk of exacerbating these problems. With SERVE, there is the possibility that the disenfranchisement rate could rise to close to 100%, if a denial of service attack is successfully mounted against SERVE. In addition, SERVE creates the risk of large-scale selective disenfranchisement, which is not present in today's absentee system. Large-scale selective disenfranchisement is especially problematic because it could be used to influence the outcome of an election.

One important difference between SERVE and in person voting is that eligible voters can vote at any time during a 30-day window starting 30 days before Election Day and extending until the close of polls on Election Day. If voters could be persuaded to vote early in this time window, the impact of denial-of-service attacks might be reduced: In the past, most denial-of-service attacks have lasted only for a few days, and when the attack subsides, affected voters could then vote, if the polls had not yet closed.

However, SERVE's 30-day window cannot be relied upon to defend against denial-of-service attacks. There are reasons to believe that a large proportion of the voting population will want to vote on Election Day. (See Appendix B for an example). This introduces the threat of *last-day denial-of-service attacks* in which the attacker mounts a denial-of-service attack starting on the morning of Election Day and lasting until polls close. Because responding to denial-of-service attacks takes time, it is likely that an attacker would be able to maintain a last-day denial-of-service attack all day long, so that the SERVE systems would remain unreachable for the entire Election Day. In such a scenario, any overseas citizen who had intended to vote on Election Day would be unable to vote through SERVE, probably would not be able to find any alternative way to vote before the close of polls, and thus would be disenfranchised.

We expect that last-day denial-of-service attacks would disenfranchise a substantial fraction of the SERVE population. There seems to be little that SERVE can do to defend against such attacks. For these reasons, we consider last-day denial-of-service attacks a significant threat to the security of SERVE's elections.

5. Conclusions

Our conclusions, based on the arguments in this report are summarized as follows:

a) DRE (direct recording electronic) voting systems have been widely criticized elsewhere for various deficiencies and security vulnerabilities: that their software is totally closed and proprietary; that the software undergoes insufficient scrutiny during qualification and certification; that it is especially vulnerable to various forms of insider (programmer) attacks; and that DREs have no voter-verified audit trails (paper or otherwise) that could largely circumvent these problems and improve voter confidence. All of these criticisms, which we endorse, apply directly to SERVE as well.

b) But in addition, because SERVE is an Internet- and PC-based system, it has numerous other fundamental security problems that leave it vulnerable to a variety of well-known cyber attacks, (insider attacks, denial of service attacks, spoofing, automated vote buying, viral attacks on voter PCs, etc.), any one of which could be catastrophic.

c) Such attacks could occur on a large scale, and could be launched by anyone from a disaffected lone individual to a well-financed enemy agency outside the reach of U.S. law. These attacks could result in large-scale, selective voter disenfranchisement, and/or privacy violation, and/or vote buying and selling, and/or vote switching even to the extent of reversing the outcome of many elections at once, including the presidential election. With care in the design, some of the attacks could succeed and yet go completely undetected. Even if detected and neutralized, such attacks could have a devastating effect on public confidence in elections.

d) It is impossible to estimate the probability of a successful cyber-attack (or multiple successful attacks) on any one election. But we show that the attacks we are most concerned about are quite easy to perpetrate. In some cases there are kits readily available on the Internet that could be modified or used directly for attacking an election. And we must consider the obvious fact that a U.S. general election offers one of the most tempting targets for cyber-attack in the history of the Internet, whether the attacker's motive is overtly political or simply self-aggrandizement.

e) The vulnerabilities we describe cannot be fixed by design changes or bug fixes to SERVE. These vulnerabilities are fundamental in the architecture of the Internet and of the PC hardware and software that is ubiquitous today. They cannot all be eliminated for the foreseeable future without some unforeseen breakthrough. It is quite possible that they will not be eliminated without a wholesale redesign and replacement of much of the hardware and software security systems that are part of, or connected to, today's Internet.

f) We have examined numerous variations on SERVE in an attempt to recommend an alternative system that may deliver somewhat less voter convenience in exchange for fewer or milder security vulnerabilities. However, all such variations suffer from the same kinds of fundamental vulnerabilities that SERVE does; regrettably, we cannot recommend any of them. We do suggest a kiosk architecture as a starting point for designing an alternative voting system with similar aims to SERVE, but that does *not* rely on the Internet or on unsecured PC software (Appendix C).

g) The SERVE system might appear to work flawlessly in 2004, with no successful attacks detected. It is as unfortunate as it is inevitable that a seemingly successful voting experiment in a U.S. presidential election involving seven states would be viewed by most people as strong evidence that SERVE is a reliable, robust, and secure voting system. Such an outcome would encourage expansion of the program by FVAP in future elections, or the marketing of the same voting system by vendors to jurisdictions all over the United States, and other countries as well. (The existence of SERVE has already been cited as justification for Internet voting in the Michigan Democratic caucuses.)

   However, the fact that no successful attack is detected does not mean that none occurred. Many attacks, especially if cleverly hidden, would be extremely difficult to detect, even in cases when they change the outcome of a major election. Furthermore, the lack of a successful attack in 2004 does not mean that successful attacks would be less likely to happen in the future; quite the contrary, future attacks would be more likely, both because there is more time to prepare the attack, and because expanded use of SERVE or similar systems would make the prize more valuable. In other words, a "successful" trial of SERVE in 2004 is the top of a slippery slope toward even more vulnerable systems in the future.

h) Like the proponents of SERVE, we believe that there should be better support for voting for our military overseas. Still, we regret that we are forced to conclude that the best course is not to field the SERVE system at all. Because the danger of successful, large-scale attacks is so great, we reluctantly recommend shutting down the development of SERVE immediately and not attempting anything like it in the future until both the Internet and the world's home computer infrastructure have been fundamentally redesigned, or some other unforeseen security breakthroughs appear.

We want to make clear that in recommending that SERVE be shut down, we mean no criticism of the FVAP, or of Accenture, or any of its personnel or subcontractors. They have been completely aware all along of the security problems we have described here, and we have been impressed with the engineering sophistication and skill they have devoted to attempts to ameliorate or eliminate them. We do not believe that a differently-constituted project could do any better job than the current team. The real barrier to success is not a lack of vision, skill, resources, or dedication; it is the fact that, given the current Internet and PC security technology, and the goal of a secure, all-electronic remote voting system, the FVAP has taken on an essentially impossible task. There really is no good way to build such a voting system without a radical change in overall architecture of the Internet and the PC, or some unforeseen security breakthrough. The SERVE project is thus too far ahead of its time, and should not be reconsidered until there is a much improved security infrastructure to build upon.

Acknowledgements

We thank Kim Alexander, Dr. Steve Bellovin, Lillie Coney, Prof. David Dill, Prof. Doug Jones, Yoshi Kohno, Prof. Deirdre Mulligan, Prof. Ron Rivest, Prof. Gene Spafford and Adam Stubblefield for helpful comments.

References

[AK96] Ross Anderson and Markus Kuhn, "Tamper Resistance - a Cautionary Note," *Proceedings of the Second Usenix Workshop on Electronic Commerce*, pages 1-11, November, 1996.

[Garman81] John R. Gamran, "The "bug" heard round the world," *ACM Software Engineering notes*, 6(5):3, October, 1981.

[HW01] Kevin J. Houle, George M. Weaver, "Trends in Denial of Service Attack Technology", October 2001.

[Kohno03] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach, "Analysis of an Electronic Voting Machine", Johns Hopkins Information Security Institute Technical Report TR-2003-19, July 23, 2003.

[MVS01] David Moore, Geoffrey M. Voelker, Stefan Savage, "Inferring Internet Denial-of-Service Activity", Usenix Security 2001.

[MPSSSW03] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, Nicholas Weaver, "Inside the Slammer Worm", IEEE Security & Privacy 2003.

[Pfleeger03] Charles P. Pfleeger and Shari Lawrence Pfleeger, *Security in Computing third edition,* Prentice Hall, 2003.

[SPW02] Stuart Staniford, Vern Paxson, Nicholas Weaver, "How to Own the Internet in Your Spare Time", Usenix Security 2002.

Appendix A

The main body of the report covers some of the most serious security risks in great detail, but we did not attempt to give an exhaustive list. This appendix is intended to supplement that list. In this appendix, we briefly discuss several other security issues that also pose serious risks for SERVE.

Vulnerabilities in Servers Could Breach Election Security

SERVE uses centralized computers (servers) to record and forward votes. If those servers are compromised, every ballot cast through SERVE could be modified or replaced, and the integrity of the entire election would be irreparably damaged. Since the servers are a central single point of failure, it is absolutely vital that they resist attack.

The risk of intrusion into SERVE's centralized computers is, unfortunately, significant. SERVE has deployed a careful and well-designed firewalling architecture designed to prevent many kinds of direct attacks; however, there remain possible vulnerabilities in the software exposed to the outside world that could enable attackers anywhere on the Internet to penetrate SERVE's defenses and gain control of the servers. Some examples of the kinds of vulnerabilities are buffer overruns, format string vulnerabilities, directory traversal bugs, race conditions, cross-site scripting bugs, SQL injection bugs, cryptographic failures, and session authentication weaknesses. It is important to understand that there are no comprehensive methods for avoiding, or testing for, these vulnerabilities and bugs. As a result, even the most widely used, exhaustively tested software in the world usually has many such vulnerabilities. SERVE has deployed several mitigation strategies that we will not enumerate here, but these strategies are not sufficient, in our view, to reduce the risk to acceptable levels.

High-confidence software can be found in safety-critical systems such as nuclear reactor control subsystems, fly-by-wire passenger airplanes, the FAA's air-traffic control system, the Space Shuttle, and certain military applications. Designers of safety-critical systems typically avoid the use of commercial software, because it is widely accepted that standard commercial programming practices pose an unacceptable risk for such applications. Designers of safety-critical software employ known techniques for building highly reliable software. These elaborate and costly techniques have not been used in the development of SERVE. Therefore, we must recognize that the standard commercial practices used in SERVE, and in the commercial-off-the-shelf (COTS) software it builds upon, come with an unavoidable risk of failure.

Existing Processes are Inadequate for Certifying Voting Software

Election law in most states requires that all voting systems—whether electronic or not—be qualified by an authorized federally licensed laboratory known as an Independent Testing Authority (ITA), and then submitted to the state for certification. The ostensible purpose of these procedures it to make sure that the voting system meets the voluntary federal voting system standards promulgated by the FEC (and in the future, by NIST), and that they conform to the state's election laws. It is tempting to place a lot of faith in certification procedures as a means for preventing security failures. We believe such faith is unwarranted. We argue that even a lengthy, conscientious testing and examination program by the most qualified people cannot give us the necessary security guarantees. In fact, in general, no process can, since in most cases the problem of establishing that a program meets any particular security requirement is known to be fundamentally unsolvable [Pfleeger03].

With electronic and Internet voting systems the most important part of the ITA qualification process is review and testing of the software. We argue, however, that this review and testing does not, and cannot, guarantee that the software in voting systems actually does the job it is supposed to do. Ideally one would like the testing to be designed to verify that the software is at least *correct*, i.e. that it captures and counts votes properly under all normal conditions, in any possible election, and with any legal voter behavior. It should also verify that the software is *robust* and *reliable*, i.e. that it works reasonably even in the presence of various bug and failure scenarios, including aberrant behavior by users (such as clerks and voters, etc.); and that the software is *secure*, i.e. that it contains no internal malicious logic (Trojan horse code) and is not vulnerable to any of a vast range of potential external attack scenarios. Unfortunately, however, neither

the ITAs nor anyone else could perform review and testing even remotely comprehensive enough to establish any of these properties because the time and labor involved would be truly astronomical. There are fundamental limits to what testing can accomplish; it is a truism of the software world that *while testing can be used to verify that bugs and security vulnerabilities are present, it can never prove that they are absent.*

What the ITAs actually do to qualify voting system software is considerably less than the ideal. They run mock elections with test votes to verify that the software appears to work as required under normal conditions. This amount of testing is probably sufficient to detect simple program errors and obvious defects, although those will likely already have been found by the developers. (We do not really know because, as we note later, results of this testing in SERVE are secret.) But generally speaking such a testing process cannot be expected to do more than filter out the most obvious bugs, leaving intact more subtle bugs that get triggered less frequently. And it is safe to say that the ITAs do very little testing at all to look for software reliability, security, or malicious code problems, since testing is generally ineffective for those purposes.

Besides testing, the ITAs also examine the source code of the software. Most of the examination is in the form of automated scripts that run over the source and look for simple properties required by the FEC standards, e.g. that there are sufficient comments in the code, that "modules" in the programs are not too long, and that each has a single entrance and exit. These are syntactic and stylistic requirements that are only crude indicators of good engineering practices; they indicate nothing at all about the code's correctness, reliability, or security.

We have been told that the ITA engineers also examine the source code by eye, searching for suspicious-looking code fragments that might indicate the presence of a bug, a security vulnerability, or malicious logic. If so, it may give ITA engineers a better idea of the software engineering skill that went into the program development. But it is unlikely that someone not on the development team will spot many subtle bugs in a large codebase, because he or she can never take the time to fully understand its overall code structure in the necessary detail. And, contrary to many people's intuition, it is unlikely in the extreme that anyone, whether on the development team or not, would detect malicious logic that was deliberately disguised by a clever programmer, no matter how much effort was put into the search.[11] It is much easier to hide a needle in a haystack than to find it.

However, even if the ITAs were to do an excellent job of examining the software used in election systems, we would still have major concerns. Regrettably, the ITAs operate under the same cloak of secrecy as voting machine vendors. Both the tests conducted by the ITAs and the results of those tests are secret. Still, there are some things that we know about how ITAs have evaluated computerized touch screen machines (DREs), and there are other things that we can deduce from some glaring failures in the DREs that were not detected by the ITAs. It is reasonable to assume that the shortcomings of ITAs with respect to DREs will carry over to their certification of Internet voting.

For example, there are fundamental differences between modern software systems and previous voting systems. In particular, today's software-based systems are orders of magnitude more complex than mechanical and paper voting machines, and this complexity poses tremendous challenges for verification. In a mechanical voting machine, there are only so many moving parts and only so many ways that the system can misbehave. In contrast, modern software systems are exceptionally complex, with the electronic equivalent of hundreds of thousands or millions of moving parts; such machines can fail in convoluted and unpredictable ways. The very fact that Microsoft and other software vendors are forced to issue frequent software patches (see below) demonstrates that the complexity of modern software systems makes it impossible to detect all software errors, let alone malicious code, with testing and code reviews. Because current methods are good at exposing only simple failures, these methods are not a reliable way of certifying software.

Even if current practices were effective at checking that the voting system behaves as it should during normal operation, security is by definition about the behavior of the system when it is under attack by a malicious entity, a (hopefully) abnormal situation. Because it is hard to anticipate how attackers might

---

[11] For an example of software problems that were undetected by ITA testing, see [Kohno03].

behave, it is hard to test for security flaws. Existing practices almost certainly can be improved, but there are limitations on what can be achieved given our current state of knowledge.

It is even harder to detect intentional security flaws that have been purposefully planted by insiders. After careful consideration, we have concluded that if a malicious insider with access to SERVE software wanted to insert a back door in SERVE's software systems, it is likely that the attacker could camouflage the attack well enough to avoid detection by the ITA's during the certification process, especially in the absence of a meaningful code review.

As if these problems with the certification process were not enough, there is a giant loophole in the FEC requirements: commercial-off-the-shelf software does not have to be tested at all by the ITAs. It is simply assumed to be bug-free, Trojan free, and invulnerable to external attack.

And finally, we note that for systems where security is critical, testing should be done in a truly hostile environment. As Anderson and Kuhn point out in [AK96], "although it is necessary to design commercial security systems with much more care, it is not sufficient. They must also be subjected to hostile testing." The SERVE system has undergone ITA testing but has not existed in a truly hostile environment. There are well known examples of code that underwent extreme testing and then failed when deployed in the field. One example that illustrates this is that the first attempt to launch a space shuttle failed due to a software synchronization error [Garman81]. Very few software systems are tested as rigorously as that which runs on the space shuttle, and yet a serious bug was not found until after the failure in the field.

Given these difficulties, it is no surprise that recent experiences with certification of software voting systems have not been encouraging. Regrettably, there are no good solutions in sight. The computer security community has been struggling with these problems for decades, and they are unlikely to be resolved any time soon.

Commercial-Off-The-Shelf Software Poses a Major Risk to Election Security

SERVE relies heavily on commercial-off-the-shelf (COTS) software. Voters will vote from computers running a Microsoft operating system, and SERVE's infrastructure is built on top of commodity operating systems and applications. We are concerned that heavy reliance on COTS software introduces significant risks.

One of the fundamental problems with using COTS software in a voting system is that it is exempt under FEC guidelines from evaluation by the ITA's during federal qualification. It does not have to conform to FEC coding standards and its source code need not be inspected at all. It does not have to be tested at all except in the context of the whole voting system it is part of. Most of the software comprising SERVE—millions of lines, including the cryptographic core (i.e. the software performing the cryptographic operations) —is COTS, and thus exempt from close scrutiny.

Some advocates of the COTS exemption argue that COTS software is everywhere—that it is the most widely used software in the world, and thus the most thoroughly tested and reliable. We believe that this argument is fundamentally mistaken. While we agree that COTS software is widely used, we disagree that widespread usage is reason for confidence in its security. Indeed, broad usage experience has only underscored the fact that the most widely used COTS operating systems and applications are riddled with bugs and plagued by security defects. New security vulnerabilities in COTS software are discovered every day. For instance, over 4,000 new COTS vulnerabilities were reported in 2002 alone. Such statistics make it likely that the COTS software that SERVE relies upon will have many unknown vulnerabilities, some of which could compromise election security if discovered and exploited.

This raises the question: "What do we do if a new security vulnerability is discovered in COTS software upon which SERVE relies?" Currently, when a security hole is discovered, the software vendor issues a security patch, which happens on a fairly frequent basis. Since security patches modify the voting system, SERVE is required to resubmit the system to the ITA and to the states for certification. This procedure normally takes about two weeks from patch issuance to installation.

Today, it is routine to see hackers exploiting new security vulnerabilities only days after they are initially

announced or after the patch is first made available. Thus, on the one hand a two-week delay may be too long to prevent exploitation of vulnerabilities. On the other hand, two weeks may be insufficient to install, *adequately* test, and certify a new patch. Therefore, we believe that the heavy use of COTS software in SERVE, its known security risks, and its exemption from examination during qualification represent significant risks for SERVE.

Appendix B

The ICANN experience

Internet voting has been occurring for several years, most typically with private elections, such as stock holder elections or the election of officers of private organizations.

Internet elections might possibly be acceptable if the stakes are not high. For example, if the issues on which stockholders are voting are not controversial, then there is little incentive to subvert a stockholder election. By contrast, there is a lot of incentive to subvert national presidential and congressional elections.

In 2000 the Internet Corporation for Assigned Names and Numbers (ICANN)[12] held an election over the Internet. Anyone age 16 or older with an Internet address was eligible to vote. The world was divided into five regions, and candidates were nominated from each region by a nominating committee. There was also a Member Nomination process that involved the "endorsement" of a potential candidate via the Internet.

Problems with accessing the official website occurred in every phase of the ICANN election, including voter registration, endorsement, and voting. ICANN significantly underestimated the computing resources that would be required, especially since people tended to wait until near the various deadlines to attempt to access the ICANN website. Also, some of the processes that ICANN had put in place to try to minimize fraud, such as requiring "activation" of membership after a member had registered, were confusing to voters. Many voters were unable to vote because they did not activate their membership by the deadline.

Even though individual passwords were mailed to physical addresses, some people claim to have voted multiple times. There were also numerous reports of people being disenfranchised. Disenfranchisement occurred because the ICANN website was overwhelmed by the demand, because some voters never received their passwords or subsequently lost them, and because some voters did not realize that registration was not sufficient for voting. In spite of difficulties with registration, about 158,000 people registered. While 76,183 of the registered voters activated their membership, the number who managed to vote in the election was only 34,035.[13] These numbers suggest large-scale disenfranchisement of voters at every stage of the process, with fewer than 1/4 of the initially registered voters actually voting. According to the Markle Foundation report, "the technical weaknesses in the registration system made it virtually impossible to assess the integrity of the voters' list, the security of the PINs, and secrecy of vote."

There were also rumors of country or corporate competition to pack the voter list during the later phases of the registration period. Technical problems experienced during the first day of voting, which took place over a ten day period, created, according to the Markle report, "a credibility problem... with voters and interested parties watching the process."

---

[12] One of the authors of this paper, Barbara Simons, was nominated by the endorsement process; she was the runner-up in North America, losing to Karl Auerbach.

[13] "Report on the Global, On-Line, Direct Elections for Five Seats Representing At-Large Members on the Board of Directors, a report by the Markle Foundation, http://www.markle.org/News/Icann2_Report.Pdf

Appendix C

An Alternative to SERVE

Although we are concerned that the SERVE architecture has too many serious vulnerabilities to be used in public elections, we agree nonetheless that there is a need to lower barriers to voting for overseas Americans and the military. Here we offer an alternative architecture that we believe has most of the benefits that SERVE offers, with far less vulnerability. We are not proposing that this scheme be adopted, but we do suggest that it is a much better starting point for a voting system for that population.

The troubles with SERVE derive from three fundamental design choices: It uses the Internet heavily, with all of the vulnerabilities that implies (e.g., denial of service, spoofing, and man-in-the-middle attacks). It relies on voters using private, unsecured PCs with proprietary, commercial software configured to accept mobile code, with all of the vulnerabilities that implies (e.g., virus attacks, various kinds of privacy violations). And SERVE itself is proprietary software, with all of the vulnerabilities that implies (e.g., security holes, bugs, insider fraud).

We suggest building a system that avoids the dangers of these architectural features. Here is how it might be done:

a) The system would be based on kiosks, to be located at consulates and military bases at major sites around the world. Voters would come to the kiosk to vote, rather than vote from their own PCs.

b) The software on the kiosk would be booted from a clean copy, maintained and configured by trained elections officials so that the software environment on the voting machine is known and controlled.

c) The kiosk is never connected to the Internet; hence, no Internet-related attacks are possible. It receives software and databases via disks (such as DVD Write Once Read Many (WORM)) delivered by certified mail in advance of the election. It does not transmit ballots back to the counties by Internet; rather, it prints them, and they are mailed back to the counties, just like any other absentee ballot.

d) The kiosk has three databases that allow it to perform the voting functions. All three together could fit on one DVD-WORM disk.

   i) An identification database to authenticate all voters who have registered to vote via the SERVE system. This would have at most 6 million records, since that is the size of the eligible population.

   ii) A voter registration database to indicate which ballot image (style) each voter is supposed to receive. This data is compiled from information contributed by any or all of the 3000+ county jurisdictions in the U.S. that wish to participate. There are approximately 200 million registered voters in the U.S., but only at most the 6 million records corresponding to those eligible to vote via SERVE would be needed.

   iii) A ballot image database, containing a PDF or XML representation of each ballot style used in every county in the U.S. that is participating in SERVE. There are on the order of 100,000 such ballot styles. These also would be provided by the LEO in the counties participating in SERVE.

e) The voter at the kiosk identifies himself or herself using military ID or other authentication information provided by SERVE. The kiosk verifies that the voter is registered to vote, and looks up which ballot style to issue. Then the voter indicates his or her choices on a touch screen, and prints out a completed ballot. The voter examines the printed ballot to make sure it correctly indicates his or her choices. (If not, the voter approaches the election official for help in voiding the ballot and re-voting.) This last step provides for a *voter-verified paper ballot*, and assures that no potential bugs or Trojan logic anywhere in the system's code can incorrectly record the voter's intent. Finally, the voter

deposits the completed ballot in an envelope addressed to his home county for mailing back, again like a traditional absentee ballot.

The system we have described is essentially a remote ballot printer located near most Americans throughout the world. Its design would need to be fleshed out with more detail and additional security procedures. These procedures would be charged with preventing multiple voting, managing encryption keys for the databases, and determining what kinds of electronic record of the voting transactions the machines keep and what they do with these records.

We stress that this architecture is not a complete system; it is only a starting point. The details would need to be worked out, and that process could always introduce unforeseen issues. For this reason, we believe it is crucial that any such system undergo hostile review, and that all relevant communities be involved in the design and evaluation of the system. However, with appropriate care, we believe such a system could deliver significant benefits. Such a system would reduce the voting transaction from three trips through the mail down to one, and with proper development we expect it could be made far less vulnerable than SERVE to any kind of remote or programmed attack.

Appendix D

In this appendix, we elaborate on some of the issues that are often misunderstood about software and the difficulty of finding hidden code or flaws in programs. We also discuss why the Internet and the current personal computers are not appropriate platforms for voting applications.

Determining that software is free of bugs and security vulnerabilities is generally impossible

In mathematics there is a long history of profound impossibility results, i.e. theorems stating that certain problems are fundamentally impossible to solve regardless of how much effort or intelligence is applied to them. For example, it is impossible to trisect an angle (divide it into three equal parts) by classical means of ruler and straight edge construction; it is impossible to solve a 5th-degree (x^5) polynomial equation using only addition, subtraction, multiplication, division and extraction of roots; it is impossible to construct a consistent and complete axiomatization (set of rules) for arithmetic.

Computability theory has a many such results, referred to as unsolvable or incomputable problems. They are typically of the form "there exists no computer program that can do X". The first and most famous such result, due to Alan Turing, is known as the Halting Problem: there exists no program H that can determine (in a finite number of steps) whether or not an arbitrary other program P ever halts. It does not matter what programming language is used, or what computer is used, or how fast it is, or how much memory is available, or how long the program is, etc.; there simply is no program H that can solve the Halting Problem.

If a problem is unsolvable in this sense, then humans cannot solve it either. This may seem contrary to the common observation that there are things humans can do easily that no computer program can do, e.g. understand the English language; but these are examples of things no program yet written can do, rather than examples of things no program at all can do. In general, if humans can perform an information processing task, then they do so by some method in the brain. We may not know the method; but if we did, we could write a (complex) program that emulates that same method and performs the task just as well. There does exist a program that understands English, even if no one knows at the moment how to write it, so understanding English is not an unsolvable problem.

Unfortunately many of the important questions one might ask about the behavior of software--its correctness, reliability, or security--turn out to be unsolvable. Here is a partial list of such problems that are relevant to program correctness (absence of bugs), program security, and the privacy required by election software. They are stated informally because the mathematical apparatus to state them precisely is beyond the scope of this report; but each could be suitably formalized and proved mathematically.

It is impossible to determine by any finite means, for an arbitrary program, whether or not it:

- halts (a compressed phrasing of the Halting Problem);
- has a particular type of bug (e.g. array bounds error, fencepost error, dangling pointer);
- is correct (i.e. meets its specifications);
- has hidden functionality (i.e. does additional things not mentioned in its specifications, especially malicious actions);
- preserves privacy (does not store or transmit information that it should not);
- is reliable (i.e. does something reasonable in spite of various kinds of failures, e.g. memory, communication, or software failures);
- is secure (i.e. performs its job in spite of various kinds of attacks).

Reading this list one should get the impression that just about anything you really want to know about the behavior of important software, including election software, is impossible to determine for certain. And indeed that is the case. If there were a surefire way to detect bugs in programs, then commercial software would be bug-free. And if there were a foolproof way to find or avoid security vulnerabilities, then

operating systems companies would use those methods, and would not have to issue security patches on a regular basis.

If all of these important software problems are unsolvable, then how does working, secure software get built at all? This is the subject of software engineering. The general answer is that, with enough careful mathematical and algorithmic analysis ahead of time, with enough methodological care and discipline on the part of programmers, with ferocious drive for simplicity of design, with enough systematic and randomized testing of the software, with formal proofs that key parts of the program have key properties, and with enough openness so that the software can be scrutinized by many experts, then the prevalence of defects such as bugs, or security vulnerabilities, can often (but not always) be made tolerably low. But there is no method for writing totally bug free software, or totally secure software; and if by a miracle such software were to be created, there is no general way of proving it to be bug-free or secure. This is a mathematical fact of life. Anyone who claims that some nontrivial program is bug-free or secure simply because it has been widely used or thoroughly tested is misinformed. No amount of use or testing is sufficient to prove that.

Though there has been substantial research on new methods for software engineering—for instance, proof carrying code and static program analysis, among others, these technologies are currently immature, and they are a long way from being applicable to the complex type of software needed for elections.

Malicious code

Software is generally designed to serve a clear, documented purpose, and people rely on it doing what its specifications and documentation say. But software can be written to do other things in addition to, or instead of, what it is supposed to do, i.e. it may have secret or hidden functionality that is not documented. Sometimes this is legitimate, but sometimes the hidden functionality is malicious, i.e. it does something that the programmer or vendor wants, but the user does not want, and does it without the knowledge of the user. Such malicious code might spy on the user by surreptitiously sending private data to an Internet site; it might throw ads in the face of the user; it might disable security protections on the computer to allow later break-ins by unauthorized people; it might install other unwanted programs; it might do random violence by deleting data or files; or it might do any of a thousand other malicious things.

The terms *Trojan horse*, *virus*, and *worm* all refer to types of malicious code, differing only in the means by which they get transported to the computer and get executed. Most people are aware of email viruses, which is malicious code in the form of an email attachment, but there are many other infection routes, including, for example, malicious scripts (ActiveX controls, JavaScript programs, or Java applets) that can enter your computer as an invisible side-effect of visiting a web page. Malicious code is one of the most serious security threats in any application, because it is so easy to install, and so difficult to detect.

In the context of the SERVE voting system, malicious code is a threat in two separate ways. First, it is possible that an insider (one of the developers who builds SERVE) might insert malicious code into the SERVE software itself, perhaps to spy on votes or to selectively throw some away. (We don't suggest this is at all likely; but the fact is that it cannot be excluded.) The other threat is malicious code infecting voters' computers, in such a way that it spies on the voting process, or prevents voting, or even changes votes without detection.

It is important to understand, as discussed above, that there is no foolproof test for whether or not malicious code is installed. Virus checkers, for example, can detect the presence of viruses that have been seen before, studied by experts, and for which a signature has been extracted; but it cannot reliably detect new viruses. Even experts with access to the source code of a program may not be able to tell if there is malicious code in it, since it is relatively easy to disguise malicious code so that it is extraordinarily difficult to find.

The most powerful defense against malicious logic is for the security of the election not to depend on detecting it at all, but to structure the voting system so that we can guarantee that it works correctly even if malicious logic is present. Voter verified audit trails are among the simplest and strongest features that can provide such guarantees (though there may be others). But in our opinion some form of protection against malicious logic is imperative for any software-based voting system.

Security weaknesses of the Internet

The Internet is revolutionizing communication, commerce, and entertainment, but in ways that were never envisioned by its designers. The basic data transmission, naming, and routing protocols still in use on the Internet today, called collectively TCP/IP, were designed in the late 1970's and early 1980's, a time when all network users formed a single community, when the network engineers all knew each other personally, and there was universal trust among the users. There was little need for, security because the community of users was small and everyone was cooperative. The key goals of the Internet's original protocols were scalability, ease of use, communication performance, and reliability; security was not a priority, and the issues were not well understood at the time anyway.

But today, after a million-fold growth in size, the Internet is a voluntary worldwide federation of networks of great diversity, with no central authority and no common culture or goals. It links hundreds of millions of people and organizations that are mostly strangers to one another, but sometimes competitors or even enemies. In such an environment, security is of profound importance, and many security protocols have been layered on top of the basic ones with the intent to secure certain kinds of applications, e.g. email or financial transactions, against certain specific threats.

Yet, for all of the importance of security today, the Internet has no general security architecture; in fact, it is well known to be full of very general vulnerabilities. It relies on the voluntary cooperation of thousands of corporations around the world to keep its naming and routing infrastructure consistent, and it uses open, forgeable source addresses on every packet it transmits. As a result, it is not possible to guarantee exactly where an outgoing data packet will be routed, or where an incoming packet came from. These limitations are the ultimate reason why spoofing and man-in-the-middle attacks are so easy to perpetrate and difficult to defend against in any Internet application, not just elections. Likewise the routers and servers distributed all over the world that form the Internet's higher-level infrastructure are themselves just computers with their own array of security vulnerabilities, known and unknown; they are often easily tampered with, by either insiders or outsiders.

It is extremely hard, to build a secure, all-electronic Internet-based application, e.g. a banking system, a legal system, or an election system, on top of such a fundamentally weak security foundation. In our opinion, no one should attempt it without voter-verified audit trails and out of band (i.e. non-Internet) channels of communication between the LEO and the voter, or else an as-yet unknown security breakthrough.

Security weaknesses of the PC

The PC platform that is used as a voting machine under SERVE is another dangerously weak element of the security architecture. A PC running Windows software is very easily compromised, and never more so than when it is connected to the Internet.

The PC was originally designed as a personally owned and maintained, single-user system (which is why it was called a "personal" computer). This fact seemed to minimize the need for sophisticated security since the owner/user could always trust himself/herself. The Internet was not originally a security consideration because the PC family of architectures was designed in the early 1980's, long before there was any thought that PCs would connect directly to the Internet. In any case, it was generally assumed that whatever light security might be useful, such as password protection, could be satisfactorily implemented entirely in software.

As a result, PC hardware has evolved to this day with essentially no concern for security at all. Most PC motherboards, for example, contain no tamperproof, cryptographic keys that can be used as the basis for identification, and no source of true random numbers for cryptographic key generation. Furthermore, the key human interface devices for PCs (keyboard, pointer, screen, or speakers) are not designed to perform cryptographic operations. This has the effect of forcing critical data to be manipulated in the clear (unencrypted) on the main part of the PC, where it is vulnerable to malicious software.

The Microsoft software suite for the PC was also originally designed without security in mind. The primary design goals of the Windows and Internet Explorer have been maximal functionality and ease of use, not security. Only recently has Microsoft made security an important priority, but their operating

systems are still well known to be thoroughly riddled with security vulnerabilities. Indeed, for years hardly a week went by when Microsoft did not issue one or more security patches. Indeed, to help manage the problem Microsoft has started batching the patches so that groups of them are released monthly.

Windows is especially vulnerable to malicious software attacks. There are so many forms of software, and so many vectors of transmission, that it is impossible to control them all. Users are constantly encouraged to download and install software, sometimes without knowing it, including OS and browser updates and patches, device drivers, plug-ins for browsers and other applications, scripts associated with web pages and office documents, shareware programs, and of course, full applications. Any of these types of programs can contain malicious logic that might completely undermine the security of the PC, and of any votes cast on it, without the user/voter ever finding out.

These security limitations of PC hardware and software are widely acknowledged. There is an industry-wide consortium known as the Trusted Computing Platform Alliance (http://www.trustedcomputing.org) that has specified hardware additions to the PC architecture that are intended to remedy some of its security deficiencies. So far there are no hardware implementations of the TCPA spec and no operating system support for it. It is possible that within a few years TCPA-equipped PCs will be begin to be sold, and a few years after that they will largely replace all of the current generation of PCs. But there is as yet no public specification of what security features and tools Microsoft will implement using the TCPA hardware, or whether they will suffice to permit safe Internet voting from a PC.

Appendix E

Author Biographies

David Jefferson is a computer scientist at Lawrence Livermore National Laboratory where he does research in scalable parallel computation (supercomputing). He has also worked for many years in the field of election technology and security, serving as chair of the Technical Committee of the California Secretary of State's Task Force on Internet Voting in 1999-2000; as a member of the executive committee of the NSF panel on Internet voting in 2000-2001; and a member of the California Secretary of State's Task Force on Touchscreen Voting in 2003. He has been active nationally as a speaker and advisor on computerized voting system issues, and is a current member and past chair of the board of directors of the California Voter Foundation. Before joining LLNL, he was a computer scientist at DEC/Compaq Labs in Palo Alto, and before that had been Associate Prof. of Computer Science at UCLA, where he was best known as the co-inventor of the Time Warp method of parallel discrete event simulation.

Aviel D. Rubin is an Associate Professor of Computer Science and Technical Director of the Information Security Institute at Johns Hopkins University. Prior to joining Johns Hopkins Rubin was a research scientist at AT&T Labs. Rubin is author of several books including Firewalls and Internet Security, second edition (with Bill Cheswick and Steve Bellovin, Addison Wesley, 2003), White-Hat Security Arsenal (Addison Wesley, 2001), and Web Security Sourcebook (with Dan Geer and Marcus Ranum, John Wiley & Sons, 1997). He is Associate Editor of ACM Transactions on Internet Technology, Associate Editor of IEEE Security & Privacy, and an Advisory Board member of Springer's Information Security and Cryptography Book Series. Rubin serves on the board of directors of the USENIX Association and on the DARPA Information Science and Technology Study Group.

Barbara Simons is a technology policy consultant. She earned her Ph.D. from U.C. Berkeley, and was a computer science researcher at IBM Research, where she worked on compiler optimization, algorithm analysis, and scheduling theory. A former President of the Association for Computing Machinery (ACM), Simons co-chairs the ACM's US Public Policy Committee (USACM). She served on the NSF panel on Internet Voting, the President's Export Council's Subcommittee on Encryption, and the President's Council on the Year 2000 Conversion. She is on several Boards of Directors, including the U.C. Berkeley Engineering Fund and the Electronic Privacy Information Center, as well as the Advisory Board of the Oxford Internet Institute and the Public Interest Registry's .ORG Advisory Council. She has testified before both the U.S. and the California legislatures. She is a Fellow of ACM and the American Association for the Advancement of Science. She received the Alumnus of the Year Award from the Berkeley Computer Science Department, the Norbert Wiener Award from CPSR, the Outstanding Contribution Award from ACM, and the Pioneer Award from EFF.

David Wagner is an Assistant Professor in the Computer Science Division at the University of California at Berkeley. He has extensive experience in computer security and cryptography, and over 50 technical publications. He and his Berkeley colleagues are known for discovering a wide variety of security vulnerabilities in various cell phone standards, 802.11 wireless networks, and other widely deployed systems. In addition, he was a co-designer of one of the Advanced Encryption Standard finalists, and he remains active in the areas of systems security, cryptography, and privacy. Wagner is an Alfred P. Sloan Research Fellow and a CRA Digital Government Fellow.