OSET INSTITUTE

Position Paper

Version 4.3

The Appropriate Use of Open Source Technology in Government Mission Critical Computing

Prepared For:
U.S. Election Administration Leadership

Prepared By:
E. John Sebes
Co-Founder & Chief Technology Officer

Dr. Clifford Wulfman, MSCS, Ph.D
Sr. Member of Technical Staff

February 2018

Preface

The OSET Institute was founded on the idea that election technology infrastructure is a critical component of government IT; so critical in fact, that it should arguably be a public asset on which the commercial industry (*or election organizations themselves if properly resourced*) can build and deliver finished open standards, open data, and, accordingly, open-source based systems. Historically inherent in our name, OSET ("*Oh-Set*") are a pair of words, "open" and "source." We have always maintained that open source is neither necessary nor sufficient for higher integrity, lower cost, easier to use election administration systems. However, publicly available technology (*i.e., open source*) is an important ingredient to ensuring transparency and trust in the technology. In the decade since the Institute's founding, "open source" as a phrase used in conjunction with voting systems has grown to be a provocative and even in some limited situations, controversial topic. *It should not be.*

Open source does *not* mean "free source." Open source primarily addresses transparency, as the phrase elements imply. Open source is both a process of development and a means of distribution. Applying technology to elections, the objective is elections whose processes are Verifiable, Accurate, Secure, and Transparent (*a principle called the "VAST mandate.*") If voting technology can be developed transparently, and made available in an unencumbered manner with incentive to continually innovate while taking care to rapidly identify errors, flaws, and vulnerabilities, then there is a higher probability for public elections achieving the VAST mandate.

Therefore, we believe it is essential to understand what exactly open source technology is and is not; can and cannot do; and the appropriate uses of open source methods and means in mission-critical government computing, particularly election administration, which has become a matter of national security. In this paper, Dr. Clifford Wulfman, a senior member of technical staff at the OSET Institute, and John Sebes, co-founder and CTO, explain just that. We hope it is helpful to your continuing pursuit of innovation in this vital aspect of democracy administration.

Executive Summary

The term “open source” can be used and abused in ways that confound logical, fact-based technology policy discussions about government procurement of critical software. Such confounded discussions are a real problem, because Open Source Software (“OSS”) is a nearly inevitable part of the foundation of any government-critical software, as well as a useful method for filling technology gaps with non-proprietary software for which there is no profitable commercial vehicle for its creation and support.

Among the common misperceptions about open-source software are the following.

Common Misunderstandings

1. **OSS is communally developed without controls.**
   Reality: While OSS is frequently developed by highly distributed networks of contributors, integration of new code and revisions is carefully controlled by project managers using commonplace techniques employed by software-development organizations of all stripes.
2. **OSS can be modified by anyone.**
   Reality: While many OSS projects use public software repositories that allow code to be duplicated and reused, the primary repository remains entirely under the control of its owners or custodians.
3. **OSS depends on free donations from unvetted volunteers.**
   Reality: Repository custodians have complete control over what, if any, contributions are incorporated into the code base.
4. **OSS is free.**
   Reality: In many government-computing situations, *free* simply means that the contractor will not charge license fees for non-proprietary software. The term does not necessarily have ideological connotations.
5. **OSS is haphazardly dependent on externally developed libraries and components.**
   Reality: Code adoption and reuse can take place on a spectrum from disciplined minimality to sheer expediency for speed of delivery, depending on the choices of an effort’s leaders, and the team’s effectiveness in executing on those choices.
6. **OSS is more (or less) secure and reliable than proprietary software.**
   Reality: The so-called “security-by-obscurity” model of software development was debunked long ago, and cyber-security experience has shown that source-code disclosure does not advantage adversaries. At the same time, while making source code available for public inspection increases the probability that security flaws will be discovered and repaired, it does not guarantee it, and does not obviate the need for C & A. Experience has shown that the technology community, both commercial and noncommercial, is able to provide resources for custodianship when it is evident that a body of software is truly critical.

However, just as non-proprietary software methods (*e.g., development, distribution, licensing, tech transfer, commercial integration and support*) are not an intrinsic barrier to effective creation of government-critical software, neither are these methods a panacea or a guarantee of success. A non-proprietary software organization still must set appropriate goals for the adopters' needs; successfully execute on those goals during initial development, and thereafter; support successful C & A activities; and provide organizational continuity for ongoing custodianship of the software.

These requirements for success may be serious challenges in a government technology market that hasn't been able to support profitable commercial activity to develop needed solutions.

1. Overview

Open-source technology is an essential part of most modern computing, particularly the open-source software ("OSS") that underlies most mobile apps and web services used worldwide, and the public network infrastructure beneath that. But the term "open source" is widely and variously used to refer to many concepts that seem inherently inconsistent with technology that is foundational and essential. Only two aspects of the open source approach — non-proprietary licensing and distribution — are relevant to government-critical computing.

The list of other, irrelevant connotations of open source is long: OSS is communally developed without controls; source code can be modified by anyone; OSS development depends on gifts or volunteer efforts; OSS is free; OSS is haphazardly developed with many unknown dependencies on a large body of other OSS; OSS source-code publication creates higher software quality from "many eyes;" OSS is has higher quality and reliability than other software; OSS has lower quality and reliability than other software; OSS is more secure than other software; OSS is less secure than other software; OSS necessarily has more (*or less*) of any number of desirable qualities.

While these conceptions have become common because at least some of them often apply to software identified as being "open source," there is no *necessary* relation: no particular piece of open-source software need possess all (*or even any*) of these features. Furthermore, most of the uses of the term "open source" are actually irrelevant to government-critical computing. But because this irrelevance is not widely understood, misconceptions of "open source" continue to foster confusion in technology policy discussions.

2. Decoupling Critical Computing and the Term "Open Source"

In the U.S., almost all government computing relies on open-source software to a significant degree. Perhaps the largest portion of U.S. government computing — homeland security, defense, and intelligence computing — is also one of the largest areas of computing anywhere to explicitly and strategically adopt OSS. This area of "government-critical computing" is an expedient adopter of non-proprietary software that meets a specific need without the encumbrances of commercial software acquisition. More than a lead adopter, U.S. government-critical computing is one of the largest drivers of the creation of non-proprietary software: that is, software developed under government contracts, with taxpayer funds, resulting in publicly owned technology. In many situations, technology transfer of such publicly owned technology is accomplished by methods often associated with open-source software.

2.1. OSS is Foundational to Critical Computing

As a result, to understand the foundation and evolution of government-critical computing means understanding not just OSS, but also the variations in its ecosystem for development, technology transfer, and use. However, many people involved in technology and policy have a less-than-complete view of OSS, and that varying ecosystem, because of the wide variety of ways in which OSS is described without regard for these variations. As a result, those confusions and contradictions can cloud policy discussions about government computing and impede the adoption of technology that can create real public benefit.

2.2. "Open Source" has a Narrow Meaning in Critical Government Computing

Many of the ideas about *open source* have their roots in Eric S. Raymond's *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary* (O'Reilly, 2009). Raymond's book has enduring influence, and it has led to a common perception of an "open source project" as a sort of commune, or communal effort to pool resources to make technology in a way that wouldn't happen in a purely for-profit, strictly proprietary software effort. Those ideas influenced a lot of work that is called "open source" by the people who do the work, or by observers of the work.

But in fact, the term "open source" means very little when it comes to critical government computing and, particularly, election technology, where it pertains to only two things: licensing practices and distribution practices.

Sometimes people create software that they want others to use without constraint, and so they distribute it publicly in a raw form — source code — that anyone can use to build and use the software. That's distribution. However, in most of these cases, software creators don't want others to take the software, claim to own it, and demand fees for its use. Likewise, potential adopters — especially those in government computing — have a procurement process where software right-to-use needs to be very clear. To meet both needs, the software's creators protect the software with a usage license that's designed to prevent appropriation while allowing its use in a variety of commercial settings, including those that involve government procurement of bundles of software, integration, support, and services.

Fortunately for software creators in such situations, software-licensing-law experts have done years of work designing a variety of licenses for just these situations. These are called "open-source licenses" for software. Use of these licenses enables "open-source distribution" with license protection.

These two practices – open-source licensing and open-source distribution – are better described by the term *non-proprietary*. In government-critical computing there are multiple reasons why it is useful to have non-proprietary software, and careful licensing and distribution are important to the use of non-proprietary technology. But choosing a licensing and distribution model is completely separate from choosing a software development model, or a project management model, or a corporate business model, or a non-profit organization operational model.

3. Irrelevant Connotations of the term "Open Source" to Critical Government Computing

3.1. The Commune

A particularly vexing "open source" misperception is that all open-source projects are communal. It is certainly true that many "open source projects" are communal efforts, with a highly distributed network of contributors, working over time to maintain and extend a large complex body of software like an operating system, or DBMS, or web-application software stack. Many of these have a governance structure called "commit rights" that vary from a stable inner circle of distributed contributors, to an individual in sole control, to a core team of people who are permanent full-time employees of a company that manages the project and the delivery of its results.

Critical software needs a disciplined software development process, but in terms of discipline and control, it doesn't matter whether a team's source code control uses a publicly visible control platform like GitHub.com or a completely closed private system. The point is that the controls of the source-code control system are used by the managers of a project to carefully separate the main line of software from any number of separate sandboxes used by developers. Developers and managers need to use these control features in order to have a disciplined release-engineering process — which is essential to making software releases that clearly defined and documented for adopters.

But there is nothing mysterious about these controls. They are commonplace techniques used by software development organizations of many types, from Mil-Spec government contractors to actual software communes, and all stripes in between. Some "open source" projects may be communal in nature and also choose to operate with a loose control structure and not much release engineering. However, for a critical-software effort, those are inappropriate choices, regardless of whether some might describe the effort as "open source."

3.2. The Fork

Another serious misperception arises from public software repositories, and the ability of anyone to make a copy of a software repository in order to do their own work on their own "fork." Forking is an obvious consequence of the choice to use a software repository that is public, as part of a choice about licensing and distribution. If anyone can see it, anyone can copy it. The license may limit what the copier can do with their copy in commercial activity, but anyone can copy, and then do what they like with their copy.

Does that mean that anyone can modify any piece of software that's described as "open source"? *Absolutely not*. The primary repository remains in place, regardless of who copies anything from it, and its contents remain under the control of its owners or custodians. There is no necessary connection between those custodians' efforts and people who have made copies for their own tinkering.

This control concept is essential for government-critical software and its release-engineering process. At the end of a release cycle, the software is packaged into a whole that needs to exist standalone, for repeatable testing, often in the context of a government-regulated accreditation

and certification model. It is a particular release of the critical software, tested and approved, that a government adopter organization can choose to use — not some ad hoc download from branch of some repository.

Release engineering and certification and accreditation (C & A) practices are essential parts of the process of deploying and maintaining government-critical software. C & A applies regardless of how the software in question is licensed or distributed, regardless of whether it is secret and proprietary, regardless of whether the source code is available for public use in any manner.

3.3. The Gift

A related perception is about the "open source" practice of "giving back" elaborations of a particular software source-code base. Anyone can make a copy, work on it to create some new feature or extension, and then make that innovation available to the custodians of the source code base that was copied. Indeed, some OSS licenses place constraints on certain types of usage and on the ability of a third party (*neither custodian nor adopter*) to withhold the source code for extensions.

However, nothing about these gifts compels a custodian to accept them. For critical systems, it may seem alarming that the source code is being tinkered with by an unknown number of parties, some of whom make available the results of their tinkering. However, the real issue isn't how many offers of gifts are made to a custodian. The real issue is the extent to which a custodian may choose to accept such a gift, incorporate it in part of the source code base, and (*possibly but not necessarily*) include in a future software release some code that was derived from a gift.

For critical software, this might be less commonplace than in a communal software-development effort. However, for *government-critical* software, C & A activities are the more important parts of the process. C & A requires detailed build tools and documentation, full logs of source-code control, and the ability for accreditors to see the provenance of every change made in the source-code control system. Uncontrolled or unwise incorporation of gifts might occur in a poorly managed development effort, but they can't be hidden during C & A.

3.4. Free Software

Perhaps the most confusing aspect of open-source software has to do with the notion of freedom (*vs. being free from or of encumbrances*).

In some contexts, "free software" means software that users can obtain and use without charge. A great deal of such free software is not open source, but is in fact proprietary software available under commercial licenses that happens not to require payments of fees for the right to use it. In other contexts, "free software" means software that users can run, study, change, and redistribute without restrictions. This concept is central to the free software movement (FSM) or free/open source software movement (FOSSM) or free/libre open source software (FLOSS), an ideological position from which certain kinds of software, or even all software, is so important to public benefit that is unethical for anyone to claim exclusive commercial proprietary rights to software, and to change fees for a right to use software.

Needless to say, there is no logical or causal connection between (**a**) a development team choosing to license and distribute non-proprietary software with an OSS license, and (**b**) the people in the team holding any particular ideological or political beliefs about "free software."

Free adoption is not a good fit for government computing. A typical government organization might possibly include an individual employee who downloads and uses some open-source software, for example Google's popular web browser, Chrome. There may even be some borderline cases where a government organization's IT department would make a similar adoption, employing, for example, popular open-source tools for network security. But for any actual government IT project of any size, the project has a budget to spend on technology acquisition and services like system integration and support. Where spending public funds is involved, so also is procurement. A procurement might well include some non-proprietary software acquired with an OSS license; but that would be one part of a larger set of deliverable items and services delivered by a commercial contractor to the government.

In this latter situation, "free" simply means that while the contractor will pass through its expenses for acquiring proprietary software (*including license fees*), it will not be charging license fees for non-proprietary software. The contractor will, however, be charging for its services in integrating, deploying, and supporting the total system that they were contracted to build. One of the important values of "open source" in this context is that for non-proprietary software (*but not proprietary software acquired by the contractor*), the contractor is able to adapt the non-proprietary software to meet specific government requirements that are not met "out of the box."

In other words, just as not all "free software" is "open source", in many cases "open source" software is not necessarily free to a government organization. In the latter case, there are no software license fees, but part of a procured project's expenses may be specifically for the contractor's services in connection with non-proprietary software.

3.5. Undisciplined Complexity

A more subtle myth about "open source" is, like the commune, an aspect of the culture of a particular software effort's development practices. The myth arose because in fact many open-source projects operate more from expediency than discipline and create software with considerable complexity derived from its "building blocks."

The building blocks in question are just more software packages. Almost any modern software development effort will rely on code reuse of pre-existing software that already meets part of the needs of the software being developed. Such reuse is not mere expediency. Modularity, data hiding, and code re-use are also important parts of critical software development, where it is preferable to use existing software packages that have a proven track record in usage, rather than re-implementing functionality from scratch and risking the creation of instabilities (bugs) that have already been ironed out from existing stable software.

However, reuse of building blocks can take place on a spectrum from disciplined minimality to sheer expediency for speed of delivery. Both proprietary and nonproprietary software development efforts can be anywhere on this spectrum, depending on the choices of an effort's leaders, and the team's effectiveness in executing on those choices.

As with other choices discussed above, there is no logical connection between these choices, and choices to develop non-proprietary software. It's certainly true that many "open source projects" operate for speed and expediency, without any claim to critical software or to commercial grade quality and reliability. Indeed, looking at public repositories of public source code, many reasonable observers would classify a majority of public repositories to be software somewhere along the spectrum of hobbyist activities — including many repositories that are not principally about software, but are labeled as "open source" in some sense meaningful to the repository's owner. The reverse is also true — public repositories include many large projects with a great deal of code that is "open source" but high quality, supported by many professionals and their employers.

Any effort to develop critical software should make more critical choices about which building blocks to use, and how much effort to invest in assessing their internals and their dependencies. But the success of execution on those choices is hardly guaranteed — it must be proven in a process of certification and accreditation (C & A).

3.5.1. Variations in Complexity, and a Note for Election Technology Specifically

Another factor of software complexity is basic requirements. Not every non-proprietary system needs to be inherently complex. Not every simplistic proprietary system will be devoid of expedient but unnecessary complexity.

For example, a great many proprietary consumer products sold as part of the "Internet of Things" are inexpensive products built on open source-operating systems and application software stacks, with limited attention paid to components and how to use them. Without any significant care in how the building blocks are used by their vendors, consumers end up with simple devices like baby monitors (*main ingredients: audio microphone and Wi-Fi hardware; operating system; and a sliver of custom code*) with all of a full-service server operating system in default configurations enabling adversarial usage ranging from surveillance to spam botnets.

At the other end of the spectrum are military and/or communications embedded systems that are no less special-purpose systems with limited requirements, but have been built as critical components, independently tested, certified, and accredited before fielded use. As components of government-critical computing, these systems can be built to a high standard of care by a large experienced government contractor, while also being comprised of non-proprietary software. In many cases, the non-proprietary nature of the software arises directly from the acquisition of a "custom system" as a work build for hire by the government, where the builder retains no proprietary rights in usage or distribution of the software, or in its intellectual property.

Within the field of election technology, individual components of voting systems, such as ballot-casting or counting devices, can be defined as relatively simple but critical systems, doing limited functions much like a military embedded system with an optical sensor, and the ability to capture images, interpret them, and record both raw and interpreted data. Where a critical system has relatively simple requirements, it can be built with low complexity and judicious choices of building blocks that have appropriate levels of complexity, demonstrated reliability, other characteristics.

3.5.2. Open Source Depends on Open Source

One particular subspecies of the complexity myth is the mistaken notion that open-source software depends on open-source building blocks, and that open-source building blocks are necessarily (*as a result of being labeled "open source"*) of low quality, or have minimal support, or have sprawling dependencies on other open source software.

Again, that may be true by choice or default in some efforts, but it is hardly a foregone conclusion, particularly in a critical software effort that must make judicious choices and be assessed on those choices by accreditors. Furthermore, there is no requirement for a body of non-proprietary software to depend solely on "free" building blocks (though some open-software licenses impose this restriction). There is no necessary correlation between robust support of a building block, and the software license terms under which such a building block is available for use.

3.6. Many Eyes and Reliability

Software security authorities are often divided into two camps: those who believe that publishing source code makes software more vulnerable to security exploits and so should be discouraged ("*security by obscurity*"); and those who believe that making source code available for public inspection makes it more likely that security flaws will be discovered and repaired ("*with a thousand eyes all bugs are shallow*").

The so-called security-by-obscurity concept was thoroughly debunked by computer scientists decades ago (*see "Kerckoff's Law of cryptography"*), and cyber-security experience has shown that source-code disclosure does not advantage adversaries, who typically rely on automated probing techniques to discover vulnerabilities in running code. Probes and dynamic analysis are effective across scale and complexity, compared to source code analysis with efforts that increase super-linearly with size and complexity of source code. Professional adversaries find it much most cost effective to rely on automation and tools than to grind over large quantities of course code.

Confidence in critical software is also hard to muster if the closed proprietary software's owners both boast of industrial grade security or military grade cryptography, while also admitting fear that review of their software will enable adversaries to undo all the boasted security. For critical software, vendor assurance is not the basis of confidence; independent test and review in a C & A process provides credible assurance in critical software's adequacy for its purpose.

Lastly, claimed security-by-obscurity also creates a moral hazard for for-profit vendors of proprietary software. Profitability caps technical efforts; limited technical effort can result is lower software quality; the belief that limited quality will not be evident (*because source code is private*) may lead to poor decisions about efforts for quality vs. other goals. Where there is little public harm from security incident, such decisions might also have small connection to public interest. But if software is critical, and failures can create public harm, then hidden deficits in software quality rise in importance above profitability.

The thousand-eyes concept relies on the probability that flaws in source code are more likely to be discovered if more people are looking for them — more likely, perhaps, but not guaranteed,

so for critical software it is unwise to assume that open-source software is less buggy than proprietary software. The story of OpenSSL is instructive.

OpenSSL is the foundation for cryptographic communication security in an enormous portion of modern computing. Its source code is public, available for analysis as well as use, and many adopters have examined the code, including technologists at some of the world's largest and most successful technology companies. Nevertheless, a bug introduced in 2012 made machines using OpenSSL vulnerable to serious security exploitations, and this bug (dubbed "Heartbleed") remained latent though in plain sight until 2014.

Prior to Heartbleed's discovery, the organization supporting OpenSSL had few resources devoted to software maintenance. When the flaw was revealed, the technology community – largely for-profit companies – stepped up to provide resources for custodianship.

Heartbleed showed how critical software requires not just availability, but also effective custodianship for maintenance over time. It also showed that the technology community is able to provide resources for custodianship when it is evident that a body of software is truly critical.

The characteristics of software – security, quality, reliability, hardware fault-tolerance, resilience in response to communication interruptions, and many others – are outcomes of software architecture, design, and implementation efforts. The outcome in any specific case depends on human efforts, not on *post facto* issues like whether the source code is published, or how rights to use the software are sold or controlled, or whether observers use the term "open source" to describe the software. Open software makes it easier for more interested parties to get some visibility on such outcomes, but visibility alone neither assures nor threatens the security of critical functionality.

3.7. Custodianship

Heartbleed showed that, in at least some cases, "open source" foundational and/or critical software will be curated and sustained by organizations with sufficient resources, over many years, including significant support and involvement from commercial organizations that non-exclusively profit from the customization and delivery of the software. There is no necessary or logical connection between profit motive and sustainability.

This lack of connection is amply demonstrated in government-critical computing. In U.S. elections, the 2016 election cycle saw approximately a quarter of jurisdictions using products that are no longer manufactured, including products from vendors years out of business, or acquired, or liquidated with service contracts transferred to other vendors as part of antitrust activity. Other voting-system components still in use, including the back-office hub hardware/software for voting-machine management, depend on versions of Microsoft operating-system products that are no longer supported and in which new vulnerabilities continue to be discovered but not remediated.

Not only are these scenarios evidence; they are also well-represented in public policy for government-critical computing. Just as government procurement sometimes requires open-sourcing of non-proprietary software (described in more detail below), similar procurement processes require proprietary software delivery to include source-code escrow. Both methods

protect government acquirers from situations in which critical software in continuing use turns out to be orphaned by changes in the original delivering organization.

4. Relevant Connotations: Licensing and Distribution Practices

Having set aside many inappropriate uses of the term "open source" – at least as regards government-critical computing – let's circle back to two uses of the term that are relevant and ask why they are relevant.

As described in Section 1, the term "open source" is a convenient but confusing shorthand for now-common approaches to distribution and licensing that make software available for broad use while protecting it from abuse. A better term for these approaches is *non-proprietary*, and there are many reasons for the existence of non-proprietary software, ranging from hobbyist activity, to startups, to well-supported projects to extend and maintain foundational software like Linux, MySQL, Apache, OpenSSL and many others.

But what is the role of non-proprietary software in government-critical computing? There are three main use cases: foundational use, outward technology transfer, and inward technology transfer. Foundational use is simply the adoption of well-supported, well-maintained open-source software like Linux and OpenSSL which forms the base for government-critical computing. The other two use cases have to do with software-development efforts to meet government-specific needs that have not been met by the commercial market.

4.1. Outward Technology Transfer

Not infrequently, projects performed for a government organization by a commercial organization result in an integrated system that includes the software custom-developed by the project, together with pre-existing software – often a mix of nonproprietary and proprietary software that might include software owned by the contractor. In these cases, it is a common practice in parts of the Federal government, such as DoD and DHS, to require that such software be published and distributed under an "open source software license" rather than, for example, granting exclusive IP rights to the contractor so that they can sell that software and related services to other government entities, and so that they can remain the sole source of future extensions and support on the custom software. Although such future support might indeed be delivered by the same contractor, that future work can be competitively bid among any competitors who might wish to extend or support the publicly-owned non-proprietary software.

4.2. Inward Technology Transfer

It also happens that a government agency is unable to procure new custom software – perhaps for budgetary reasons. In this case, the technology gap can be filled by a public-service organization that creates non-proprietary software. However, use by a government organization will still depend on several factors: a licensing scheme that gives the government a clear right to use; ready availability for use by the government's system integrator in its integration efforts; ability for the software to be adapted or customized by the system integrator to meet customer-specific needs. An OSS licensing and distribution model meets these needs.

5. Summary

This paper has described several ways that the phrase-term "open source" can be used (*and abused*) in ways that confound logical, fact-based technology policy discussions about government procurement of critical software. Such confounded discussions are a real problem, in cases where the phrase "open source" impedes appropriations or procurement of non-proprietary software, especially non-proprietary software that is a nearly inevitable part of any government-critical software acquisition.

We began with a half-dozen misperceptions or misunderstandings that need to be examined and understood with intellectual honesty in any discussion about the appropriateness of open source developed and distributed software technology. Such analysis begins with the fact-based observation that open-source technology is an essential part of most modern computing, particularly the open-source software ("OSS") that underlies most mobile Apps and web services used worldwide, and the public network infrastructure beneath that.

Defense and national-security-critical systems use non-proprietary software as foundational technology, and as part of outward technology transfer for government funded custom software. Non-proprietary software also has an important role to play in filling technology gaps for which the commercial market lacks an incentive to fill—again, both as foundational and custom software.

Election technology is mission critical. It is now well settled to be a matter of national security. OSS is an appropriate method and means of innovating election technology. OSS is neither necessary nor sufficient to increase integrity, lower cost, or improve usability of election technology, but it is an excellent means of increasing transparency and lowering cost in vital innovations required for critical election technology infrastructure.

This paper therefore, has made an effort to clarify a number of misunderstandings of open source software technology, and to explain its important role in mission critical computing, such as the technology of election administration and operation.

We invite and encourage your comments.
Write us: hello@osetfoundation.org

6. Selected Bibliography

1. INSIGHT-Tech firms let Russia probe software widely used by U.S. government (Reuters, January 25, 2018)
   https://www.reuters.com/article/usa-cyber-russia/insight-tech-firms-let-russia-probe-software-widely-used-by-u-s-government-idUSL1N1OD2GV

2. Code Review Isn't Evil. Security Through Obscurity Is. (EFF, January 30, 2018)
   https://www.eff.org/deeplinks/2018/01/code-review-not-evil-security-through-obscurity

3. Lily Hay Newman. "The Pentagon Opened Up to Hackers—And Fixed Thousands of Bugs." November 10, 2017.
   https://www.wired.com/story/hack-the-pentagon-bug-bounty-results/

4. Pierluigi Paganini. "Bug bounty programs and a vulnerability disclosure policy allowed Pentagon fix thousands of flaws." Security Affairs. November 13, 2017.
   http://securityaffairs.co/wordpress/65491/hacking/bug-bounty-program-pentagon.html

5. Response of Red Hat, Inc. to the Request for Comments Regarding Draft Report to the President on Federal IT Modernization (September 20, 2107)
   https://github.com/GSA/modernization/issues/58

6. DoD clarifying guidance to dispel FUD. On October 16, 2009, the DoD published clarifying guidance that sought to dispel the FUD of OS.
   https://web.archive.org/web/20100331033135/http:/cio-nii.defense.gov/sites/oss/2009OSS.pdf

7. Wheeler, David A. & Dunn, Tom. "Open Source Software in Government: Challenges and Opportunities." *DHS Science and Technology Directorate, Cybersecurity Division*. August 29, 2013. This white paper identifies key challenges and opportunities in the government application of OSS so that inappropriate roadblocks can be countered or mitigated. These challenges and opportunities were identified in interviews with experts, suppliers, and potential users, where users include both government contractors and government employees. These interviews were conducted in 2011 as part of the DHS Science and Technology Homeland Open Security Technology (HOST) project. See:
   https://www.dhs.gov/sites/default/files/publications/Open%20Source%20Software%20in%20Government%20%E2%80%93%20Challenges%20and%20Opportunities_Final.pdf
   and
   https://www.dhs.gov/science-and-technology/csd-host

