Choosing IT Platforms In The Age Of Stuxnet

Mohammad Dadashzadeh, Oakland University, USA

ABSTRACT

This paper addresses the question of choosing/investing in IT (hardware/software) platforms that avoid quick obsolescence and the underlying dilemmas of choosing proprietary software versus open source software, and opting for managed services such as public cloud computing versus in-house hardware/communication infrastructures. These dilemmas in strategic information systems planning have become more significant in light of the recent revelations of security backdoors in commercial software, encryption backdoors in communication software, and governmental access to private data on managed services for national security reasons. This paper considers enterprise-wide challenges and strategies for adopting open source software/hardware in response to these security concerns.

Keywords: Strategic Information Systems Planning; Information System Security; Cybersecurity; Cyber Espionage; Cyberwarfare; Stuxnet; Open Source Software; Defense in Depth

INTRODUCTION

Despite its complexity and pitfalls, long-term planning for information technology investments is widely recognized as a necessary undertaking for organizations. Indeed, effective strategic planning for information systems is increasingly regarded as a differentiating factor of successful enterprises. The importance of systematic planning for, and performance evaluation of, IT investments is more appreciated when one considers that a company's information system architecture, i.e., the totality of investments in information and communication technologies (ICT) in support of product development, service delivery, business processes, employee productivity, management planning, control, and decision making, is constrained by the following facts:

- It cannot be built overnight;
- There is no complete turnkey solution;
- No solution is final! Organizations evolve. So, should the IT supporting the organization;
- In many cases, decisions made today impact the organization's technology infrastructure for the next 3 or more years (Dadashzadeh, 2009).

Therefore, having an architectural blueprint that would allow change to be more readily implemented would be an invaluable asset.

Enterprise Architecture (EA) is a well-defined practice for describing the enterprise by providing blueprints of the enterprise from different perspectives including both systems and technology (Boar, 1998; Lankhorst, 2017). By documenting what each architectural element is, why it exists, how it functions, who it impacts, where it operates, when it is triggered, and its (multi-way) relationships with other architectural elements, it is hoped that the impact of changes as varied as changing the business model to drop a product line, or changing from a centralized database to a distributed one, or changing from Windows to Linux operating system can equally be easily understood and rapidly implemented. The complete documentation that EA demands, especially for capturing the multi-way relationships between architectural elements, remains an area of research. None of the established EA methodologies today offers a complete solution. Nevertheless, a fundamental impetus for EA, that is, aligning business strategy with the IT plan, can be accomplished via a more modest mechanism based on ensuring that the annual calendars for re-visiting/updating the organization's business plan and IT plan coincide, and that the IT plan is an input to the business planning process while the business plan is an input to the IT planning process.

Figure 1 presents such a conceptual model of alignment of business planning and IT planning (Dadashzadeh, 2009). The model depicts the inputs and outputs of the planning process emphasizing that while there are many established methodologies for IT planning, the output of each, i.e., organization's IT plan, must document the following deliverables: IT policies, objectives, and strategies; organizational information requirements; required application portfolio; consensus project ranking; and multi-year implementation schedule. And, that this collective output, i.e., organization's IT plan, becomes an input to the strategic business planning process which, in turn, defines *both* the desired future positions (such as core competencies, products/services, market share, etc.) the firm strives for *and* the strategic focus that IT should exercise in helping to realize them, as inputs to the IT planning process.

In a fundamental sense, the long-term plan for information technology investments, i.e., IT plan, guides the continuing development and sustainment of the application portfolio that supports the organization's product development, service delivery, business processes, employee productivity, management planning, control, and decision making, on a foundation built upon data integration and communication integration. This vision of the firm's information architecture is illustrated in Figure 2 (Dadashzadeh, 2009) where enterprise applications such as ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), SCM (Supply Chain Management), and PLM (Product Lifecycle Management) span multiple application portfolio categories (such as TPS, MRS, and DSS) depicted.

Beyond the obvious challenges for the IT planning process in determining the organizational information requirements and needed application portfolios in support of the organization's strategic business plan, an important consideration is the choice of the IT hardware and software platforms to build and sustain the firm's information architecture upon. This paper addresses the dilemma of choosing/investing in IT platforms that avoid quick obsolescence and questions the soundness of long established strategies for doing so in an age of cyberwarfare as exemplified by the Stuxnet virus. The remainder of this paper is organized as follows. In the next section, we consider the dilemma of choosing IT platforms in the context of strategic planning for information systems. Following that, information system software vulnerabilities from the early malwares to today's weaponized codes are briefly reviewed. Next, we address the shortcomings of information security best practices and the ramifications on the dilemma of choosing and securing IT platforms. We conclude with summary and closing considerations.

Figure 1. Conceptual Model of Alignment of Business Planning and IT Planning

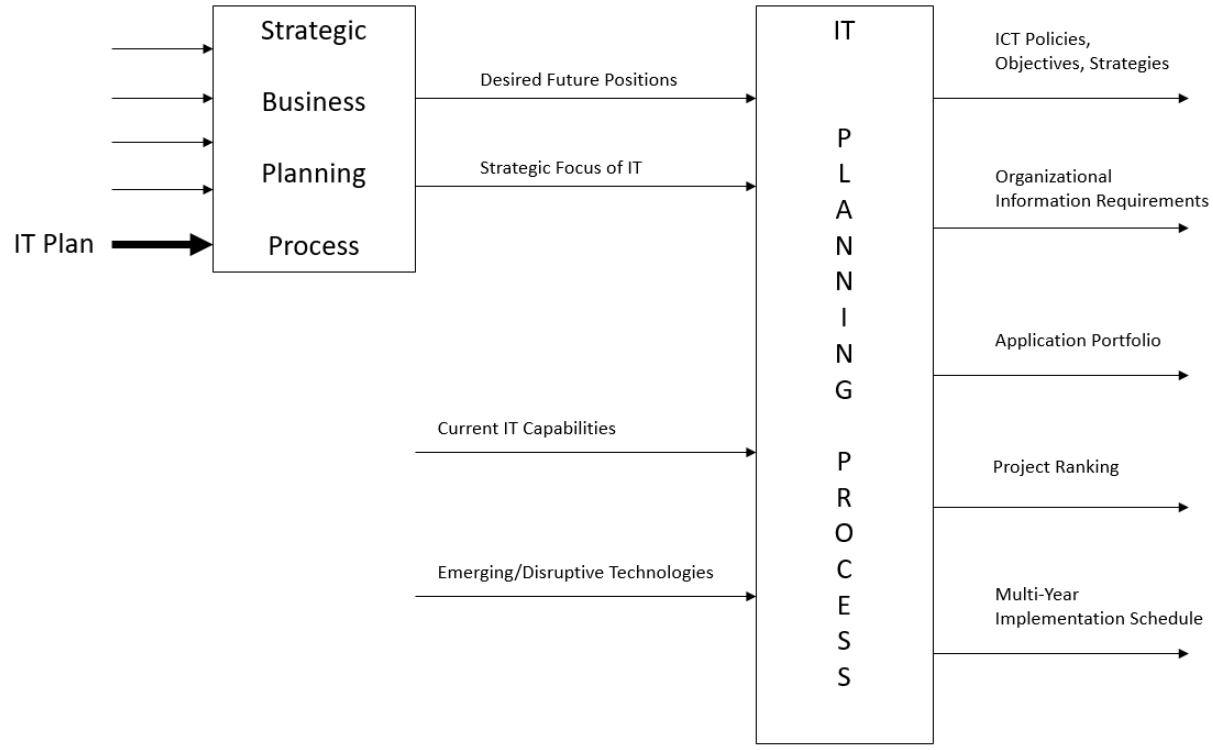
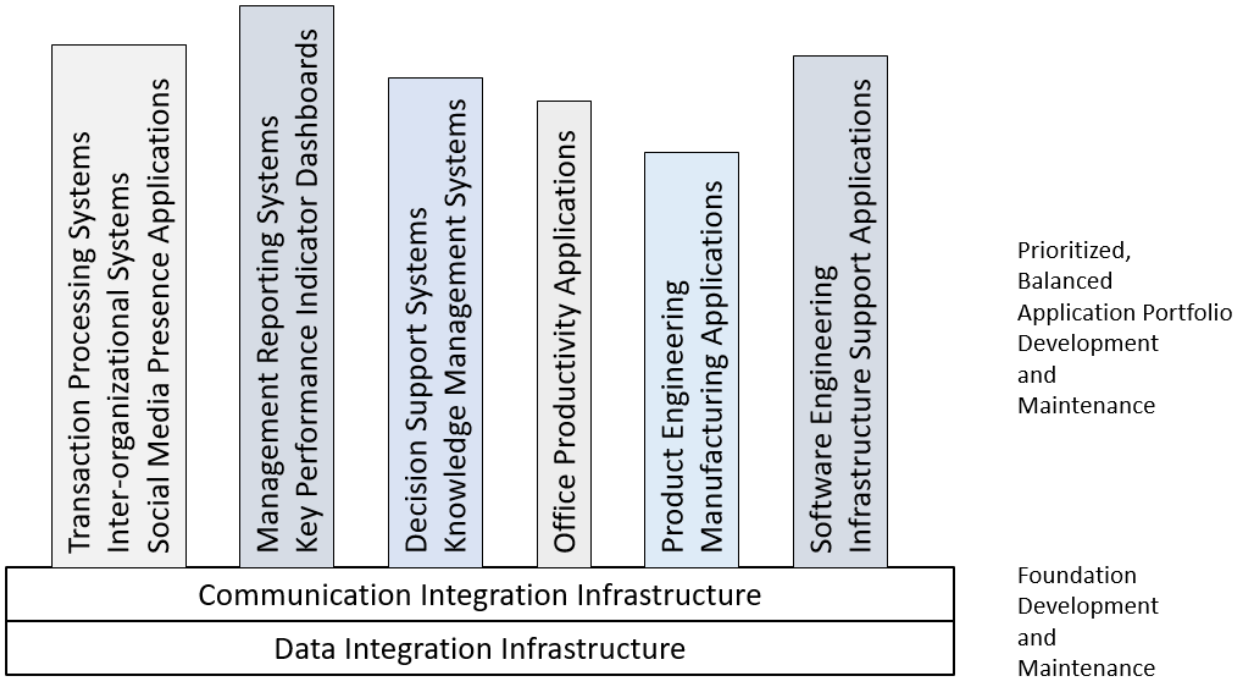


Figure 2. Information Architecture Vision for IT Investment Planning



CHOOSING IT PLATFORMS

The relatively brief history of information technology has shown that no hardware or software platform can be permanent. Advances in technology will force organizations to upgrade frequently and at times abandon/discard existing technology to remain efficient and/or competitive. Software applications, in particular, have undergone expensive re-writes when batch applications were replaced with real-time ones which, in turn, were later re-written to operate in client-server fashion. With the advent of Internet, client-server applications needed to be re-written to become browser-based. And, soon thereafter, the urgency of enterprise application integration demanded yet another re-writing of software to operate using service oriented architecture. Therefore, since all hardware and software platforms sooner or later become obsolete, it is only realistic to choose IT platforms that avoid *quick* obsolescence.

IT platforms can avoid quick obsolescence when they provide maneuverability, that is, when they manifest many of the following attributes (Boar, 2001):

- Maintainability. The ease of maintaining the platform.
- Adaptability. The ease of changing the platform.
- Flexibility. The ability to grow or contract the platform as required.
- Portability. The ability to move components within the platform.
- Modularity. The ability to add, modify, or remove pieces of the platform.
- Scalability. The ability to scale the platform by dimensions of number of sites, users, transactions, usage volume, etc.
- Interoperability. The ability to work cooperatively with multiple heterogenous components.
- Open Standards. The compliance of the platform with open standards that would allow interoperability and integration.

A maneuverable platform affords the organization more time and smoother transition when abandoning a platform becomes unavoidable. Nevertheless, the importance of actively gathered intelligence on emerging/disruptive

technologies as an input to the IT planning process (see Figure 1) cannot be overstated as it infuses maneuverability in planning information technology investments for building/acquiring the needed application portfolio.

It should not be surprising that maneuverability is negatively correlated with the cost of change. The less maneuverable a platform is, the more costs incurred in maintaining or changing it. That is, in large part, why mainframes were abandoned in favor of network computing and why on-premises networks of servers and PC's are increasingly being abandoned for cloud computing. Although the shift to the time-shared model reflected in cloud computing appears to be a pendulum swing back to the days of time-shared mainframes, the outward similarity ignores the considerable maneuverability gains afforded by cloud computing. Indeed, the extent of use of managed services, including software-as-a-service, to reduce the cost of IT operations support and maintenance to be able to budget for more growth initiatives is an important output of the IT planning process.

The ICT policies, objectives, and strategies output of the IT planning process (see Figure 1) formulate a variety of guidelines needed for managing the development and maintenance of the organization's information architecture. They include:

- Budgeting for IT (e.g., baseline or zero-based, centralized or charged back)
- Centralization versus decentralization of IT support/services
- Customized development versus off-the-shelf packages
- Outsourcing versus in-house development/maintenance
- IT project specification change control policies
- Managed services guidelines
- Hardware/software standardization policies
- End-user computing policies
- Bring-Your-Own-Device policies
- Social media usage policies
- Data privacy policies
- Information security policies
- Disaster recovery planning and accountability
- Guidelines for external audit of IT performance

Clearly, many of the above guidelines impact the choice of IT hardware and software platforms. Whether it is the choice of the standard laptop issued to the employees, or the mobile phone operating system standardized upon, or the wireless hardware infrastructure employed on premises, or the antivirus software deployed at each endpoint, maneuverability remains an appropriate measure of the right choice. Even important selection considerations for a hardware/software component such as viability of its vendor, its reliability, and its security are captured in the maintainability attribute of maneuverability. The less secure (or reliable) a hardware/software component is, the less maintainable it becomes, and thereby the less maneuverable it is. The interesting question that arises, which is addressed in the next section, is that under what circumstances the security attribute trumps all other considerations for maneuverability?

SOFTWARE SECURITY

A very conservative estimate would place the number of bugs in most modern software at one per 1,000 lines of well-written source code (Perrin, 2010). Windows NT 3.1 was released in 1993 with 4.5 million source lines of code (SLOC). Using the conservative estimate of 1 bug per 1,000 SLOC, NT 3.1 had 4,500 security vulnerabilities. Security updates/patches help reduce the number of discovered bugs in software, but bug fixes are not immune to containing bugs. Some estimate that 10% of security patches actually introduce new vulnerabilities. The implications are especially disarming when one considers the rate at which most software grows in SLOC to add new features. As a case in point, the Windows NT-based operating system releases grew at an annual average rate of 4.5 million SLOC with Windows Server released in 2003 having 50 million SLOC.

Security vulnerabilities in software can be exploited to implant Trojan code (named after the wooden horse the Greeks used to infiltrate Troy) that is specifically designed to damage, disrupt, steal, or in general inflict some other "bad" or

illegitimate action on data, hosts, or networks (Cisco, 2017). Executing legitimate software infected with Trojan code allows it to utilize the authorization and access rights of the host software to achieve any number of attacks on the host and its accessible resources including creating backdoors giving malicious users/programs access to the system, or activating and spreading other malware such as viruses and worms. Although starting as a mere computing challenge to write self-replicating software, malware has grown to become a serious threat that no individual, or organization, or IT platform is immune from as the following highlights from the "timeline of computer viruses and worms" demonstrate (Wikipedia, 2017a).

In 1971, the concept of self-replicating software was demonstrated by the experimental "Creeper" program on DEC PDP-10 computers running the TENEX operating system and connected via the ARPANET. The first Trojan appeared in 1975 as a legitimate game on a multi-user UNIVAC. When executed by a user, it would surreptitiously copy itself to every directory the user had access to. It spread on the computer when users with overlapping directory permissions would discover the game and run it, and it spread "in the wild" to other UNIVACs when tapes were shared. In 1982, "Elk Cloner" became the first microcomputer virus, a boot sector virus, to spread in the wild. When an Apple II booted from an infected Apple DOS 3.3 floppy disk, a copy of the virus was placed in the computer's memory. When an uninfected disk was inserted into the computer, the entire DOS (including boot sector and Elk Cloner) would be copied to the disk, infecting it and allowing the virus to spread from disk to disk (Wikipedia, 2017b).

The term "computer virus" was coined in 1984 by Frederick Cohen as "a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself" (Cohen, 1987). He demonstrated a virus-like program on a VAX 11/750 minicomputer. In 1984, Ken Thompson described the modification of a C compiler that when used to compile the Unix operating system, would insert a backdoor into the login command (Wikipedia, 2017c). In his Turing Award Lecture, he asked "To what extent should one trust a statement that a program is free of Trojan horses?" and offered that "Perhaps it is more important to trust the people who wrote the software" (Thompson, 1984).

In 1986, the first IBM PC compatible virus epidemic was caused by the "Brain" (aka "Pakistani Flu") boot sector virus. Also in 1986, a working model of Virdem, the first file virus for the DOS operating system, that could replicate itself via addition of its code to executable DOS files in .COM format was presented to the Chaos Computer Club, an underground hacker forum, in Germany. The exponential proliferation of both boot sector and file viruses and their increasingly destructive payloads led to the emergence of the antivirus software industry and the continual "Race to Zero," which refers to the hacker challenge of developing zero-day exploits – malware that is so new that detection/protection against it does not exist yet. The "Cascade" virus, widespread in late 1980s and early 1990s, is notable for being first in using an encryption algorithm to avoid being detected. "Ghostball," discovered in 1989, was the first multipartite virus infecting both executable .COM files and boot sectors. The first known ransomware, AIDS Trojan, appeared in 1989 on several thousand floppy disks mailed to subscribers of PC Business World magazine and a World Health Organization AIDS conference mailing list (Wikipedia, 2017a).

The "Morris worm" or Internet worm, released in November 1988, infected DEC VAX and Sun machines running BSD Unix connected to the Internet. It became the first worm to spread extensively in the wild, gaining considerable mainstream news coverage, and eventually resulting in the first felony conviction under the U.S. Computer Fraud and Abuse Act. It worked, in part, by exploiting buffer overflow vulnerabilities in Unix. Known security vulnerabilities in the indexing software included as part of Microsoft's Internet Information Server (IIS) were exploited in August 2001 by the "Code Red II" worm that installed a backdoor to allow attacks. Interestingly, Microsoft had already released a security fix on June 18, 2001, "however not everyone had patched their servers, including Microsoft" (Wikipedia, 2017d). In 2003, the "SQL Slammer" worm exploited vulnerabilities in Microsoft SQL Server causing massive Internet access disruptions worldwide. In 2008, "Conficker" infected anywhere from 9 to 15 million Microsoft servers leading to Microsoft's setting "a bounty of $250,000 for information leading to the capture of the worm's author(s)" (Wikipedia, 2017a). Although almost all advanced malware techniques used by Conficker were well known, its combined use of many made it unusually difficult to eradicate.

In many ways, the discovery of the "Stuxnet" virus in 2010 (Kushner, 2013) is a watershed event for software security and the strategic planning dilemma of choosing IT platforms. Stuxnet was designed to target Siemens Step7 software on computers attached to programmable logic controllers (PLCs) which allow the automation of electromechanical processes such as those used to control machinery on factory assembly lines, computerized prison doors, amusement

rides, or centrifuges for separating nuclear material (Wikipedia, 2017e). Stuxnet would introduce an infected rootkit to the PLC and Step7 software, modifying programs and giving unexpected commands to the PLC while returning normal operation feedback values to users. Reportedly, one fifth of Iran's fast-spinning nuclear centrifuges in Natanz plant were ruined in this manner. Several facts regarding Stuxnet are especially noteworthy (Sanger, 2012; Zetter, 2014):

- It "was signed by a digital certificate to make it appear that it had come from a reliable company" (Kushner, 2013).
- It used 20 zero-day exploits including two vulnerabilities having to do with privilege escalation that allowed Stuxnet "to gain system-level privileges even when computers had been thoroughly locked down" (Kushner, 2013).
- The Windows shared print-spooler vulnerability used by Stuxnet to spread in networks with shared printers (Kushner, 2013), was in fact publicly discussed in April 2009, but remained unpatched by Microsoft until September 2010.
- In 2007, a virtual replica of Natanz plant was built at American National Laboratories allegedly to test Stuxnet prior to its release.

The above facts combined with other revelations have led many experts to believe that Stuxnet was an American cyberweapon built to sabotage Iran's nuclear program in what would appear as a series of unfortunate accidents (Wikipedia, 2017e). As such, the apparent accommodation of software companies to supply digital certificates or not patch known vulnerabilities for Stuxnet usage should be considered a natural expectation/duty in support of the country's national security interests. Indeed, the subsequent discovery of "Duqu," a worm thought to be related to the Stuxnet, in 2011, and "Flame," a modular computer malware 20 times more complicated than Stuxnet used for targeted cyber espionage, in 2012, have made Ken Thompson's 1983 Turing Award Lecture on *Reflections on Trusting Trust* eerily relevant today. Can you trust code that you did not totally create yourself?

Governments' single-minded obsession to weaponize software and surveil all communication in the interest of cybersecurity come with the price of leaving software vulnerable to hacking and our privacy open to unlawful surveillance. Therefore, it was not surprising when in 2012, U.S. corporation Chevron admitted that Stuxnet had invaded its machines. Or, when in May 2017, the ransomware "WannaCry," utilizing hacking tools designed by the Equation Group, a group that is widely suspected of being tied to the U.S. National Security Agency (NSA), held PCs at Telefonica in Spain and NHS Hospitals in England hostage.

Edward Snowden's revelations about NSA's successful efforts to obtain virtually unchecked surveillance power included (Franceschi-Bicchierai, 2014; Greenwald, 2014):

- Secret court orders allow NSA to sweep up Americans' phone records;
- NSA's data collection program named PRISM collects Internet communications from at least 9 major U.S. Internet companies;
- Britain's version of the NSA taps fiber optic cables around the world;
- NSA spies on foreign countries and world leaders;
- NSA's XKeyscore program allows analysts to mine "nearly everything a typical user does on the Internet" by filling in a simple on-screen form giving only a broad justification for the search with no further review;
- NSA's efforts to crack encryption and undermine Internet security;
- NSA's Advanced Network Technology (ANT) catalog of devices for cyber surveillance in support of its elite hacking team named Tailored Access Operations (TAO);
- NSA cracks Google and Yahoo data center links;
- NSA collects text messages;
- NSA intercepts all phone calls in two countries (Franceschi-Bicchierai, 2014).

As impressive as the above list of achievements is, it is NSA's cooperative agreements and "tailored solutions" with IT companies that give it the proverbial competitive edge (Harris, 2015). NSA helps companies find weaknesses in their products. In return, it asks them not to fix some of those. Those vulnerabilities are then utilized by teams such as the Equation Group to create hacking tools for spying or attacking foreign governments that install the products in their

intelligence agencies, their militaries, and their critical infrastructure. In 2015, "Kaspersky Lab's research findings on another highly sophisticated espionage platform created by the Equation Group, noted that the group had used two of the same zero-day attacks used by Stuxnet *before* they were used in Stuxnet, and their use in both programs was similar" (Wikipedia, 2017e), leading the researchers to conclude that "the Equation Group and the Stuxnet developers are either the same or working closely together" (Kaspersky Lab, 2015).

Of course, backdoors and unpatched security vulnerabilities could also be used by hackers. The 2016 release of hacking tools attributed to NSA's Equation Group by the Shadow Brokers, a hacker group leaking the information, made several exploits including one named EthernalBlue (using a Windows server message block vulnerability) available on the Internet. Both the "WannaCry" ransomware of May 2017 and its improved, more lethal version, the "Petya" ransomware released in June 2017 took full advantage of the availability of EthernalBlue. NSA, of course, does not acknowledge that its tools are used in these attacks, nevertheless pressure is increasing on it to help the world defend against the weapons it continues to disavow.

Given this new age of cyber espionage, when unpatched security flaws and even Trojan codes may be lurking in system software such as an operating system, or in application software such as an accounting package, under what circumstances should the concern for security vulnerability trump all other merits of an otherwise highly maneuverable platform? The simplest answer is perhaps the organization's risk of, and recovery cost from, becoming a target of industrial or military espionage. The greater the risk of being a candidate for targeted espionage, the greater should be the weight of favoring a more secure, albeit less maneuverable platform. And, based on the fundamental observation that "you cannot trust code that you did not totally create yourself," open source software becomes the alternative to choose.

SECURING IT PLATFORMS

Open source software (such as Linux) is considered more secure than its closed source counterparts (such as Windows) for several reasons:

- Open source software (OSS) is available for inspection and can be audited when its security is in doubt.
- Although not necessarily better engineered or bug-free, open source software can be checked without blindly trusting the people who wrote the software.
- Open source software allows anyone to fix/patch broken code while closed source software can only be patched when the vendor chooses to do so.
- Over time, open source software becomes more secure since more software developers are inspecting, testing, fixing, and verifying the code.

Therefore, if securing the IT platform is important, an organization can invest in the resources necessary to verify open source software before deploying it.

Although, building the organization's information architecture (see Figure 2) *entirely* on open source software may not be feasible, critical infrastructure pieces are available as open source. These include: operating systems for servers, desktops, and mobile devices; web browsers; web servers; database management systems; programming language compilers; domain name system (DNS) servers; networking router operating systems (firmware); and encryption software for secure communications and digital signatures. Having a verified, secure open source software infrastructure in place allows greater confidence in information security best practices that still must be employed. The layered security practices (Conklin, 2016), for example, assume that any single defense may be flawed, and thus an aggregate of different types of defenses should be used to cover the gaps in the others' protective capabilities. Layered security is not about redundancy, but instead it is about multiple types of security measures, each protecting against a different attack vector. In the layered approach to security, firewalls, intrusion detection systems, malware scanners, encryption, strong passwords, and security auditing procedures can each serve to protect the IT resources in ways the others cannot.

Layered security is one component of a defense-in-depth strategy that acknowledges achieving total, complete security is not possible. Instead, the various technological components of a layered security defense should be in place to hinder the progress of a threat until it ceases to threaten or some other, perhaps non-technological measures can be brought forth to

neutralize/eradicate it. Defense-in-depth is a comprehensive security strategy approach that addresses the variety of elements that need to be in place (including multiple layers of security) to protect against known and emerging threats. These elements include (Department of Homeland Security, 2016):

- Risk Management Program – Identifying threats, characterizing risks, maintaining asset inventory, change management
- Cybersecurity Architecture – Standards, policies, procedures
- Physical Security – Access control to server rooms, escorting visitors, barriers, locking control panels and cabling, remote site video monitoring and logging
- Network Architecture – Demilitarized zones, virtual LANs
- Network Perimeter Security – Firewalls, one-way diodes, remote access authentication, jump servers, intrusion detection and prevention system
- Host Security – Computer hardening, authentication and authorization control, patch and vulnerability management, endpoint security protection
- Device Security – Device hardening, protecting the actual hardware, controllers, drives, motors, valves, and other automation and control devices
- Software Development Security – Specification control, design review, code review and walk-through, application/acceptance testing review, and maintenance/change management
- Security Monitoring and Analytics – Intrusion detection systems, security audit logging, security incident and event monitoring, security analytics, user behavior analytics
- Vendor Management – Supply chain management, managed services/outsourcing, cloud services usage management
- Human Factor Management – Policies, procedures, training and awareness

Defense-in-depth is thus a holistic approach that employs an organization's available resources to provide effective layers of cybersecurity threat identification, prevention, monitoring, detection, protection, and recovery for all assets. A robust defense-in-depth strategy begins proactively by formulating policies and procedures, training individuals for security awareness and accountability and with the know-how to do their jobs securely, identifying threats and assessing risks, developing a comprehensive inventory of assets, applying security controls based on asset priority, deploying effective countermeasures to manage vulnerabilities, enforcing incident reporting and response, and incorporating lessons learned throughout the process.

It is important to emphasize that using open source software combined with implementing a defense-in-depth security strategy cannot protect against all vulnerabilities and weaknesses in securing IT platforms. Nevertheless, they provide the organization with a fighting chance against emerging threats of an intrusion by malicious actors (insiders as well as outsiders) on the organization's information architecture using computer-based exploits. Perhaps, a measure of paranoia is the key to success in security and an appropriate acid test for your security strategy by remembering that:

- "You cannot trust code that you did not totally create yourself" (Thompson, 1984).
- You cannot trust executable code on any platform that you did not compile yourself.
- You cannot trust open source software that you compiled with a closed source compiler.
- You cannot trust hardware/software installations/changes/upgrades that you did not perform yourself.
- You cannot trust hardware devices with firmware that you did not write/compile yourself.
- You cannot trust any piece of hardware to be free from tampering introduced in the supply chain to subvert it.
- You cannot trust software alone to create security solutions that cannot be undermined by the underlying hardware.

In short, when it comes to securing IT platforms in the age of Stuxnet and cyber espionage, it is time to re-think the truism of *trust but verify*. Indeed, as difficult as it may be, one must heed the advice of Cheswick and Bellovin (1994) that "any program, no matter how innocuous it seems, can harbor security holes. We thus have a firm belief that everything is guilty until proven innocent."

CONCLUSION

The discovery of the Stuxnet virus in 2010 brought to the limelight the new age of cyberwarfare and weaponized code. Revelations that trusted software companies like Microsoft would leave some zero-day vulnerabilities unpatched to accommodate national security agencies to build hacking tools for spying or attacking foreign governments that had installed their "trusted" products in their intelligence agencies, their militaries, and their critical infrastructure (Harris, 2014), foretold unintended consequences in exposing all of us to criminal hacking and ransomware. Governments' efforts to invade our privacy in the name of cybersecurity and to undermine encryption standards and digital certificates are making the Internet less secure, and all of us much more vulnerable to unlawful surveillance. In such an environment, building an organization's information architecture on hardware/software platforms from "trusted" vendors should no longer be considered as maintainable or as maneuverable as before, and open source platforms that allow verification would present the more judicious choice.

Open source software is by no means a panacea. Verification is expensive but one retains more control and there will be less costly surprises. However, not only open source software must be verified, but it must also be compiled with an open source compiler that, in turn, should be verified before the resulting executable code can be trusted. And, since the ability to infect hard drive firmware by reprogramming it and creating hidden sectors (for persistent data storage) that survive disk formatting and operating system re-install has been proven by the Equation Group (Kaspersky Lab, 2015), proprietary firmware in any hardware device (including routers) becomes a security risk and must be replaced with an open source version that can be verified. Unfortunately, backdoors and Trojan code are not limited to software (Karri, Rajendran, Rosenfeld, & Tehranipoor, 2010). It has been shown that hardware can be tampered with in the supply chain to surreptitiously collect and ultimately exfiltrate data (Payton, 2017), and that backdoors can be fabricated into the silicon chip design (Skorobogatov & Woods, 2012). Therefore, open source software combined with a comprehensive defense-in-depth security strategy would be needed for securing IT platforms.

The age of Stuxnet, however, has given rise to a much bigger concern than a re-thinking of maneuverable IT hardware and software platforms that avoid quick obsolescence. Despite its made for movie allure, cyberwarfare risks causing catastrophe. Today's nuclear missile command-and-control systems must contend with backdoors in silicon, corrupted firmware, Trojan code, and all the other tools of cyberwarfare (Schlosser, 2013). In a recent paper, Andrew Flutter (2016) suggests that "a nuclear command-and-control system might be hacked to gather intelligence about the system," to shut it down, to spoof it, to mislead it, or to cause it to launch a missile. Could a foreign agent launch another country's missiles against a third country? Could false early warning data corrupted by hackers set off the launch of a nuclear missile? These are the unintended consequences of cyberwarfare that are, unfortunately, likely to be addressed only after a catastrophe. It is high time we debated the unintended consequences of governments' single-minded obsession to weaponize software or to invade our privacy in the name of cybersecurity – even if it may only be posthumously.

AUTHOR BIOGRAPHY

Mohammad Dadashzadeh is a professor of Management Information Systems at Oakland University in Rochester, Michigan. His interest in software security can be traced back to unlocking copy protection schemes of cassette-based programs on Radio Shack TRS-80 Microcomputer. He is a co-author of Iranian Standard Code for Information Interchange and a co-founder of Sinasoft. Although computer programming remains a protected hobby, Dr. Dadashzadeh's education, research, and consulting have emphasized enterprise-wide data integration and strategic IT planning. He has served as the editor-in-chief of *Journal of Database Management* and more than 6,000 people have attended his professional training seminars worldwide.

REFERENCES

Boar, B. (1998). *Constructing blueprints for enterprise IT architectures.* New York, NY: Wiley.
Boar, B. (2001). *The art of strategic planning for information technology.* New York, NY: Wiley.
Cheswick, W.R., & Bellovin, S.M. (1994). *Firewalls and internet security.* Boston, MA: Addison-Wesley.
Cisco. (2017). What is the difference: Viruses, worms, trojans, and bots? Retrieved 07/01/2017 from http://www.cisco.com/c/en/us/about/security-center/virus-differences.html
Cohen, F. (1987). Computer viruses: Theory and experiments. *Computers & Security, 6*(1), 22-35.

Conklin, W.A. (2016). *Principles of computer security.* New York, NY: McGraw-Hill.
Dadashzadeh, M. (2009). A new methodology for developing the MIS Master Plan. *Review of Business Information Systems, 13*(1), 15-23.
Department of Homeland Security. (2016). *Recommended practice: Improving industrial control system cybersecurity with defense-in-depth strategies.* Retrieved 07/01/2017 from https://ics-cert.us-cert.gov/sites/default/files/recommended_practices/NCCIC_ICS-CERT_Defense_in_Depth_2016_S508C.pdf
Flutter, A. (2016). *Cyber threats and nuclear weapons: New questions for command and control, security and strategy.* London, UK: Royal United Services Institute for Defense and Security Studies.
Franceschi-Bicchierai, L. (2014). *The 10 biggest revelations from Edward Snowden's leaks*. Retrieved 07/01/2017 from http://mashable.com/2014/06/05/edward-snowden-revelations/#i2YG4TE3MPq5
Greenwald, G. (2014). *No place to hide: Edward Snowden, the NSA, and the U.S. surveillance state.* New York, NY: Metropolitan Books.
Harris, S. (2014). *Google's Secret NSA Alliance: The Terrifying Deals between Silicon Valley and the Security State*. Retrieved 07/01/2017 from http://www.salon.com/2014/11/16/googles_secret_nsa_alliance_the_terrifying_deals_between_silicon_valley_and_the_security_state/
Harris, S. (2015). *@War: The Rise of the Military-Internet Complex.* Boston, MA: Houghton Mifflin Harcourt.
Karri, R., Rajendran, J., Rosenfeld, K., & Tehranipoor, M. (2010). Trustworthy Hardware: Identifying and Classifying Hardware Trojans. *Computer, 43*(10), 39-46.
Kaspersky Lab. (2015). *Equation group: Questions and answers.* Moscow, Russian Federation: Kaspersky Lab.
Kushner, D. (2013). The real story of Stuxnet. *IEEE Spectrum, 50*(3), 48-53.
Lankhorst, M. (2017). *Enterprise architecture at work: Modeling, communication and analysis.* Berlin, Germany: Springer.
Payton, I. (2017). *Hacking the Belkin E Series OmniView 2-Port KVM Switch.* Retrieved 07/01/2017 from http://www.talosintelligence.com/files/publications_and_presentations/papers/Talos_BelkinWhitePaper.pdf
Perrin, C. (2010). *The danger of complexity: More code, more bugs.* Retrieved 07/01/2017 from http://www.techrepublic.com/blog/it-security/the-danger-of-complexity-more-code-more-bugs/
Sanger, D. (2012). *Confront and conceal: Obama's secret wars and surprising use of American power.* New York, NY: Random House.
Schlosser, E. (2013). *Command and control: Nuclear weapons, the Damascus Accident, and the illusion of safety.* New York, NY: The Penguin Press.
Skorobogatov, S., & Woods, C. (2012). Breakthrough silicon scanning discovers backdoor in military chip. *Lecture Notes in Computer Science, 7428*, 23-40.
Thompson, K. (1984). Reflections on trusting trust. *Communications of the ACM, 27*(8), 761-763.
Wikipedia. (2017a). *Timeline of computer viruses and worms.* Retrieved 07/01/2017 from https://en.wikipedia.org/wiki/Timeline_of_computer_viruses_and_worms
Wikipedia. (2017b). *Elk Croner.* Retrieved 07/01/2017 from https://en.wikipedia.org/wiki/Elk_Cloner
Wikipedia. (2017c). *Backdoor (computing).* Retrieved 07/01/2017 from https://en.wikipedia.org/wiki/Backdoor_(computing)
Wikipedia. (2017d). *Code Red II.* Retrieved 07/01/2017 from https://en.wikipedia.org/wiki/Code_Red_II
Wikipedia. (2017e). *Stuxnet.* Retrieved 07/01/2017 from https://en.wikipedia.org/wiki/Stuxnet
Zetter, K. (2014). *Countdown to Zero Day: Stuxnet and the launch of the world's first digital weapon.* New York, NY: Crown Publishers.