A Twenty-Five Year Perspective

Karen Mercedes Goertzel, Booz Allen Hamilton

Abstract. The security risks associated with software and its development processes have been recognized for 40 years or more. But only in the past quarter century have efforts to understand and address the root causes of system security vulnerabilities evolved and coalesced into systematic efforts to improve software security assurance across government and leading industry sectors. Along with these programs have arisen efforts to reshape the software engineering profession, and to establish a robust software security technology and services industry.

This article provides a capsule history of the most significant of the software assurance efforts of the past 25 years, organized by the main problems they have striven—and continue to strive—to correct. At the end of the article, a number of more extensive, detailed software assurance landscapes are recommended to the reader, to complement and elaborate upon the information presented here.

Background

In 1974, a vulnerability analysis of the Multics multilevel secure operating system highlighted the potential of software design flaws and coding errors to be exploited as a means to compromise the security of the Multics system [1]. The report also discussed the potential for malicious insiders and external penetrators to exploit the lack of security awareness in Software Development Life Cycle (SDLC) processes and the absence of security protections for code development and distribution mechanisms to surreptitiously access and subvert the code prior to deployment. These process-level weaknesses were true not just for Multics, but for all of the software that made up the DoD's World Wide Military Command and Control System.

The 1974 report may be the first formal documentation of the direct correlations between (1) errors and flaws in a system's software and the vulnerability of that system, and (2) the lack of security controls in SDLC processes and the potential for malicious subversion of the software that results. However, the report's matter-of-fact tone suggests both problems were likely already well-recognized by then. And so the twin concerns that continue to drive virtually all software security assurance efforts to this day were already documented by 1974.

Over the next 20 years or so, any focus on improving software-level security assurance was limited to software-intensive systems with very high-confidence requirements used in the DoD, the Department of Energy, and the intelligence community (and in some of their non-U.S. counterparts abroad), e.g., the ballistic missile defense software developed under the Strategic Defense Initiative (SDI), and software used in high-assurance cryptographic systems, operating system kernels, and cross-domain solutions. Not until the mid-1990s did the broader security implications of the poor quality of most software explode into the broader consciousness. This awareness came thanks to the coincidence of the rise of universal Internet connectivity and the World Wide Web (and with it the exposure of increasing amounts of software that had previously operated only standalone or on private networks) with the global undertaking to examine and correct Y2K errors in the vast installed base of commercial and privately-produced software code. People were looking harder at their software than ever before, and what they found was not reassuring.

One result of the recognition that most software contained entirely too many exploitable errors and flaws was a deeper investigation into the root causes of the problem and, once identified, into the means to correct them. As a result, the late 1990s onward saw a growing ferment of commercial, academic, and government activity, including research, policy, process improvement, and propaganda—all falling under the rubric of "application security" or "software assurance."

By 2005, the President's Information Technology Advisory Committee was able to neatly summarize the twin security dilemmas that plague modern software:

"Today, as with cancer, vulnerable software can be invaded and modified to cause damage to previously healthy software, and infected software can replicate itself and be carried across networks to cause damage in other systems.... Vulnerabilities in software that are introduced by mistake or poor practices are a serious problem today. In the future, the Nation may face an even more challenging problem as adversaries—both foreign and domestic—become increasingly sophisticated in their ability to insert malicious code into critical software [4]."

The ability to exploit software's vulnerabilities to compromise its availability and the confidentiality and integrity of the information it handles, and the ability to exploit vulnerabilities in SDLC processes to intentionally subvert the functionality produced by those processes (by tampering with intended logic or implanting malicious code) continue to provide the impetus for all of today's software and software supply chain security assurance efforts. The rest of this article describes a representative sampling of such efforts over the past 25 years.

Exploitation of Software Vulnerabilities

The National Bureau of Standards published one of the first major taxonomies of operating system security vulnerabilities in 1976. Of the seven categories of vulnerabilities it identified, five constituted software vulnerabilities—(1) inadequate input/parameter validation, (2) incorrect input/parameter validation, (3) bounds checking errors, (4) race conditions, and (5) other exploitable logic errors [3]. The other two vulnerability categories were information flow-related.

By the mid-1990s, researchers recognized the need to clearly delineate software security from information security concerns, and several proposed taxonomies categorizing and characterizing software vulnerabilities were published by the Naval Research Laboratory, [5] Purdue University [6], the Open Web Application Security Project (OWASP) [7], and The MITRE Corporation [8].

In parallel with these efforts to "taxonomize" software vulnerabilities arose attempts to characterize techniques for exploiting software vulnerabilities, and tools and techniques such as attack trees [9], attack patterns [10], and threat modeling [11]

emerged to help developers characterize the attacks and exploitations that were most likely to target their software, and their likely outcomes.

Meanwhile, the typical timeframe between a vendor's discovery of a new vulnerability in its commercial software product and its ability to develop and release a fix, or "patch," to mitigate that vulnerability shrank from months to weeks to days to virtually nothing. This "zero-day vulnerability" problem, which had only been speculated about at the start of the millennium, was commonplace reality by the end of its first decade. In a struggle to maintain even tenuous control over the situation, both the software industry and the government began demanding exclusive rights to information about vulnerabilities discovered by their own and third-party "security researchers." In 2005, the first known sale of a vulnerability occurred. An ex-employee of the NSA sold information about an exploitable Linux flaw on an exclusive basis to the U.S. government; the alleged price: $80,000. Today, vendors routinely offer bounties for exclusive information about vulnerabilities in their products (Google reportedly spent upward of $460,000 in the first two years of its Vulnerability Reward Program). Buyers are motivated by the desire to keep news of vulnerabilities quiet long enough for patches to be released and applied, while many researchers seek to turn vulnerability-selling into a profitable industry. Some even market subscriptions to vendors whose software is affected. Others focus on the lucrative government market for vulnerabilities that can be exploited in information operations or cyber espionage [12].

Attacks targeting or exploiting software bugs, and the variety and capability of malicious logic have increased exponentially with the proliferation of network-connected software-intensive systems, services, and applications, including embedded systems. These risks plague not only the embedded software and microcode in military weapon systems, but in industrial control systems, networking devices, medical devices, onboard vehicle and avionic diagnostic systems, global positioning systems, mobile communications devices, consumer electronics, and an growing number of "smart" appliances in homes and workplaces. Many such systems are expected to operate continuously and cannot tolerate operational disruptions. A growing number are peripatetic, with no fixed location and only intermittent wireless connectivity. As a result, all are poor candidates for the traditional "push" approach to "just in time" software patching and updating.

At the same time, with miniaturization, hardware has also become so powerful that the lines between embedded software, firmware, and "fused-in" hardware logic have increasingly blurred. Researchers have also demonstrated the ability to load malicious firmware into information and communications technology (ICT) devices in order to subvert their operation. For example, a Columbia University research team installed malicious firmware in an HP LaserJet printer, then used it to illicitly forward documents from the print queue, and also to physically damage the printer [13]. Indeed, the problem of malicious firmware was explicitly documented by Scott Borg, the Director of the U.S. Cyber Consequences Unit and the Internet Security Alliance, in a 2008 strategy paper for the White House [14].

The emergence of post-manufacture reprogrammable integrated circuits in the 1990s obscured these distinctions even further, by expanding the threats to hardware logic beyond its fabrication and manufacturing processes. While all integrated circuits (ICs) are vulnerable to subversion during design and manufacture, field-programmable gate arrays (FPGAs) extend the attacker's window of opportunity, because their logic can be maliciously altered after manufacture. As long ago as 1999, researchers identified techniques for implanting, and resulting effects of, "FPGA viruses" [15], and demonstrated the ability to alter the bitstream used to reprogram the FPGA to insert malicious logic into its main memory [16]. A few years later, 2007, researchers at University of Illinois at Urbana-Champaign proved the feasibility of maliciously modifying non-reprogrammable IC logic to add post-deployment-exploitable "hardware Trojans" and "kill switches" [17].

In 2012, the Defense Advanced Research Projects Agency (DARPA) initiated its Vetting Commodity Information Technology (IT) Software and Firmware program "to look for innovative, large-scale approaches" for verifying that the software and firmware embedded in commodity IT devices purchased by DoD are "free of hidden backdoors and malicious functionality" [18]. In addition, software code analysis tool vendors such as Grammatech are expanding their products to support inspection of firmware for presence of vulnerabilities and malicious logic.

The need to expand the definition of "software" to include firmware and hardware logic reinforces the needs to also expand the focus of "software assurance" to address management of security risks in the supply chains for commercial software and hardware, as consumers—in DoD and beyond—continue to increase their reliance on COTS software, and reduce the amount of custom-development that allows them full lifecycle visibility and control over how their logic-bearing products are built and distributed.

Inadequate SDLC Processes and Technologies

In 1985, Canadian computer scientist David Lorge Parnas felt compelled to resign his position with the Strategic Defense Initiative Organization (SDIO) Panel on Computing in Support of Battle Management. In his letter of resignation he explained why he could no longer in good conscience associate himself with the SDI software development effort [19]. Given what was at stake—preventing a nuclear holocaust—SDI software could not afford to be less than 100 percent dependable. And 100 percent dependable software was (and still is) an impossibility. SDI software was so unprecedentedly huge and complex, Parnas explained, and its development methodology was so problematic, that any attempt to build assurably trustworthy SDI software was doomed to fail. Much of the fault lay in the limitations of conventional software development approaches—limitations that could not be overcome by the also-deficient emerging techniques of artificial intelligence, automatic programming, and formal methods. Parnas' letter provided the impetus for the SDIO to reconsider how its software would be developed. In 1990 two SDIO researchers published the Trusted Software Development Methodology (TSDM)—arguably the world's first secure SDLC methodology [20].

After TSDM, a number of "secure SDLC methodologies" were published. The most widely discussed of these is Microsoft's Trustworthy Computing Security Development Lifecycle (SDL) [21]. Others of note include John Viega's Comprehensive Light-

weight Application Security Process [22] and Gary McGraw's Seven Touch Points [23]. More recently, the BITS Financial Services Roundtable published a Software Assurance Framework [24]. In addition, a number of efforts have been undertaken to define a maturity model specific to software assurance processes; these include the software elements of the Systems Security Engineering Capability Maturity Model (SSE-CMM) [25], the Trusted Capability Maturity Model [26], the Federal Aviation Administration safety and security extensions to integrated capability maturity models [27], OWASP Open Software Assurance Maturity Model [28], and the Cigital/Fortify Building Security In Maturity Model [29].

The majority of SDLC security enhancements involve secure coding (also referred to as secure programming), the goal of which is to prevent avoidable code-level vulnerabilities, and security code review and software security testing, the goal of which is to detect design- and implementation-level vulnerabilities not avoided earlier in the SDLC. Secure coding requires inclusions of certain logic such as input validation of all parameters and explicit security-aware exception handling, avoidance of coding constructs and program calls associated with security vulnerabilities (e.g., printf in C and C++), use of type-safe and taintable programming languages, compilers that impose bounds checking, and "safe" libraries. Techniques for secure coding have been extensively documented in books, papers, and Web sites on the topic since the beginning of this century, and the Carnegie Mellon University Software Engineering Institute (CMU SEI) Computer Emergency Response Team's Secure Coding Initiative began publishing secure coding standards for C/C++ and Java in 2008 [30].

Other SDLC security enhancements have focused on protecting development artifacts both pre- and post-deployment. This includes secure software configuration management (SCM), with supporting secure SCM systems, and application of cryptographic integrity mechanisms to software executables prior to distribution, to name a few.

Subverted SDLC Processes and Malicious Logic

Outside of DoD, the primary motivation behind defining security-enhanced SDLC processes has been preventing avoidable but non-malicious vulnerabilities in software. But intentional subversion of software is a more potentially devastating problem. The shortcomings of SDLC processes for building DoD software, whether in the U.S. or offshore, and their exploitability to subvert or sabotage that software, have been repeatedly documented by the General Accountability Office [31].

Information on subversions by intentional malicious logic inclusions involving DoD or intelligence community software or developers is, unsurprisingly, virtually always classified. In other organizations, it also remains highly sensitive, for obvious reasons that if the SDLC vulnerabilities exploited and methods used to do so were widely known, they would provide other rogue developers (both inside and outside of software teams) with tried-and-true methods to copy. Because of this secrecy, it is difficult to provide examples of actual malicious code subversions. The fact that there is so much concern over the possibility is thought by many to prove the fact that such subversions have, in fact, occurred…and often. But coming up with unclassified examples is well-nigh impossible.

One of the most persistent examples has the dubious distinction of never having been authoritatively corroborated by any of the alleged participants. But it continues to stand as an "Emperor's New Clothes" type of object lesson, so it's worth mentioning here. The story goes that in 1982 a software time bomb was planted by agents of the U.S. Central Intelligence Agency in the software of a Canadian natural gas pipeline controller product. This subversion was performed in anticipation of that product falling into the hands of Soviet agents. The goal was to use the subverted software to sabotage the Trans Siberian gas pipeline (on which the controller was expected to be installed) in a manner so spectacular that it not only destroyed the pipeline, but also lead the Soviets to mistrust all the other sensitive Western technologies they had obtained through their industrial espionage program over the previous several years [32]. Less spectacular malware subversions in the private sector have led to prison terms for perpetrators such as Michael Don Skillern and Jeffrey Howard Gibson [33].

Given such examples (and, one suspects, many more in the classified literature), it is not surprising that prevention of subversion via malicious code has been at least as potent a driver for DoD's software assurance initiatives (and, more recently, its software supply chain risk management efforts) as avoiding software vulnerabilities. In 2007, NSA undertook a project to define guidelines focused specifically on adapting the SDLC to eliminate opportunities for pre-deployment malicious inclusions in software [34].

Non-functional Security Analysis and Testing of Software

Until the late 1980s, with the exception of code with very high confidence requirements (cryptographic code, multilevel secure trusted computing base code, etc.), security testing of software meant testing the functional correctness of software-implemented network-, system- and application-level security controls (e.g., authentication, access control, data encryption). If the software belonged to a system that handled classified data, some amount of penetration testing would be performed as part of system certification, focused on attempts to escalate privileges and inappropriately leak or steal sensitive data. Even the security analyses required for attaining higher levels of assurance under the Trusted Computer Security Evaluation Criteria and the Common Criteria focused on security function correctness and information flow vulnerabilities. To this day, the Common Criteria requires no security analysis to find exploitable code-level vulnerabilities or malicious logic.

One exception has been the expansion of fault tolerance, or resilience, testing to observation of executing code's behavior under the stressful conditions associated not only with unintentional faults but with intentional attempts to exploit software errors or induce failures (in the software itself, or the execution environment or infrastructure components on which it depends). Starting in the 1990s, researchers at University of Wisconsin at Madison took the lead in this kind of stress testing when they began a 20-year investigation into use of fuzzing as a means of testing software's ability to withstand denial of service attacks that targeted its weaknesses and exploited its flaws [35].

The 1990s also brought a growing awareness of software-level vulnerabilities in Web applications and other Web-facing

software. Publication of the OWASP Top 10 persuaded many software developers and buyers that they needed a cost-effective way to verify that their software could keep attackers out while still providing legitimate users with a conduit to the Web. The means they focused on were security code reviews (via static analysis to find known undesirable patterns in source code) and vulnerability scans (mainly of COTS software). Unfortunately, neither technique has proven very useful for detecting byzantine security faults or embedded malicious logic [36].

Software security testing techniques and tools over the past decade have vastly improved in terms of increased automation, improved accuracy with regard to minimizing "false positives" and "false negatives," and standardization and interoperability of outputs via efforts such as MITRE's Making Security Measurable and the future promise of software assurance ecosystems [37].

But while individual testing techniques and (semi)integrated software security testing toolsets have evolved quickly in sophistication and accuracy since the early 1990s [38], methodologies for software security testing are still rudimentary. There is still no software security counterpart of the network security integrated "situational awareness" view. Nor does there appear to be much research to conceive a "wholistic" strategy for choosing exactly the right combination of complementary techniques and tools to achieve maximally deep and comprehensive software security analysis and test coverage that remain flexible enough to adapt to the particular software technologies and program architecture of the test subject, are usable by testing teams of varying skills and knowledge, and feasible given varying available amounts of time and budget. Lack of such a strategic testing methodology means that anything more than automated vulnerability scanning remains too time consuming and costly for all but the most "critical" and "high confidence" software…the very software that, because it is considered critical or high confidence, is the most likely to have been engineered with caution under controlled conditions, and is therefore in less need of extensive security testing.

Software Intellectual Property: Piracy, Theft, and Tampering

From the 1980s onward the single greatest "security" concern of software vendors has been the protection of their intellectual property (IP). DoD too is concerned with protecting software IP, though for different reasons.

Vendors' main concern has been piracy—the unauthorized copying and distribution of licensed software. DoD, on the other hand, is most concerned about adversaries gaining access to the IP inherent in source code of their critical software, either via reverse engineering from binaries or direct source code theft, as a step towards producing tampered, malicious versions, or studying its operation and vulnerabilities to better target or counter the systems in which it is used (e.g., weapon systems), or to obtain code on which to base comparable capabilities for their own use (in essence, piracy).

Piracy is a major concern to vendors because of the revenue loss it represents. In the 1980s, dozens of vendors rushed out hardware "dongles" for mandatory co-installation on computers on which their software was installed. The dongles ensured that the software could run only on the system for which it was licensed, and to which dongle was attached. This meant the code would not operate if copied to another system. The problem was that enterprise users had a legitimate need for backup copies of software as part of continuity of operations planning. And like any other small item, dongles were easy to misplace. So in the face of customer complaints, by the mid-1990s, most vendors had abandoned the devices in favor of digital rights management controls that accomplished essentially the same protections, and is still used by many software vendors today [39]. Over the past decade, the software industry has launched numerous anti-piracy initiatives and campaigns, individually and via their industry trade associations [40].

Protection against executable software reverse engineering led DoD, in December 2001, to establish its Software Protection Initiative (SPI). SPI develops and deploys intellectual property protections within national security system software to prevent post-deployment reverse engineering and reconnaissance, misuse, and abuse by adversaries [41]. Since its inception, the SPI has sponsored much of the significant research and development of technologies for software IP protection (e.g., anti-reverse engineering), software integrity protection (e.g., tamper-proofing), and software anti-counterfeiting.

Preventing source code theft is a problem for both vendors and government software projects. It requires both secure configuration management and effective cybersecurity protections for the computing and networking infrastructure relied on by software teams. Google discovered this to its great consternation in 2009, when Internet-based intruders stole the source code of the password management system used in most Google Web services, including Gmail. The method by which the intruders got access to the code reads like Web Application Insecurity 101: a Google China employee clicked on a link in a Microsoft Messenger message that redirected him to a malicious Web site. From there, the intruders accessed and took control of his computer, and a few short hops later, found and took control of the software repository in which the development team at Google headquarters stored the password management system code [42].

Doing Something About It: Software Assurance Initiatives and Public-Private Partnerships

In 1998 Microsoft, the world's largest software vendor, could no longer keep up with the exponential increase in reported vulnerabilities in its operating system and Web products. The company set up an internal security task force to investigate the vulnerabilities' root causes, then following the task force's recommendations, established product line security initiatives and "pushes" from 1999-2004 that ultimately coalesced into the Microsoft Trustworthy Software Development program. Two significant artifacts of the program were mandated company-wide and widely published for adoption by third-party suppliers to Microsoft (and anyone else who cared to adopt of them)—the "STRIDE/DREAD" threat modeling methodology and the SDL methodology. While other software vendors also adopted software assurance measures in the same timeframe (e.g., by the early 2000s Oracle Corporation had committed to a fairly rigorous software assurance regime), few of the others were as forthcoming or influential as Microsoft.

Seeing the world's leading software vendor change its *modus operandi* in so public a manner was an important factor in

increasing software buyers' awareness of the need for more secure software products. The OWASP Top 10 was another. Soon, organizations were rushing to discover whether their Web applications harbored any of the Top 10—and to demand that their software vendors do the same. This engendered a new industry of semi-automated tools for static security analysis of source code, and automated scanners for finding vulnerabilities in (mainly Web) application executables.

In the mid-2000s, consortia software vendors (often led by Microsoft), software security tool vendors, and corporate software users seemed to spring up every few months, including the Web Application Security Consortium in 2004, the Secure Software Forum and Application Security Industry Consortium in 2005, and SAFECode in 2007. 2007 also saw Concurrent Technologies Corp. announce the short-lived Software Assurance Consortium.

The financial services sector has also been active in its pursuit and promotion of software assurance in the context of payment and banking application security. The Visa USA Cardholder Information Security Program Payment Application Best Practices expanded and evolved into the Payment Card Industry Security Standards Council's Application Data Security Standard, now a de facto standard across the financial services industry worldwide. In the U.S., the BITS Financial Services Roundtable has undertaken a Software Security and Patch Management Initiative and Product Certification Program and produced a Software Security and Patch Management Toolkit and Software Assurance Framework for use by its members and the broader financial services community.

In the public sector, the Defense Information Systems Agency (DISA) may have been the first since the SDIO to take on the challenge of identifying and promoting methods, techniques, and supporting tools for secure software development. The three-year Application Security Project began in 2002 as a means of reducing the likelihood of OWASP Top 10 vulnerabilities in DoD Web technology-based application systems. The project's broad agenda included (1) producing developer guidance based on recognized full-SDLC best practices for secure application development; (2) assembling a portable, automated application security testing toolkit and supporting methodology with which it could offer an application vulnerability assessment service to DoD software programs; (3) defining a "reference set" of security requirements for DoD developers to leverage in their application specifications. By the end of 2004, however, DISA shifted its focus away from attempts to proactively improve the processes by which DoD software was built to reactively assessing the security of DoD software. This shift was reflected in the move of the Project to DISA's Field Security Operation, which reinterpreted the content of the Project's deliverables into a single Application Security and Development Security Technical Implementation Guide (STIG) [43] and supporting checklist. It was left up to software project managers to figure out how to ensure their teams developed software that could pass the STIG checks.

Elsewhere in DoD, security of mission critical software and risks posed by the increasing offshoring of that software were driving new initiatives. In December 1999, the Defense Science Board (DSB) suggested that the Assistant Secretary of Defense (ASD) for Command, Control, Communications, and Intelligence "develop and promulgate an Essential System Software Assurance Program" [44]. It took the ASD for Networks and Information Integration (NII) nearly four years to do just that: In June 2003, the DoD Software Assurance Initiative undertook to establish methods for evaluating and measuring assurance risks associated with commercial software, including accurate detection of the software's pedigree and provenance. In 2004, ASD(NII) joined with the Office of the Under Secretary of Defense for Acquisition, Technology and Logistics to form a Software Assurance Tiger Team for strategizing how DoD and broader Federal government would reduce its exposure to software assurance risks. The Tiger Team enlisted industry partners via the National Defense Industrial Association (NDIA), Aerospace Industries Association, Government Electronics and Information Technology Association, and Object Management Group.

The Software Assurance Initiative soon reached broad consensus on the impracticality of relying on pedigree and provenance to justify confidence in acquired software. This triggered a shift in their philosophy: all commercial software was to be considered potentially vulnerable and malicious, and engineering techniques had to be adopted to render DoD systems resilient against its destructive effects. Thus, the Initiative recast itself as the DoD System Assurance Program and, with the assistance of NDIA, developed *Engineering for System Assurance* [45] (1st edition, 2006; 2nd edition, 2008), which was expanded and adopted as a NATO engineering standard in 2010 [46].

In 2005, NSA established its Center for Assured Software (CAS) as the focal point for software assurance issues in the defense intelligence community (and in broader DoD). CAS collaborates closely with the DHS/DoD/NIST co-sponsored Software Assurance working groups and fora. CAS also influences, and in some case leads, development of DoD software assurance-related standards and policy, research, and evaluation processes, and strives to push the state of the art in software analysis tools and assessment methods. In 2009, the CAS undertook an Assurance Development Processes strategic initiative to establish trustworthy best-practice-based software development processes across DoD and the intelligence community. For several years, NSA also ran a Code Assessment Methodology Project to evaluate the security of source code to be used in high-assurance, critical DoD systems.

More recently, DoD's software assurance concerns have turned to the problems of securing the software supply chain, as a component of the larger Comprehensive National Cybersecurity Initiative (CNCI) ICT Supply Chain Risk Management (SCRM) Initiative 11, which is described—together with broader DoD and other Federal government ICT SCRM activities and programs—in the DoD Information Assurance Technology Analysis Center (IATAC) 2009 state of the art report on ICT SCRM [47].

In 2003, in parallel with DoD's efforts, DHS was assigned responsibility for responding to the *National Strategy to Secure Cyberspace*'s call for establishment of a national program to "reduce and remediate software vulnerabilities" and for facilitating "a national public-private effort to promulgate best practices and methodologies that promote integrity, security, and reliability in software code development, including processes and procedures that diminish the possibilities of erroneous code, malicious code, or trap doors" being introduced into code under

development. These responsibilities led, a year later, to the DHS Software Assurance Program. Coordinated with and complementing the efforts of DoD and the NIST Software Assurance Metrics and Tools Evaluation program (largely funded by DHS), the main thrusts of the DHS Program have been to publish secure SDLC information and guidance, promote security in software practitioner education and training, software assurance professional certification, and standardization of software assurance-related taxonomies, tool outputs, and metrics, as well as general awareness-raising. As with DoD, the Program also more recently shifted its focus to address security risks in the commercial and open source software supply chains.

Taking on the DSB's challenge that computer science academic curricula were "inadequate in terms of stressing practices for quality and security, or inculcating developers with a defensive mindset" [48], DHS made software assurance education and training one of its key thrusts from its inception. In 2006 it published a software assurance "common body of knowledge" for use in developing university curricula and courseware [49]. By 2010, the Program could boast of the Institute of Electrical and Electronics Engineers Computer Society's recognition of the Master of Software Assurance Reference Curriculum collaboratively developed by researchers and educators at several universities under DHS sponsorship [50]. IATAC's 2007 state of the report, *Software Security Assurance* [51] lists numerous examples of universities with dedicated graduate-level teaching of software assurance, advanced degrees in software assurance-related disciplines, software security research projects and labs, as well as professional training vendors with secure software development offerings, and emerging (now established) professional certifications for developers and project managers in the discipline of software security assurance and secure programming—most notably the Certified Software Security Lifecycle Professional administered by the International Information Systems Security Certification Consortium and the SANS Software Security Institute's Secure Programming Skills Assessment and Certified Application Security Professional certification. While the education/training and certification landscape described in the IATAC report has shifted somewhat in the subsequent six years, it remains generally representative.

Unlike DoD System Assurance's limited public partnerships, DHS's outreach is literally global, encompassing U.S. federal, state, and local and allied government users and producers of software, software and software security tool vendors, and academia [52]. The Program's main outreach mechanisms are its semi-annual Software Assurance Forums and more-frequent working group meetings (co-sponsored with DoD and NIST). The driving philosophy behind DHS efforts is that a general move towards more secure software worldwide will benefit federal government and DHS-protected infrastructure sectors in particular. DoD benefits from DHS's more global approach through active co-sponsorship of and participation in DHS-spearheaded endeavors. The efforts of DoD and DHS have also inspired comparable undertakings by allied governments. For example, in 2011, the United Kingdom established its own Trustworthy Software Initiative in response to 2010's *National Security Strategy of Cybersecurity* [53], which identified lack of secure, dependable, resilient software as a critical risk to the UK's cybersecurity posture. And NATO published *Engineering Methods and Tools for Software Safety and Security* in 2009 [54].

A number of significant software assurance research initiatives are ongoing in the U.S. and abroad, especially in Europe, including the Network of Excellence on Engineering Secure Future Internet Software Services and Systems project sponsored by the European Commission's Seventh Framework Programme for Research (FP7) [53].

Conclusion

It has often been claimed that we already know how to build secure software. If this is true, why don't we just do it? But no matter how much lip service they pay to wanting software that has fewer vulnerabilities and is less susceptible to malicious inclusions, most suppliers and consumers still make their how-to-build and what-to-buy decisions based on cost, or on a "value proposition" that boils down to how fast innovations can be turned into available product, and cost. Few buyers are willing to wait longer and pay more so software can undergo the disciplined engineering needed to assure its trustworthiness and dependability. Nor are they willing to forego desirable innovations just because the level of security risk they pose is not (and possibly cannot) be known. Nor will suppliers willingly invest in and enforce software assurance measures that few customers demand.

But the continuing, and indeed growing, reliance on COTS and open source software means that, to succeed in the long term, software assurance efforts cannot remain limited to the small subset of software deemed "high consequence" or "trusted." It is impossible to predict which of today's "general purpose" software products will end up in tomorrow's high consequence, trusted systems, just as it was impossible to predict in 1988 that Microsoft Excel and Internet Explorer (to name two examples) would, in spite of their persistent, myriad vulnerabilities and susceptibility to malicious insertions, emerge as vital components of mission critical national security systems.

Instead of attempting to second guess which software needs to be trustworthy and dependable, software assurance should be applied systematically and comprehensively to all software. For this to happen, future software assurance efforts need to finally take on the elephant in the room: the need to change the psychology of the suppliers and consumers. Awareness campaigns and polite suggestions of software assurance content for post-graduate academic software engineering curricula attempt to persuade and reeducate developers and consumers long after their bad habits have been formed, and are thus far too little far too late. A "software assurance mentality" needs to be inculcated during the very earliest years in which future developers and users encounter software and begin to understand how it works and its value to them. Software (and machine logic) are nearly universal today, and are only going to become more integral to every aspect of daily life by the time the next generation of developers and users reach working age. For this reason, tomorrow's developers and users need to be taught from early childhood that threats to the security of software and logic-bearing devices are threats to their own personal privacy, health, safety, financial security, and, ultimately, happiness.

Other Attempts to Characterize the Software Assurance Landscape

This article has only touched on highlights of the last 25 years of software security assurance initiatives and trends. There have been several earlier efforts to depict the software assurance landscape in varying levels of detail. Interested readers are encouraged to take a look at:

- Goertzel, Karen Mercedes, et al. *Software Security Assurance* (see reference 51).
- Davis, Noopur. *Secure Software Development Life Cycle Processes: A Technology Scouting Report*. Technical Note CMU/SEI-2005-TN-024, December 2005 <http://www.sei.cmu.edu/reports/05tn024.pdf>
- Jayaram, K.R., and Aditya P. Mathur. *Software Engineering for Secure Software–State of the Art: A Survey*. Purdue University Center for Education and Research in Information Assurance and Security and Software Engineering Research Center Technical Report 2005-67, 19 September 2005 <http://www.cerias.purdue.edu/assets/pdf/bibtex_archive/2005-67.pdf>
- DHS. *Software Assurance Landscape*. Preliminary Draft, 28 August 2006 <https://www.owasp.org/images/6/6c/Software_Assurance_Landscape_-_Preliminary_Draft_1.doc>
- Graff, Mark D. *Secure Coding: The State of the Practice*, 2001 <http://markgraff.com/mg_writings/SC_2001_public.pdf>
- Essafi, Mehrez, et al. *Towards a Comprehensive View of Secure Software Engineering. Proceedings of the International Conference on Emerging Security Information, Systems, and Technologies (SecureWare 2007)*, Valencia, Spain, 14-20 October 2007 <http://www.researchgate.net/publication/4292729_Towards_a_Comprehensive_View_of_Secure_Software_Engineering/file/79e4150bb0ce96e522.pdf>

ABOUT THE AUTHOR

Karen Mercedes Goertzel, CISSP, is an expert in application security and software and hardware assurance, the insider threat to information systems, assured information sharing, emerging cybersecurity technologies, and ICT supply chain risk management. She has performed in-depth research and analysis and policy and guidance development for customers in the U.S. financial and ICT industries, DoD, the intelligence community, Department of State, NIST, IRS, and other civilian government agencies in the U.S., the UK, NATO, Australia, and Canada.

7710 Random Run Lane–Suite 103
Falls Church, VA 22042-7769
703-698-7454
goertzel_karen@bah.com

ADDITIONAL SUGGESTED READING

(Contact the author to request a comprehensive list of published books on software security assurance.)

1. DHS/US-CERT. Build Security In Web portal. <https://buildsecurityin.us-cert.gov>
2. David A. Wheeler's Personal Home Page. <http://www.dwheeler.com>
3. CMU SEI. *Insider Threats in the Software Development Lifecycle*, 23 February 2011 <http://www.cert.org/archive/pdf/sepg500.pdf>
4. Fedchak, Elaine, et al. *Software Project Management for Software Assurance*. Data and Analysis Center for Software (DACS) Report Number 347617 (Rome, NY: DACS, 30 September 2007) <http://www.thedacs.com/get_pdf/DACS-347617.pdf>
5. *International Journal of Secure Software Engineering* <http://www.igi-global.com/journal/international-journal-secure-software-engineering/1159>

REFERENCES

1. Karger, Paul A., and Roger R. Schell, *Multics Security Evaluation: Vulnerability Analysis*. U.S. Air Force Electronics Systems Division Report ESD-TR-74-193, Volume II, June 1974. <http://seclab.cs.ucdavis.edu/projects/history/papers/karg74.pdf>
2. Abbot, Robert P., *The RISOS [Research into Secure Operating Systems] Project: Security Analysis and Enhancements of Computer Operating Systems*. National Bureau of Standards Interagency Report Number NBSIR 76-1041, 1976.
3. The Karger/Schell Multics analysis in 1974 had also identified buffer and stack overflow errors and lack of adequate input validation as the system's most significant, proven-exploitable vulnerabilities. More than a decade later, in 1988, a buffer overflow in the Berkeley Unix finger daemon was exploited by Robert Tappan Morris to launch the world's first Internet worm. 25 years on, the 2011 Veracode *State of Software Security Report* found that exploitable buffer overflows and memory management issues were still the most prevalent vulnerabilities in commercial software. *Plus ça change, plus c'est la même chose.*
4. PITAC. *Report to the President on Cyber Security: A Crisis of Prioritization*, February 2005. <http://www.nitrd.gov/pitac/reports/20050301_cybersecurity/cybersecurity.pdf>
5. Landwehr, Carl E., et al. "A Taxonomy of Computer Program Security Flaws, with Examples". Naval Research Laboratory Center for High Assurance Computer Systems technical report NRL/FR/5542-93-9591, 19 November 1993. <http://chacs.nrl.navy.mil/publications/CHACS/1994/1994landwehr-acmcs.pdf;>
6. Du, Wenliang, and Aditya Mathur, "Categorization of Software Errors That Led to Security Breaches". *Proceedings of the National Information Systems Security Conference*, 1998. <http://www.cis.syr.edu/~wedu/Research/paper/nissc98.ps;>
7. Open Web Application Security Project (OWASP). "Top Ten Most Critical Web Application Security Vulnerabilities", 2002 (and revised several times since then). <https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project>
8. The Common Weakness Enumeration (CWE) <http://cwe.mitre.org/> is a standard dictionary of "root causes" at the specification, design, and implementation levels of exploitable software security vulnerabilities, such as those listed (along with system and network vulnerabilities) in MITRE's Common Vulnerabilities and Exposures (CVE; <http://cve.mitre.org>). In 2012 CWE was adopted by the International Telecommunications Union (ITU) as *Recommendation X.1524 : Common weakness enumeration.* <http://www.itu.int/rec/T-REC-X.1524-201203-I/en>
9. Schneier, Bruce. "Attack Trees". Dr. Dobb's Journal, December 1999. <http://www.schneier.com/paper-attacktrees-ddj-ft.html>
10. Moore, Andrew P., et al. "Attack Modeling for Information Security and Survivability". CMU/SEI-2001-TN-001, March 2001. <http://www.cert.org/archive/pdf/01tn001.pdf>. The culmination of attack pattern-definition attempts is MITRE's Common Attack Pattern Enumeration and Classification (see <http://capec.mitre.org/>), a standardized list of the most common techniques for exploiting the vulnerabilities that can result from CWEs.
11. Howard, Michael, and David LeBlanc. Chapter 2, "Security Design by Threat Modeling". *Writing Secure Code*. (Redmond, WA: Microsoft Press, 2002).
12. Gonsalves, Antone. "The Shadowy World of Selling Software Bugs, and How It Makes Us all Less Safe". *readwrite hack*, 4 October 2012. <http://readwrite.com/2012/10/04/the-shadowy-world-of-selling-software-bugs-and-how-it-makes-us-all-less-safe>
13. Rashid, Fahmida. "Researchers Hijack Printer Using Malicious Firmware Update". *eWeek*, 29 November 2011.

14. Borg, Scott. "Securing the Supply Chain for Electronic Equipment: A Strategy and Framework". Whitepaper developed for the White House by the Internet Security Alliance, November 2008.
15. Hadži, Ilija, et al. "FPGA Viruses". University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-99-06, January 1999. <http://www.cis.upenn.edu/~jms/papers/fpgavirus.pdf>
16. Torres, Lionel, et al. "Security and FPGA: Analysis and Trends". Deliverable SP1 for the ICT for Emerging Regions Project of France's Agence Nationale de Recherche, 31 January 2007. < http://www.lirmm.fr/~w3mic/ANR/PDF/D1.pdf >
17. King, Samuel T., et al. "Designing and Implementing Malicious Hardware". *Proceedings of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats*. San Francisco, CA, 15 April 2008. < http://static.usenix.org/events/leet08/tech/full_papers/king/king.pdf http://www.whitehouse.gov/files/documents/cyber/ISA - Securing the Supply Chain for Electronic Equipment.pdf>
18. DARPA Press Release. "New DARPA Program Seeks to Reveal Backdoors and Other Hidden Malicious Functionality in Commercial IT Devices". 30 November 2012.
19. Later published under the title "Software Aspects of Strategic Defense Systems". Communications of the ACM [Association for Computing Machinery], Vol. 28 No. 12, December 1985. <http://klabs.org/richcontent/software_content/papers/parnas_acm_85.pdf>
20. Watson, John, and Edward Amoroso. "A Trusted Software Development Methodology". *Proceedings of the 13th National Computer Security Conference*, Volume II. Washington, D.C., 1990. TSDM was later renamed Trusted Software Methodology.
21. Lipner, Steven B. "The Trustworthy Computing Security Development Lifecycle". Proceedings of the 20th Annual Computer Security Applications Conference, Tucson, AZ, 6-10 December 2004.
22. Viega, John. "Security in the Software Development Lifecycle". IBM developerWorks, 15 October 2004. <http://www.ibm.com/developerworks/rational/library/content/RationalEdge/oct04/viega/viega.pdf>
23. McGraw, Gary. "The Seven Touch Points of Secure Software". Dr. Dobb's Journal, 1 September 2005. <http://www.drdobbs.com/the-7-touchpoints-of-secure-software/184415391>
24. BITS Financial Services Roundtable. *Software Assurance Framework*. January 2012. <http://www.bits.org/publications/security/BITSSoftwareAssurance0112.pdf>
25. SSE-CMM Project. *Systems Security Engineering Capability Maturity Model*, Version 1.0. 21 October 1996.
26. DHS. Section D.5. *Security in the Software Lifecycle*, Draft Version 1.2, August 2006. <http://www.cert.org/books/secureswe/SecuritySL.pdf> Note that this book was significantly revised and republished by DHS in 2008 as Enhancing the Development Lifecycle to Produce Secure Software.
27. Ibrahim, Linda, et al. "Safety and Security Extensions for Integrated Capability Maturity Models", September 2004. <http://www.faa.gov/about/office_org/headquarters_offices/aio/library/media/SafetyandSecurityExt-FINAL-web.pdf>
28. Version 1 of OpenSAMM was released in 2008. See OWASP's OpenSAMM Web pages and the OpenSAMM Web site. <http://www.opensamm.org> <https://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model>
29. Version 1 of the BSIMM was published in 2009. The current version is Version 4. See the BSIMM Web site. <http://bsimm.com/>
30. CMU SEI Secure Coding Standards Web page. <http://www.cert.org/secure-coding/scstandards.html>
31. Government Accountability Office (GAO). *DoD Information Security: Serious Weaknesses Continue to Place Defense Operations at Risk*. GAO/AIMD-99-107, August 1999. <http://www.gao.gov/products/AIMD-99-107>. GAO. *Defense Acquisitions: Knowledge of Software Suppliers Needed to Manage Risk*. GAO-04-678, May 2004. <http://www.gao.gov/new.items/d04678.pdf>. GAO. *Offshoring of Services: an Overview of the Issues*. GAO-06-5, November 2005. <http://www.gao.gov/new.items/d065.pdf>
32. Murdoch, Steven. "Destructive Activism: The Double-Edged Sword of Digital Tactics". Joyce, Mary, editor, *Digital Activism Decoded: The New Mechanics of Change* (New York, NY: International Debate Education Association, 2010). Also Weiss, Gus W., "Duping the Soviets: The Farewell Dossier". *Studies in Intelligence*, Volume 29 Number 5, 1996. <https://www.cia.gov/library/center-for-the-study-of-intelligence/kent-csi/vol39no5/pdf/v39i5a14p.pdf>
33. Gabrielson, Bruce, Karen Mercedes Goertzel, et al. Appendix E, "Real World Insider Abuse Cases". *The Insider Threat to Information Systems* [Unclassified, For Official Use Only, U.S. Government and Contractors Only]. (Herndon, VA: IATAC, 10 October 2008).
34. NSA. *Guidance for Addressing Malicious Code Risk*, 10 September 2007. <http://www.nsa.gov/ia/_files/Guidance_For_Addressing_Malicious_Code_Risk.pdf>
35. Miller, Barton P., et al. "An Empirical Study of the Reliability of UNIX Utilities". *Communications of the ACM*, Volume 33 Number 12, December 1990. <ftp://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz.pdf>; University of Wisconsin-Madison Fuzz Testing of Application Reliability Web page. <http://pages.cs.wisc.edu/~bart/fuzz/fuzz.html>
36. The annual Underhanded C Code Contest is designed to illustrate how ineffective code reviews are for finding malicious inclusions in code. Contest submissions must be malicious while maintaining "plausible deniability", i.e., their malicious logic must by definition not be detectable through static analysis. See <http://underhanded.xcott.com/>
37. Such as KDM Analytics' Software Assurance Ecosystem, <http://www.kdmanalytics.com/swa/ecosystem.php>, and Object Management Group's Software Assurance Ecosystem, < http://sysa.omg.org/docs/SwA Ecosystem/Assurance Ecosystem.ppt>.
38. Information on the wide range of tools available can be found on the DHS Build Security In Web portal, <https://buildsecurityin.us-cert.gov/>, the NIST Software Assurance Metrics and Tools Evaluation portal, <http://samate.nist.gov/>, and in Goertzel, Karen Mercedes, et al. *Vulnerability Assessment* (Herndon, VA: IATAC, May 2011), <http://iac.dtic.mil/iatac/download/vulnerability_assessment.pdf>. Outsourcing to expert service providers such as Veracode and Aspect Security may be an alternative to doing one's own security analyses and tests.
39. Belovich, Steve, Ph.D. "IT Security History and Architecture–Part 5 of 6". Infosec Island, 24 August 2010. <http://infosecisland.com/blogview/6931-IT-Security-History-and-Architecture-Part-5-of-6.html>
40. The most noteworthy are those of the Business Software Alliance, Software and Information Industry Association, Entertainment and Leisure Software Publishers Association, Information Technology and Innovation Foundation, Federation Against Software Theft, Entertainment Software Association, Content Delivery and Storage Association, and the Association for Copyright of Computer Software. See: Section 4.7.27, "Anti-Piracy Initiatives". Goertzel, Karen Mercedes, et al. *Security Risk Management in the Off-the-Shelf (OTS) Information and Communications Technology (ICT) Supply Chain: a State-of-the-Art Report* [U.S. Government and Contractors Only] (Herndon, VA: IATAC, 17 August 2010).
41. Hughes, Jeff, and Martin R. Stytz. "Advancing Software Security: the Software Protection Initiative". <http://www.preemptive.com/documentation/SPI_software_Protection_Initative.pdf>
42. Markoff, John. "Cyberattack on Google Said to Hit Password System". *The New York Times*, 19 April 2010. <http://www.nytimes.com/2010/04/20/technology/20google.html>
43. DISA. *Application Security and Development STIG*. <http://iase.disa.mil/stigs/app_security/app_sec/app_sec.html>
44. Defense Science Board (DSB). *Final Report of the Defense Science Board Task Force on Globalization and Security* (especially Annex IV), December 1999. <http://www.acq.osd.mil/dsb/reports/globalization.pdf>
45. DoD System Assurance Working Group and NDIA. *Engineering for System Assurance*, Version 1.0, 2008. <http://www.acq.osd.mil/sse/docs/SA-Guidebook-v1-Oct2008.pdf>
46. NATO Standard AEP-67. *Engineering for System Assurance in NATO Programmes*, First Edition, 4 February 2010 <http://nsa.nato.int/nsa/zpublic/ap/aep-67(1)e.pdf>
47. Op. cit. Goertzel, et al. *Security Risk Management in the OTS ICT Supply Chain*.
48. DSB. *Final Report of the Task Force on Mission Impact of Foreign Influence on DoD Software*, September 2007. <http://www.cyber.st.dhs.gov/docs/Defense Science Board Task Force–Report on Mission Impact of Foreign Influence on DoD Software (2007).pdf>
49. DHS Software Assurance Common Body of Knowledge/Principles Organization Web page. <https://buildsecurityin.us-cert.gov/bsi/dhs/927-BSI.html>
50. IEEE Computer Society. "Computer Society Recognizes Master of Software Assurance Curriculum". Press release dated 8 December 2010. <http://www.computer.org/portal/web/pressroom/20101213MSWA> and CMU SEI Software Assurance Curriculum Web page. <http://www.cert.org/mswa/> Also DHS Software Assurance Curriculum Project Web page. <https://buildsecurityin.us-cert.gov/bsi/1165-BSI.html>
51. Goertzel, Karen Mercedes, et al. *Software Security Assurance* (Herndon, VA: IATAC, 31 July 2007). <http://iac.dtic.mil/csiac/download/security.pdf>. Note that in April/May 2013, DoD tasked the Defense Technical Information Center's Cyber Security Information Analysis Center (the successor to IATAC) to produce an update of this report, which is expected to be published in late 2013.
52. A number of other defense and civil agencies are also committed to improving software security assurance. For example, Part 10 Chapter 8 of the *Internal Revenue Service Manual* includes a section entitled "Secure Application Development". <http://www.irs.gov/irm/part10/irm_10-008-006.html>
53. Her Majesty's Government Cabinet Office. *Cyber Security Strategy*, 2010 (revised November 2011). <http://www.cabinetoffice.gov.uk/resource-library/cyber-security-strategy>
54. Broy, Manfred, et al., editors. *Engineering Methods and Tools for Software Safety and Security*. NATO Science for Peace and Security Series D: Information and Communication Security, Volume 22 (Amsterdam, The Netherlands: IOS Press, 2009).
55. Network of Excellence on Engineering Secure Future Internet Software Services and Systems Web page. <http://www.nessos-project.eu/>