April 2019

SOFTWARE SECURITY IS NATIONAL SECURITY

WHY THE U.S. MUST REPLACE IRRESPONSIBLE PRACTICES WITH A CULTURE OF INSTITUTIONALIZED SECURITY

Authored By:

Drew Spaniel, Lead Researcher, ICIT
Rob Roy, ICIT Fellow & Public Sector CTO, Micro Focus Government Solutions

ICIT Institute for Critical Infrastructure Technology
The Cybersecurity Think Tank

MICRO FOCUS
Government Solutions

Software Security is National Security

Why the U.S. Must Replace Irresponsible Practices with a Culture of Institutionalized Security
April 2019

The authors would like to thank the following individuals for their advisement and expertise around software security. The views expressed in this paper are those of the authors, not that of the experts listed below.

- Parham Eftekhari, Executive Director, ICIT
- Dr. Ron Ross, Fellow, National Institute of Standards and Technology (NIST)
- Michael Aisenberg, ICIT Fellow & Principal Cyber Policy Counsel, MITRE Center for National Security
- Jerry Davis, ICIT Fellow and Vice President and Global Chief Security Officer, Lam Research
- David Summitt, Fellow, ICIT Fellow and CISO, Moffitt Cancer Center
- Stan Wisseman, Chief Security Strategist, Micro Focus

Contents

Introduction: A Lack of Software Security Is a National Threat

Software runs the world. PCs, mobile devices, the cloud, the Internet of Things, operational technology, vehicles, appliances, utilities, and nearly every modern device runs on code. Consequently, the importance of resilient software is of utmost importance to a safe and secure world.

It is vital to national security that stakeholders, their partner organizations, and their supply chains understand and subscribe to the notion that "it takes a village" to secure software development in today's complex and interconnected global economy. As Michael Aisenberg, ICIT Fellow & Principal Cyber Policy Counsel, MITRE Center for National Security, explains, "Users of the exploding array of software systems and software-enabled devices across the 'connected' ecosystems we have come to define as the world of IoT/IIOT/CPS/SCADA have learned of the inherent trade-offs embedded in the many benefits of these globally wired systems. Unfortunately, one significant trade-off involves undisclosed, undiscovered, and unaddressed operating risks arising from the very fact of robust connectivity. These risks are particularly hazardous when visited on insecure critical infrastructure systems, where the continuity of operations is a dependency of other systems in a fabric, like the electric power grid; in a commercial operation, like chemical manufacturing; or in a public service system, like a municipal or regional water system. The risks of interrupted or improper operation may not only incur costs or delays, they may pose material threats to public health and safety or to national security."

Adversaries will attack the weakest link in the security ecosystem. They will compromise a widely used open source application, inject malware onto a public-facing server, poison a downstream supply chain update with malicious code, social engineer an un-cyber-hygienic employee, or do anything else imaginable to gain access to sensitive systems and data. It behooves stakeholders of every category to work together to minimize the threat landscape exploitable by digital adversaries. Aisenberg continues, "Emerging security solutions, in system access control, in perimeter defense, in threat identification and attribution, and in risk prophylaxis, such as the SCRM requirement of software bill of materials (sBoM), are important security responses that may offer a reduction in the risks accompanying our growing reliance on these connected systems as part of the critical infrastructure fabric. As with other software and software-enabled systems, these established and emerging security protections for connected infrastructures are only useful if used; thus, the essential consideration is the tried-and-true fundamental of system security – 'build security in, don't bolt it on.'"

The need for more secure software development is not a new concept. In 2006, at least 7,000 vulnerabilities that could either cause systems to malfunction or be exploited by malicious actors were discovered across systems operating in every market and critical infrastructure sector [1]. That number grew to 17,147 exploitable vulnerabilities in 2016 and 19,954 in 2017. Approximately 17 percent of the vulnerabilities were rated as "highly critical" in both years [2]. Most software developers defend themselves against claims of indifference to software security by pointing to statistics that show patches were available on the day of disclosure for 86 percent of the vulnerabilities in 2017 and zero-day vulnerabilities decreased by 40 percent

between 2016 and 2017, down to just 14 out of 19,954 known vulnerabilities [2] [3]. While it is positive to see organizations immediately patching flaws as they are discovered, the vast majority of organizations are still not doing their due diligence to ensure that exploitable vulnerabilities are not present in the software that they release.

While software security is becoming a higher priority, for many businesses it remains an afterthought, with 33% of applications never tested for security vulnerabilities [4]. Almost 80% of applications contain at least one critical or high vulnerability. An estimated 84% of security breaches exploit vulnerabilities at the application layer. Consequently, for the time between release and patch, adversaries can leverage the weaknesses in the code to compromise sensitive systems. In his presentation "Rethinking Cybersecurity from the Inside Out," NIST Fellow Dr. Ron Ross reminds his audience that for an application with 50 million lines of code (LOC), assuming the empirically found rate of 4.9 flaws per 1,000 LOC, the application will have between 2,400 to 12,200 potential security vulnerabilities present at release.

According to Dr. Ross, "As long as 'secure-by-design' is not a priority within the software and IT industry, we will never be able to reach our full potential for innovation as a society or as a nation. It's like designing the next-generation autonomous automobile and failing to install brakes. Fundamentals matter, especially in the world of cyber-physical systems and IoT." A more sustainable, secure, and economically responsible approach would be for software vendors to deliver products where a large portion of those vulnerabilities were remediated before the applications were ever released.

While this paper will explore systemic problems in the software security landscape and offer recommendations on how to improve application security, it is important for readers to understand the context behind this conversation. Powerful nation-state adversaries, notably China and Russia, are aggressively exploiting vulnerabilities in the code running our public and private sectors to steal intelligence, PII, and IP by the troves. This data is being used to undermine our nation and our allies and threaten democracies and free economies around the globe.

When reading this paper, we encourage you to consider not only the technical outcomes of these ideas, but the national and global impacts of inaction if we continue down the path of insecure and poorly coded software. As evidenced by the recent wave of media reports on the vulnerabilities in our defense, federal, and private sector critical infrastructures, secure software development must become a top priority to ensure the security and resiliency of our nation. Information is power, and once data is lost to an adversary, it cannot easily be reclaimed or retracted, especially if it pertains to a human rather than a system. With enough effort, one can change their Social Security number, but they cannot change their birth date, interests, or other similar characteristics. Incidents at the U.S. Office of Personnel Management (OPM), Equifax, Yahoo, Apple, Twitter, and other public and private sector organizations exemplify how each and every breach has the potential to impact tens of millions of Americans and jeopardize national security. For instance, the psychographic and demographic data of 21.5 million federal employees and contractors stolen from OPM in 2015 could shape the espionage

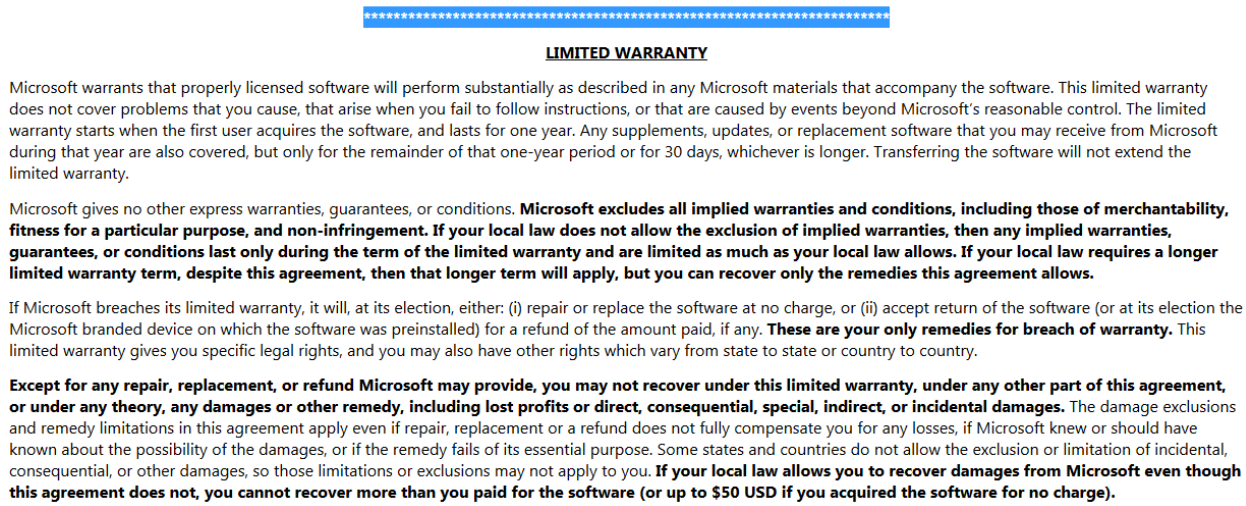
community and strategic diplomatic decisions for decades to come. Meanwhile, the 145 million Americans impacted by the Equifax breach may be the perpetual victims of cybercriminal harassment for the foreseeable future.

Systemic Problems in the Software Development Landscape

Current Practices Rely on "Crash Test Dummy" Consumers

Consumers and organizations ranging from small companies to major corporations to the federal government are unwittingly and oxymoronically paying for the expensive "privilege" of acting as crash test dummies for software manufacturers, often at the cost of profits, reputation, privacy, and other intangible assets that they cannot recover easily. Vulnerabilities in software applications are not rare or new. Have you ever been asked to provide crash data to Google or to allow Microsoft to collect information on your system? Did you know that on many operating systems, you resign your right to hold companies liable for flaws in their software the moment you first boot up the system? Consider the limited warranty for Windows 10.

Figure 1: Microsoft's Windows 10 Limited Warranty



LIMITED WARRANTY

Microsoft warrants that properly licensed software will perform substantially as described in any Microsoft materials that accompany the software. This limited warranty does not cover problems that you cause, that arise when you fail to follow instructions, or that are caused by events beyond Microsoft's reasonable control. The limited warranty starts when the first user acquires the software, and lasts for one year. Any supplements, updates, or replacement software that you may receive from Microsoft during that year are also covered, but only for the remainder of that one-year period or for 30 days, whichever is longer. Transferring the software will not extend the limited warranty.

Microsoft gives no other express warranties, guarantees, or conditions. **Microsoft excludes all implied warranties and conditions, including those of merchantability, fitness for a particular purpose, and non-infringement. If your local law does not allow the exclusion of implied warranties, then any implied warranties, guarantees, or conditions last only during the term of the limited warranty and are limited as much as your local law allows. If your local law requires a longer limited warranty term, despite this agreement, then that longer term will apply, but you can recover only the remedies this agreement allows.**

If Microsoft breaches its limited warranty, it will, at its election, either: (i) repair or replace the software at no charge, or (ii) accept return of the software (or at its election the Microsoft branded device on which the software was preinstalled) for a refund of the amount paid, if any. **These are your only remedies for breach of warranty.** This limited warranty gives you specific legal rights, and you may also have other rights which vary from state to state or country to country.

Except for any repair, replacement, or refund Microsoft may provide, you may not recover under this limited warranty, under any other part of this agreement, or under any theory, any damages or other remedy, including lost profits or direct, consequential, special, indirect, or incidental damages. The damage exclusions and remedy limitations in this agreement apply even if repair, replacement or a refund does not fully compensate you for any losses, if Microsoft knew or should have known about the possibility of the damages, or if the remedy fails of its essential purpose. Some states and countries do not allow the exclusion or limitation of incidental, consequential, or other damages, so those limitations or exclusions may not apply to you. **If your local law allows you to recover damages from Microsoft even though this agreement does not, you cannot recover more than you paid for the software (or up to $50 USD if you acquired the software for no charge).**

Figure 1 captures the limited warranty section of Windows 10 Terms of Service Agreement. Note how users are not protected from the impacts of cyber incidents and how Microsoft does not bear liability for vulnerabilities in its product.

If Microsoft releases a flawed software that is exploited by cyber-attackers, based on its end user licensing agreement (EULA), the most consumers can do is either get a refund or attempt to enter into "binding individual arbitration" in front of a neutral arbiter, not in front of a judge or jury. Class-action lawsuits and more serious recourse are not an option. The chance for a

refund or replacement product seems insignificant recompense for the potential loss of valuable personal information or intellectual property.
Following is but one of these EULAs for the Apple Mojave operating system. Users are required to accept this 560-page document the first time they use the system.

Figure 2: Apple Mojave End-User License Agreement

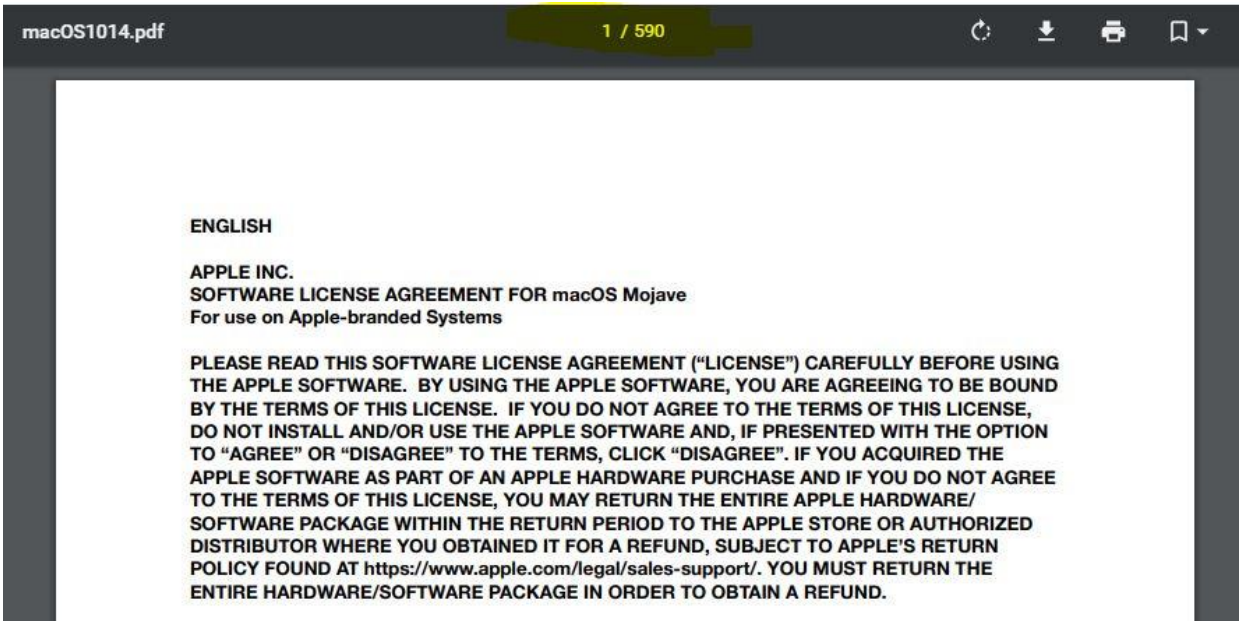


Figure 2 features the license agreement for macOS Mojave. The terms of service are 560 pages of legal jargon intentionally written to absolve Apple of legal liability by shifting it to the users and to dissuade users from having notice and choice concerning their data or from making an informed purchasing decision.

You will find the company indemnification statement within the first 10 pages:

Figure 3: Apple Mojave Indemnification Clause

B. YOU EXPRESSLY ACKNOWLEDGE AND AGREE THAT, TO THE EXTENT PERMITTED BY APPLICABLE LAW, USE OF THE APPLE SOFTWARE AND ANY SERVICES PERFORMED BY OR ACCESSED THROUGH THE APPLE SOFTWARE IS AT YOUR SOLE RISK AND THAT THE ENTIRE RISK AS TO SATISFACTORY QUALITY, PERFORMANCE, ACCURACY AND EFFORT IS WITH YOU.

C. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE APPLE SOFTWARE AND SERVICES ARE PROVIDED "AS IS" AND "AS AVAILABLE", WITH ALL FAULTS AND WITHOUT WARRANTY OF ANY KIND, AND APPLE AND APPLE'S LICENSORS (COLLECTIVELY REFERRED TO AS "APPLE" FOR THE PURPOSES OF SECTIONS 7 AND 8) HEREBY DISCLAIM ALL WARRANTIES AND CONDITIONS WITH RESPECT TO THE APPLE SOFTWARE AND SERVICES, EITHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES AND/OR CONDITIONS OF MERCHANTABILITY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR PURPOSE, ACCURACY, QUIET ENJOYMENT, AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS.

Figure 3 exhibits the terminology that companies leverage to shift risk to consumers while absolving themselves of any potential liability for vulnerable software.

Insecure software development normalized because it was found to be profitable and relatively sustainable, since consumers tend to not care, to be ill-informed, or be easily distracted or quelled using intentionally misleading narratives such as "added features" or "weekly updates." For instance, many sites and applications rely on intentionally long and confusing "terms and conditions" or privacy policies to deter users from understanding or caring how their information is collected, monetized, or leveraged. If Americans were to spend 10 minutes reading the policies for each site they use during a year, it would result in an estimated opportunity cost of at least $781 billion annually [5]. While most users have the ability to understand "terms and conditions" with enough effort, only a miniscule percentage of users of any given product have the knowledge, ability, or time to test whether their applications and systems have exploitable vulnerabilities.

Figure 4: Apple Mojave Limited Liability

8. Limitation of Liability. TO THE EXTENT NOT PROHIBITED BY APPLICABLE LAW, IN NO EVENT SHALL APPLE, ITS AFFILIATES, AGENTS, PRINCIPALS, OR LICENSORS BE LIABLE FOR PERSONAL INJURY, OR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, CORRUPTION OR LOSS OF DATA, FAILURE TO TRANSMIT OR RECEIVE ANY DATA OR INFORMATION, BUSINESS INTERRUPTION OR ANY OTHER COMMERCIAL DAMAGES OR LOSSES, ARISING OUT OF OR RELATED TO YOUR USE OR INABILITY TO USE THE APPLE SOFTWARE OR SERVICES OR ANY THIRD PARTY SOFTWARE OR APPLICATIONS IN CONJUNCTION WITH THE APPLE SOFTWARE OR SERVICES, HOWEVER CAUSED, REGARDLESS OF THE THEORY OF LIABILITY (CONTRACT, TORT OR OTHERWISE) AND EVEN IF APPLE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR PERSONAL INJURY, OR OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS LIMITATION MAY NOT APPLY TO YOU. In no event shall Apple's total liability to you for all damages (other than as may be required by applicable law in cases involving personal injury) exceed the amount of fifty dollars ($50.00). The foregoing limitations will apply even if the above stated remedy fails of its essential purpose.

Figure 4 captures Apple Mojave's limitation of liability. Note how all risk of loss or consequence is shifted to the user and how, even if found liable, the most Apple can be held accountable for is $50.

Software engineering practices rely on programmers producing large programs at a breakneck pace. Even though the developers know the software will likely contain hundreds or thousands of flaws, the "mistakes" are released into the global market with the knowledge that the faulty code, which might be used in aircraft, defense systems, healthcare networks, financial systems, energy distribution channels, telecommunications, or any other critical system, might fail as a result of internal error or external exploitation. Nevertheless, software developers release the code because they have engineered a global culture of minimized culpability for their firms. In addition, legal practices enable them to both leverage users as monitors and crash test dummies to find flaws in the code and to shift risk and responsibility onto the same users. For their part, consumers ranging from individuals to massive corporations remain unaware of the transfer of responsibility and risk, and many forever remain blissfully unaware that they never received the secure or reliable product that they paid for. Software vulnerabilities are typically reported by consumers to the manufacturer through reporting mechanisms, metadata, crash report clients, or after a breach or other cybersecurity incident has occurred.

Patching Is Retroactive and Dangerous

Patching is a retroactive, responsive, and often defensive practice that ensures that software security is determined by malicious digital threat actors, instead of by developers or their clients. While some patches are released after a user discovers a flaw, many are released long after attackers have exploited that flaw.

Though it may seem counterintuitive, in many instances, organizations have to pay service fees to patch and update their software. In some cases, these costs are aggregated with license fees. As a result of budget constraints, many organizations patch their systems irregularly, according to system priority, or in an ad hoc manner. Without a consistent patching methodology across all systems, many vulnerabilities remain unpatched for prolonged periods, every second of which can be leveraged by an adversary intent on exploiting the vulnerability, infiltrating the network, and exfiltrating sensitive information. While organizations should be held accountable for their mismanagement of systems with regard to regular patching, updating, and modernization, it is long past time that software vendors be held accountable for negligently promoting a culture that relies on "deploy now, patch later" and that forces firms to expend their limited resources to patch products that are all too often faulty upon release.

Open Source May Not Be "Free" or Secure

Open source solutions come from either zero-cost or low-cost models and often include software frameworks used to build custom applications. The former is produced by non-commercial active user communities who have limited accountability and provide minimal and sporadic support. The prolonged and delayed update and release cycle of zero-cost products increases the amount of time that a system is vulnerable to publicly known exploits. Further, adversaries can more easily inject malicious code into the product releases. Detecting changes in the binaries of the zero-cost code is trivial, but correlating those alterations with adversarial activity from potentially millions of others proves more difficult. Among the 1,300 respondents in the 2016 Future of Open Source Survey, 90% said low-cost open source solutions improve efficiency, interoperability, and innovation, and 65% had increased their use of open source

software over the last year. In addition, 65% contributed to open source projects, mainly in order to fix bugs or add functionality to a project, and 67% actively encouraged developers to engage in and contribute to open source projects. Half of the companies surveyed lacked formal policies governing open source adoption; almost half lacked formal processes to track open source code; and nearly a third lacked processes for identifying, tracking, and remediating known open source vulnerabilities, however [6]. While open source solutions may be a viable option for some organizational needs, the inability to verify the integrity of the code and the resources required to maintain and secure the code personally may serve as a significant barrier to the adoption of open source solutions in secure settings.

Open source solutions are public by definition, and as a result, knowledge of discovered vulnerabilities tends to spread fast; as a free resource, however, the known vulnerabilities may linger longer than in commercial code. If open source coding frameworks are deployed, developers should account for the potential risks and trade-offs that they are assuming by relying on the frameworks. Tools exist to discover and exploit vulnerabilities in open source code. Developers should conduct penetration testing on their open source solutions or hire firms who are experts in evaluating open source code to do so.

Cultural Norms Need to Change

Software manufacturers expend so little effort and resources ensuring that applications are secure because doing so enables them to both maximize profits (at significant additional cost to consumers) and to capture secondary profits patching and otherwise fixing the products that should not have been faulty in the first place. Buyers rarely consider whether software is free of defects. Instead, their focus is on functionality, cost, features, and other more measurable attributes. Little do they realize that lower costs may correlate with less penetration testing and security inclusion during development, and additional features correlate with increased complexity and a greater propensity for exploitable vulnerabilities [1].

Buyers are not completely unaware of software vendors' gambit; in many instances, they have no choice but to assume the risk. An estimated 77% of buyers do not trust their software to behave as expected. Many buyers recognize, however, that their options may be limited to selecting the most trustworthy offering or not adopting any software. There may not be a software option that incorporates security by design throughout the development lifecycle or that dedicates the resources to minimize software vulnerabilities, because the competitive software market relies on rush-to-market behavior, "minimized cost, maximize profits" ideologies, and incessant product churn [1].

As previously stated, over three-fourths of consumers do not trust software vendors, and one crippling side effect of that perception is that ubiquitous fear and uncertainty reduces the price that users are willing to pay for any product. As a result, even more competitive pressure is exerted on vendors, and the incentive to test software extensively and reduce vulnerabilities decreases drastically. This pervasive perception lowers profit margins and renders security testing and secure design decisions as cost prohibitive. The cycle culminates in a self-

perpetuating death spiral, in which users demand increasing functionality at lower costs while manufacturers add more features at decreased production costs. If security is included at all in the development lifecycle, it is funded as an afterthought using the last dregs of the budget [1]. One of the critical problems of this paradigm is that both buyers and manufacturers are behaving rationally according to modern economics. Numerous market incentives, ranging from outsourcing to (unstable) innovation to "rush-to-market" behavior, aggregate against secure software development. The individual self-interests of developers are allowed to affect the public good adversely, because there is no oversight or regulation sufficient to counter the aforementioned market incentives and promote mandatory secure development as an economically rational decision.

Adverse Cultural Norms Facilitate Adversarial Activity

A Cultural Renaissance in Software Security is Necessary

Software development is governed by market churn, a false drive for "innovation," and mistakenly incentivized negative behaviors that culminate in products that contain minimal layered security and numerous exploitable vulnerabilities. This culturally institutionalized insecure product design has created a fertile ground for obfuscated malicious campaigns. The Theory of Broken Windows postulates that smaller elements of disorder invite larger agents of chaos to intercede in the market. In cybersecurity, this applies in two distinct ways. Microsoft, Apple, Google, and other software giants largely govern industry standards and practices because secure design is not federally mandated or regulated. As a result, vendors of every size emulate their practices. Firms who attempt to incorporate security responsibly into each stage of their product lifecycle are often unable to compete with the firms who rush to market with the "deploy now, patch later" ideology.

On the adversarial side, script kiddies often conduct penetration testing on exposed systems and applications and sell discovered vulnerabilities. As applications popularize, as zero-day exploits are sold, or as rumors of exploitability travel through Deep Web markets and forums, more sophisticated adversaries begin to target vulnerable applications. Eventually, a client is targeted or breached by a threat, the vulnerability is discovered, and the cycle continues. The caveat to these microcosms, which software developers tend to either miss or willfully ignore, is that once a software or type of software has become a popular enough target for a category of threat actor, then more actors will begin to target that type of software (even from different manufacturers) if they believe it is lucrative or potentially exploitable. Entire communities of attackers grow and escalate on the mistakes of negligent software developers.

The "Deploy Now, Patch Later" ideology only feeds the adversarial cycle, incentivizes new entrants, and systematically lowers the security of organizations, agencies, and individuals everywhere. To change this pervasive culture, vendors, their clients, and legislators should work together to improve the cybersecurity posture of organizations radically by agreeing to leverage existing guidelines and frameworks, sharing best practices, and balancing demands for functionality with the realities of deploying truly trusted and assured software.

Guidelines and Frameworks

Jerry Davis, an ICIT Fellow and the Vice President and Global Chief Security Officer at Lam Research, advises, "Software has become the underlying pervasive substrate that enables all matters in life today. Software drives cars, flies aircraft, moves machinery, facilitates global commerce, and links the simultaneous and instantaneous communication and collaboration of billions of people and things that are thousands to billions of miles apart. Secure software development must become 'just how it is done.' Secure software development must be implemented not as a feature or an option, but rather as an organic component within the quality management framework." Numerous secure software development frameworks have been developed and promoted by respected organizations such as NIST, but their adoption is dependent on their pervasiveness in the software development community and on their desire for organizations to adopt best practice frameworks proactively. Below are some existing guidelines and frameworks which responsible software developers should be using as part of their development lifecycle:

NIST SP 800-37

NIST Special Publication 800-37, titled "Guide for Applying the Risk Management Framework to Federal Information Systems," emphasizes defining the correct set of security controls and implementing them in a robust continuous monitoring process. More importantly, it stresses the inclusion of comprehensive security from an information system's initial design phase through implementation and daily operations. SP 800-37 states, "When security requirements are considered as an integral subset of other information system requirements, the resulting system has fewer weaknesses and deficiencies, and therefore, fewer vulnerabilities that can be exploited in the future" [7].

SP 800-37 is centered on NIST's Risk Management Framework (RMF), which outlines six steps federal agencies must take to secure their information systems:

1. Security categorization, based on impact analysis
2. Security control selection
3. Security control implementation
4. Security control assessment
5. Information system authorization
6. Security control monitoring

Overall, the guidance aims to align the management of information system-related security risks with an organization's business objectives and overall risk strategy, to integrate security controls into the organization's enterprise architecture and system development lifecycle, to support continuous security monitoring and transparency of security and risk-related information, and to increase the security of information and information systems within the federal government through the implementation of appropriate risk mitigation strategies [7].

NIST SP 800-53

According to NIST Fellow Ron Ross, we should require systems and software engineers to approach development with the same level of attention to construction and security as we require of civil engineers, who must adhere to the principles of physics and engineering to build reliable and safe structures. In an interview with Tripwire, he stated, "We need to have the same confidence in the trustworthiness of our IT products and systems that we have in the bridges we drive across or the airplanes we fly in" [8]. NIST Special Publication 800-53 provides guidelines for selecting the security controls for information systems that support federal agencies. Recent revisions of the guidelines emphasize building security into products from the beginning and monitoring the systems continuously, rather than relying on periodic audits.

Additionally, the latest revision also includes new security controls that address mobile and cloud computing, insider threats, and supply chain security. The guidelines apply to all components of an information system that process, store, or transmit federal information. SP 800-53 recommends selecting an initial set of baseline security controls, then customizing the baseline controls to fit the needs and risk appetite of the organization, and finally supplementing the controls based on risk assessments. Controls include the management, operational, and technical safeguards and countermeasures that protect the confidentiality, integrity, and availability of the system and its information. SP 800-53 couples a "Build It Right" strategy with a variety of "Continuous Monitoring" security controls to give organizations the near real-time information that is essential for making ongoing risk-based decisions affecting critical business functions [7].

NIST SP 800-64

To be most effective, information security must be integrated into the SDLC from system inception. NIST SP 800-64 rev2 complements the Risk Management Framework by having a comprehensive approach of managing risk and appropriate level of security based on the levels of risk. It helps in providing guidance on integrating security functionality and assurance into the SDLC. It describes key security roles and responsibilities in SDLC and thereafter directs the relationship between Information Security and SDLC. Some key roles and responsibilities include:

Role	Responsibility
Authorizing Official (AO)	Executive responsible for acquiring/operating of an information system at an acceptable level of risk
Chief Information Officer (CIO)	Responsible for planning, budgeting, investment, performance, and acquisitions
Configuration Management (CM) Manager	Responsible for streamlining change management processes and controls changes that may affect the system's security posture
Information System Security Officer (ISSO)	Responsible for ensuring the security of the system throughout its lifecycle

Privacy Officer	Responsible for ensuring the privacy of the procured services or system
Program Manager	Manages the functional system requirements during SDLC and responsible for all business and program handling during the lifecycle process
QA/Test Director	Responsible for reviewing system specifications and determines test needs; works with program managers to plan activities leading up to field testing; also responsible for system testing, evaluation, and execution of test plans as stated in specifications
Chief Information Security Officer (CISO)	Responsible for imposing policies of integrating security into SDLC
Software Developer	Responsible for secure coding; implements controls and other CM issues
System Architect	Responsible for designing and maintaining the system architecture; also ensures quality of specifications, documentation, and other key areas
System Owner	Responsible for the procurement, development, integration, modification, operation, and maintenance of an information system

NIST 800-64 describes the secure development lifecycle in five phases:

- **Initiation Phase:** The enterprise establishes the project goals and system requirements and documentation.
- **Development/Acquisition Phase:** The system is designed, purchased, programmed, developed, or otherwise constructed.
- **Implementation/Assessment Phase:** Developers review the system design by installing the system security features and testing its functionality before placing the system into operation, as described in the specifications.
- **Operations/Maintenance Phase:** The system is operating and continuously monitored to ensure the pre-established requirements are incorporated, and hardware or software components are added or replaced.
- **Disposal Phase:** An orderly termination of the system is completed by preserving all the vital information of the system according to the record management regulations, so that it can be reactivated in the future if needed. It also ensures that the data is deleted,

erased, or written over as necessary, and that hardware and software are archived or disposed of as directed by the relevant authorities [9].

Motor Industry Software Reliability Association (MISRA) Guidelines

MISRA coding standards have been adopted by industries developing safety-critical embedded software, including automotive, telecom, aerospace, defense, and medical. MISRA sets secure software development guidelines for the C and C++ programming languages. It aims to facilitate code safety, security, portability, and reliability in the context of embedded systems, specifically those systems programmed in ISO C/C90/C99. Guidelines are classified as *Mandatory, Required* or *Advisory*. Mandatory guidelines should always be complied with. Required guidelines should be complied with, unless subject to an acceptable deviation. Advisory guidelines are considered good practice, but compliance is less formal [10].

CERT Secure Coding Initiative

Developed by the Software Engineering Institute (SEI) at Carnegie Mellon University, the CERT Secure Coding Initiative works with software developers and software development organizations to reduce vulnerabilities resulting from coding errors before they are deployed. Key steps include identifying common programming errors that lead to software vulnerabilities, publishing secure coding standards, educating software developers with the goal of advancing best practices in secure coding, and offering tools to detect and mitigate code vulnerabilities.

SAMATE

Software Assurance Metrics and Tool Evaluation (SAMATE) is sponsored by the U.S. Department of Homeland Security (DHS) National Cybersecurity Division and NIST. SAMATE focuses on software assurance and on ensuring that security is included throughout software development. The objective of Part 3, Technology (Tools and Requirements) is the identification, enhancement, and development of software assurance tools [11] [12].

OASIS SARIF

The purpose of the OASIS Static Analysis Results Interchange Format (SARIF) is to set a standard output for static analysis tools. The goals of the format are:

- Comprehensively capture the range of data produced by commonly used static analysis tools
- Be a useful format for analysis tools to emit directly, and also an effective interchange format into which the output of any analysis tool can be converted
- Be suitable for use in a variety of scenarios related to analysis result management and be extensible for use in new scenarios
- Reduce the cost and complexity of aggregating the results of various analysis tools into common workflows
- Capture information that is useful for assessing a project's compliance with corporate policy or conformance to certification standards
- Adopt a widely used serialization format that can be parsed by readily available tools

- Represent analysis results for all kinds of programming artifacts, including source code and object code

OASIS SARIF was designed for developers who use static analysis to measure, assess, and track the quality of code, and for other relevant stakeholders [13].

Common Weakness Enumeration (CWE™)

The Common Weakness Enumeration initiative was developed by MITRE to provide a unified, measurable set of software weaknesses enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems. According to MITRE, "CWE™ is a community-developed list of common software security weaknesses. It serves as a common language, a measuring stick for software security tools, and as a baseline for weakness identification, mitigation, and prevention efforts" [14].

Security Technical Implementation Guides (STIGs)

The Security Technical Implementation Guides (STIGs), developed by the Defense Information Systems Agency (DISA), are the configuration standards for DOD IA and IA-enabled devices/systems. The STIGs contain technical guidance to "lock down" information systems/software that might otherwise be vulnerable to a malicious computer attack [15]. The Application Security and Development STIG in particular enables developers to more readily identify vulnerabilities and prioritize mitigation and remediation strategies to meet Authority to Operate thresholds [16].

OWASP

The Open Web Application Security Project (OWASP) is a 501(c)(3) focused on improving the security of application software. Its mission is to make application security visible, so that people and organizations can make informed decisions about true application security risks. Principles described in the OWASP Secure Coding Guidelines include:

1. Input Validation
2. Output Encoding
3. Authentication and Password Management (includes secure handling of credentials by external services/scripts)
4. Session Management
5. Access Control
6. Cryptographic Practices
7. Error Handling and Logging
8. Data Protection
9. Communication Security
10. System Configuration
11. Database Security
12. File Management
13. Memory Management
14. General Coding Practices

Though OWASP specifically refers to web applications, many, if not all, of its secure coding principles can be applied to non-web applications as well [16][17][18]. It is important to recognize that the top 10 vulnerabilities identified by OWASP have barely changed over the past decade. In other words, despite guidance on the principles of software security, developers have failed to improve application security to the point that the leading risks have diminished. Adversaries continue to rely on the same exploits and strategies that worked a decade ago. If an organization just focused on the known vulnerabilities in one of the top categories, they could drastically reduce the risk in their applications. According to Stan Wisseman, Chief Security Strategist at Micro Focus "Rather than boil the ocean, an organization could take a measured approach to security by focusing on one category like SQL injection (No. 1 on the OWASP list) and mandate their developers to remove EVERY instance of it. This is learned behavior, because software engineering courses do NOT teach secure coding and novices introduce vulnerabilities like this as if they were spelling errors. It should be unacceptable, and these vulnerabilities are trivial to find and fix with the right tools."

Correcting the Course of Negligent Software Development

A cybersecurity renaissance, particularly with the goal of changing the culture of software development, requires a layered approach led by stakeholders from public and private sectors dedicated to enacting meaningful change. Below are some recommended areas of focus:

Security by Design

System transparency spans a spectrum that ranges from fully open source to black-box proprietary. Open source solutions, while attractive because of their low-cost or no-cost pricing models, are susceptible to implanted malware, obfuscated backdoors, system inoperability, delayed patching, and limited service options (depending on the source of the code). While some open source developers take precautions such as maintaining their code, tracking alterations, and conducting penetration testing, many open source solutions place the entire burden of security on the clients who chose to adopt the code. Meanwhile, black-box proprietary systems are applications in which vendors refuse or legally deter clients and third parties from evaluating the code. While many proprietary systems are kept closed to protect trade secrets and mitigate adversarial efforts, preventing evaluation of the code forces clients to accept vendors' offerings on blind trust, while often assuming most or all of the risk and liability. The lack of transparency can damage reputations, communication, and trust, because without the ability to evaluate code or to seek third-party evaluation, a client must simply trust that the vendor's application is secure. The lack of verification of the integrity of the code could prove devastating to the client in the immediate aftermath of a major incident. Further, the reputation of the vendor could be irreparably harmed if the incident becomes widely publicized or is particularly scandalous.

Trust in cybersecurity solutions is zero-sum; either a vendor can demonstrate that its application was developed securely with minimal vulnerabilities to the best of its knowledge and capabilities, or the vendor is unable or unwilling to meet the modern needs of clients who

must secure their systems against hyper-evolving cyber-threats. Clients should not accept blind reassurances of security from open source or proprietary vendors. Trust in software needs to be assured. The Department of Defense National Defense Authorization Act Joint Federated Assurance Center offers one example of how the government is enabling software and hardware assurance. The DOD NDAA JFAC "develops, maintains, and offers hardware and software vulnerability detection, analysis, and remediation capabilities through a federation of internal, coordinated organizations and facilities from across the Military Departments, Defense Agencies, and other DOD organizations" [17].

Security is fluid. Systems that are deemed secure one day may be proven vulnerable the next. Security is not about faith, it is dependent on iterative verification. Clients have the power to select solutions that were developed with security throughout the product lifecycle according to secure coding practices and frameworks, such as NIST 800-160 or MITRE's Deliver Uncompromised. Moreover, clients have the buying power to purchase solutions whose security has been tested by industry-appointed third parties. Vendors can best serve their clients by developing applications that incorporate security at each stage of development and that can be verified as secure through repeated and regular assessment by industry-appointed trusted third parties, both before the code is released and throughout the solution lifecycle.

Evaluate the Quality of Code

When it comes to software security, the problem is not that software developers are not debugging their products, though in many cases, they are not conducting enough penetration testing. As Dave Summitt, ICIT Fellow and Moffitt Cancer Center CISO, explains, "There are a few reasons why security is not a priority for software developers. First, developers are not trained in the area of vulnerabilities and threats. In many cases, it is perceived to be an added complexity to the process and that complexity is not understood by the developer. Typically they have one goal in mind – make the code work. Those who are formally trained in information security concepts are unfortunately often trained in academic programs that do not teach security of code. Second, there has been no push from the cyber community to incorporate security code reviews within development protocols. Companies or developers that are securing software development are typically those that have either had an issue or been made aware of an issue. The bottom line is that there does not seem to be an industrywide understanding for a good ROI for incorporating security. When a breach or a cyber incident occurs, rarely is it brought up that if proper coding had been performed, the incident may not have occurred. As a result, developers and/or the developer's company are not held accountable for issues in the code."

The underlying issue is that because development does not incorporate secure design as a fundamental practice at each stage of the product lifecycle, many vulnerabilities remain latent until long after mass release and popularization. Software vendors spend, at most, 35 percent of their production time testing their code for vulnerabilities and conducting other forms of debugging [1]. As products become more popular, they become a more lucrative target for hackers intent on exploiting any discoverable vulnerability. Even as patches mitigate some

flaws, new ones are discovered, and a desperate game of "patch and mitigate" ensues until the product is phased out and replaced with the next iteration.

No software will ever be completely secure, no matter how much security is emphasized during development. We do not live in a world of ideals; instead, in this case, as in many, the perfect is the enemy of the good. Software developers recognize the former statement while eschewing the latter. They operate on the assumption that it is not productive to take every action possible to minimize latent vulnerabilities because they know their product still will not be perfect. They rely on patching because it minimizes effort on their part and it is incentivized according to the current under-regulated software market. Instead, developers could incorporate security into the lifecycle of their products according to NIST's SP 800-160 or MITRE's Deliver Uncompromised, and they could release the best product possible for the reputation of their business and the security of their clients. They could stop responding to adversarial compromises and instead release a more secure product that is more difficult for adversaries to exploit. Threat actors have finite resources. Many target vulnerable applications that are "the lowest-hanging fruit." As a result, if an application is less vulnerable, it is feasible that fewer adversaries will successfully exploit it. Fewer successful compromises will likewise correlate with less access-as-a-service and sales of targeted exploit kits to lower-level attackers.

Institutionalize Security to Reshape Software Development Culture

Critical infrastructure sectors are shaped by regulation and oversight, such as HIPAA, FISMA, FERPA, Sarbanes-Oxley, and numerous other important policies, procedures, guidelines, legislation, and rules. The software development sector remains relatively unregulated, however, despite its impact on every critical infrastructure, every organization, and nearly every aspect of daily life. Most past attempts to institutionalize mandatory security requirements were met with derision from the development community for a variety of reasons, with some exceptions, such as the DOD NDAA.

Stop Buying Into the Notion That Secure Software Development Stymies Innovation

The most vocal admonishment of security reform was the notion that security requirements would stymie innovation. While mandatory security would have some initial impact on software churn, the argument fails to align with reality when subjected to close scrutiny. Regulation and demonstrated security are features of healthcare, technology, and every other critical infrastructure sector. As the past demonstrates, unchecked advancement may not be in the best interest of the public. Healthcare could be advanced much faster under amoral conditions that did not place limits on trials and experiments; but, as a community, we recognize that boundaries are necessary to protect the public, ensure safety, preserve integrity, and promote responsible conduct. Software development is much the same. Unregulated development could be innovative, but it could equally be reckless or irresponsible. While no system is perfect, the rules and requirements we choose to implement have a profound influence over our culture, products, and safety. Many would attest that controlling innovation through boundaries yields far superior solutions that better protect and serve the communities that rely on them. Further, software development has become stagnated in a retroactive culture that relies more on the revenue from patch management models than it does on innovating and developing better products. Requiring security at each layer of development prior to release would reduce the

number of inherent vulnerabilities, decrease the number of post-release patches, and course-correct software developers back to the path of innovation.

Security Requirements Are a Necessary Barrier

Some vendors have voiced concerns that unless they were minimized to the point of triviality, security requirements would act as a barrier of entry to startups and other market entrants who have fewer resources available than market leaders. This barrier can largely be avoided if legislators and regulators develop security requirements without the influence of market leaders and other third parties lobbying with the intent to weaponize security against their competitors. Securely developing code may be resource-intensive for smaller firms, but it is a "necessary evil." Just as doctors are required to be licensed, implementing security evaluations would serve to protect national institutions and critical infrastructure clients from negligent vendors who refuse to detect, mitigate, and remediate the flaws in their code. Over time, the software development industry may become less about the rush to market and more about delivering quality solutions to paying clients. Experience has shown that while implementing security during development is an initial cost, the benefits of training and enabling all developers in the practice of secure coding pays dividends in perpetuity, as the cost of maintaining, patching, and risk management decreases significantly.

Scalable Security Requirements Are Realistic

The final myth of secure software development is that software cannot be regulated because code is too distinct and diverse to be governed by a singular set of overarching principles. The argument is rife with logical fallacy. Security frameworks can be scalable to the sector, purpose, and purview of every piece of software. Developers who argue otherwise are either shielding their unwillingness to consider security a priority of software development or are unaware of cybersecurity frameworks, automation and machine learning tools, and best practices.

Hold Stakeholders Accountable

At the time of this writing, the only major national discussions concerning the regulation of the security of software are whether Congress should regulate the use of facial recognition software, whether cryptocurrency should be regulated, whether CEOs should be liable for breaches, and legislation regarding supply chain risk [17][18][19] [18][19][20] [19][20][21] [22]. While the former two topics may merit discussion, only the latter discussions of liability and supply chain risk are directly applicable to software security. Holding CEOs responsible for breaches is bound to be controversial in some sectors, but it may be a move in the proper direction, provided that liability is applied to the right party. When a digital adversary breaches an organization, is that firm liable if the threat actor gained access through the exploitation of a vulnerability in an application? Furthermore, to what degree is a firm liable for vulnerabilities exploited in the software supply chain? The answers to these questions are outside the scope of this publication, but they could prove important to software security in particular and cybersecurity in general.

Holding software vendors and their clients accountable for cybersecurity incidents is an emerging discussion in part because of the enactment of the GDPR. Often, when liability is attributed, it is almost always placed onto the client because of language incorporated into

service level agreements by legal teams intent on shielding software developers from taking responsibility for the integrity of their products. Risk, security, and liability are ill-measured and not uniformly defined in the legal community; consequently, even when attempts are made to hold a developer responsible for negligent practices that resulted in the adversarial exploitation of a known or preventable vulnerability, it is easy for legal teams to circumvent any semblance of due process. Congress, regulatory agencies, and industry leaders can work together to properly define and standardize important terminology and expectations of responsibility internal and external to the software development community. Moreover, clients can leverage their significant buying power to demand secure practices from their vendors. Developers who are unwilling or unable to demonstrate that their products are secure to acceptable standards, by providing documentation on their development process or by providing an artifact verifying that they tested their software for vulnerabilities, will lose market share to more reliable vendors.

Conclusion

Improving software development practices must become a priority among the technology community. Malicious adversaries depend on exploitable vulnerabilities in software applications to gain a foothold in the network, to escalate privileges, to laterally compromise systems, to avoid detection, and to exfiltrate data. If software developers incorporated layered security throughout the development lifecycle of their products, then malicious adversaries would have fewer vulnerabilities to exploit, client organizations would waste less of their budget paying for system remediation and patching, and software vendors could focus more of their resources on innovating and developing higher quality products that better secure and meet the needs of their clients.

Sources

[1] Rice, D. (2007). Geekonomics - The Price of Insecure Software. [online] Amazon.com. Available at: https://www.amazon.com/Geekonomics-Real-Insecure-Software-paperback/dp/0321735978. [Accessed: 06- Jan- 2019].

[2] P. Muncaster, "New Vulnerabilities Hit All-Time High of 20,000 in 2017", *Infosecurity Magazine*, 2018. [Online]. Available: https://www.infosecurity-magazine.com/news/flexera-20000-new-software-flaws/. [Accessed: 06- Jan- 2019].

[3] "Zero-day vulnerability: What it is, and how it works", *Us.norton.com*, 2019. [Online]. Available: https://us.norton.com/internetsecurity-emerging-threats-how-do-zero-day-vulnerabilities-work-30sectech.html. [Accessed: 06- Jan- 2019].

[4] "Build Application Security into the Entire SDLC", Microfocus.com, 2018. [Online]. Available: https://www.microfocus.com/media/brochure/build_application_security_into_the_entire_sdlc_brochure.pdf. [Accessed: 07- Jan- 2019].

[5] A. McDonald and L. Cranor, "The Cost of Reading Privacy Policies", *Lorrie.cranor.org*, 2008. [Online]. Available: http://lorrie.cranor.org/pubs/readingPolicyCost-authorDraft.pdf. [Accessed: 06- Jan- 2019].

[6] Ankerholz, Amber. "2016 Future Of Open Source Survey Says Open Source Is The Modern Architecture". Linux.com | The source for Linux information. N.p., 2016. Web. 16 June 2017. https://www.linux.com/news/2016-future-open-source-survey-says-open-source-modern-architecture

[7] "NIST Compliance", *Veracode*, 2018. [Online]. Available: https://www.veracode.com/security/nist-compliance. [Accessed: 06- Jan- 2019].

[8] "NIST Guidelines to Help IT System Developers Build-In Security | The State of Security", *Tripwire*, 2014. [Online]. Available: https://www.tripwire.com/state-of-security/latest-security-news/nist-guidelines-to-help-it-system-developers-build-in-security/. [Accessed: 06- Jan- 2019].

[9] "NIST SP 800-64-Rev.2", *Computer Security Resource Center*, 2019. [Online]. Available: https://csrc.nist.gov/publications/nistpubs/800-64-rev2/sp800-64-revision2.pdf. [Accessed: 06- Jan- 2019].

[10] "MISRA Compliance: 2016", *Misra.org.uk*, 2016. [Online]. Available: https://www.misra.org.uk/LinkClick.aspx?fileticket=w_Syhpkf7xA%3D&tabid=57. [Accessed: 06- Jan- 2019].

[11] P. Black, "Samate and evaluating static analysis tools.", *https://www.researchgate.net/publication/228996566_Samate_and_evaluating_static_analysis_tools*, 2007. [Online]. Available:

https://www.researchgate.net/publication/228996566_Samate_and_evaluating_static_analysis_tools. [Accessed: 06- Jan- 2019].

[12] "SAMATE", *NIST*, 2019. [Online]. Available: https://samate.nist.gov/index.php/Introduction_to_SAMATE.html. [Accessed: 06- Jan- 2019].

[13] OASIS Static Analysis Results Interchange Format (SARIF) Technical Committee | Charter. (2019). Retrieved from https://www.oasis-open.org/committees/sarif/charter.php

[14] "CWE -Common Weakness Enumeration", *Cwe.mitre.org*, 2019. [Online]. Available: https://cwe.mitre.org/. [Accessed: 06- Jan- 2019].

[15] "STIGs Home", *Iase.disa.mil*, 2019. [Online]. Available: https://iase.disa.mil/stigs/Pages/index.aspx. [Accessed: 06- Jan- 2019].

[16] Application Security & Development. (2019). Retrieved from https://iase.disa.mil/stigs/app-security/app-security/Pages/index.aspx

[17] Joint Federated Assurance Center Charter. (2013). Retrieved from https://www.acq.osd.mil/se/docs/JFAC-Charter-020915-SansMemo.pdf

[18] "OWASP", *Owasp.org*, 2019. [Online]. Available: https://www.owasp.org/index.php/Main_Page. [Accessed: 06- Jan- 2019].

[19] N. Singer, "Microsoft Urges Congress to Regulate Use of Facial Recognition", *Nytimes.com*, 2018. [Online]. Available: https://www.nytimes.com/2018/07/13/technology/microsoft-facial-recognition.html. [Accessed: 06- Jan- 2019].

[20] A. Rossow, "Congress Goes Crypto: An Overview Of Digital Asset Regulation", *Forbes.com*, 2018. [Online]. Available: https://www.forbes.com/sites/andrewrossow/2018/07/19/congress-goes-crypto-an-overview-into-digital-asset-regulation/#2ee2495ed01c. [Accessed: 06- Jan- 2019].

[21] D. Weeks, "It's time to regulate: The U.S. must make software companies liable for breaches", *VentureBeat*, 2018. [Online]. Available: https://venturebeat.com/2018/03/24/its-time-to-regulate-the-u-s-must-make-software-companies-liable-for-breaches/. [Accessed: 06- Jan- 2019].

[22] S.3085 - 115th Congress (2017-2018): Federal Acquisition Supply Chain Security Act of 2018. (2019). Retrieved from https://www.congress.gov/bill/115th-congress/senate-bill/3085/text. [Accessed: 06- Jan-2019].