Non-Malicious Taint: Bad Hygiene is as Dangerous to the Mission as Malicious Intent

Robert A. Martin
MITRE Corporation
Bedford, MA

ABSTRACT

Success of the mission should be the focus of software and supply chain assurance activities regardless of what activity produces the risk. It does not matter if a malicious saboteur is the cause. It does not matter if it is malicious logic inserted at the factory or inserted through an update after fielding. It does not matter if it comes from an error in judgment or from a failure to understand how an attacker could exploit a software feature. Issues from bad software hygiene, like inadvertent coding flaws or weak architectural constructs are as dangerous to the mission as malicious acts. Enormous energies are put into hygiene and quality in the medical and food industries to address any source of taint. Similar energies need to be applied to software and hardware. Until both malicious and non-malicious aspects of taint can be dealt with in ways that are visible and verifiable there will be a continued lack of confidence and assurance in the delivered capabilities throughout their life-cycle.

BACKGROUND

Every piece of information and communications technology (ICT) hardware—this includes computers as well as any device that stores, processes, or transmits data—has an initially embedded software component that requires follow-on support and sustainment throughout the equipment's life-cycle.

The concept of supply chain risk management (SCRM) must be applied to both the software and hardware components within the ICT. Because of the way ICT hardware items are maintained, the supply chain for on-going sustainment support of the software is often disconnected from the support for the hardware (e.g., continued software maintenance contracts with third parties other than the original manufacturer). As a result, supply chain assurance regarding software requires a slightly unique approach within the larger world of SCRM.

Some may want to focus on just "low hanging fruit" like banning suspect products by the the country they come from or the ownership of the producer due to their focused nature and ignore more critical issues surrounding the software aspect of ICT like the exploitable vulnerabilities outlined in this article. It is a misconception that "adding" software assurance to the mix of supply chain concerns and activities will add too much complexity, thereby making SCRM even harder to perform. Some organizations and sectors are already developing standards of care and due-diligence that directly address these unintended and bad hygiene types of issues. That said, such practices for avoiding the bad hygiene issues that make software unfit for its intended purpose are not the norm across most of the industries involved in creating and supporting software-based products. Mitigating risk to the mission is a critical objective and including software assurance as a fundamental aspect of SCRM for ICT equipment is a critical component of delivering mission assurance.

During the past several decades, software-based ICT capabilities have become the basis of almost every aspect of today's cyber commerce, governance, national security, and recreation. Software-based devices are in our homes, vehicles, communications, and toys. Unfortunately software, the basis of these cyber capabilities, can be unpredictable since there are now underlying rules software has to follow as opposed to the rest of our material world which is constrained by the laws of gravity, chemistry, and physics with core factors like Planck's Constant. This is even more true given the variety and level of skills and training of those who create and evolve cyber capabilities. The result is that for the foreseeable future there will remain a need to address the types of quality and integrity problems that leave software unreliable, attackable, and brittle directly. This includes addressing the problems that allow malware and exploitable vulnerabilities to be accidentally inserted into products during development, packaging, or updates due to poor software hygiene practices.

Computer language specifications are historically vague and loosely written. (Note: ISO/IEC JTC1 SC22 issued a Technical Report [1] with guidance for selecting languages and using languages more secure and reliably.) There is often a lack of concern for resilience, robustness, and security in the variety of development tools used to build and deploy software. And there are gaps in the skills and education of those that manage, specify, create, test, and field these software-based products.

Additionally, software-based products are available to attackers who study them and then make these products do things their creators never intended. Traditionally this has led to calls for improved security functionality and more rigorous review, testing, and management. However, that approach fails to account for the core differences between the engineering of software-based products and other engineering disciplines. Those differences are detailed later in this article.

The need to address these differences has accelerated as more of the nation's critical industrial, financial, and military capabilities rely on cyber-space and the software-based products that comprise this expanding cyber world. ICT systems must be designed to withstand attacks and offer resilience through better integrity, avoidance of known weaknesses in code, architecture, and design. Additionally, ICT systems should be created with designed-in protection capabilities to address unforeseen attacks by making them intrinsically more rugged and resilient so that there are fewer ways to impact the system. This same concern has been expressed by Congress with the inclusion of a definition of "Software Assurance" in Public Law 112-239 Section 933 [2] where they directed DoD to specifically address software assurance of its systems.

DEFINING "TAINT" AND SOFTWARE ASSURANCE

While there is no concrete definition of what "taint" specifically means within the cyber realm, we would be remiss not to look to the general use of the term, as well as synonyms and antonyms. Merriam Webster [3] provides a useful point-of-departure, as shown in Table 1 below.

Table 1: Merriam Webster Dictionary Taint Information

Taint Synonyms	blemish, darken, mar, poison, spoil, stain, tarnish, touch, vitiate

Taint Antonyms	decontaminate, purify

[Note: Taint is also defined in the Universal Dictionary of the English Language.

Taint - n. trace of physical corruption or decay; degradation, imperfection; contamination, pollution. Vb. To infect with physical or moral deterioration and corruption; to render unwholesome or noxious. To become infected, corrupted, by something noxious, by decay. Taintless - adj. Without taint; uncorrupted, pure.

It is important to note that 'taint' is a state, consequently independent of intent. Taint for ICT components is expressed in terms of 'stateful' properties associated with programmable logic in the components that could have malware, exploitable weaknesses, or vulnerabilities – independent of how the components might have become tainted (e.g., through negligence, sloppy manufacturing hygiene, or malicious intent) – in development or supply chain management.]

So a “tainted” ICT product could be described as one that is blemished, marred, spoiled, or in need of being purified and/or decontaminated. Within the Department of Defense (DoD) community one must also make use of the definition of “software assurance” provided by the United States Congress in Public Law 113-239 Section 933. Therein, software assurance is defined to mean “the level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software, throughout the life cycle”. [Note – this is the same definition used by the Committee on National Security Systems (CNSS) 4001 Glossary.] Taken together, software assurance would also ensure that the software was not tainted, since tainted software would offer an attacker the opportunity to make the software function in an unintended manner.

Similarly, within the software vulnerability community, vulnerability has been defined [4] as an occurrence of a weakness (or multiple weaknesses) within software, in which the weakness can be used by a party to cause the software to modify or access unintended data, interrupt proper execution, or perform incorrect actions that were not specifically granted to the party who uses the weakness.

This paper proposes that for any product making use of software, “taint” would generally be considered any weakness/issue that impacts its ability to function as intended or that otherwise introduces vulnerabilities or malware.

CYBER HYGIENE – AN EVOLUTION OF SYSTEMS ENGINEERING NEEDED

Some offer that we could address our collective cyber software hygiene and assurance problems if only information assurance, cybersecurity, and supply chain assurance could "fit" into the general systems engineering practice. However, as described in the following sections, there are three things that are different about software-based systems when engineering them for today’s cyber world that go beyond what most consider good hygiene and systems engineering. Consequently, past norms of “good practice” related to software that remain in use in government and industry may be inadequate given today’s pervasive use of software-based products in our new cyber world.

Fragile In Unexpected Ways and Deployed in an Unknown Manner

Software based systems often have additional features, interfaces, and functionality due to the use of 3rd party libraries, general purpose commercial and open source applications, and the multiplicity of features in system libraries and system calls. Moreover, developers often make risk decisions for which they are not held accountable (such as disabling compiler warnings about security flaws in code). As witnessed by the myriad of patches we are regularly called to address due to the exploitable vulnerabilities they address, today's software-based systems are inherently frail, and susceptible to attack and manipulation in varying ways. To address the differences between what is conceived and what is delivered we need to think about how software-based systems are actually integrated and deployed, as opposed to how they were designed and envisioned. Generally speaking, the real "deployed" system is what attackers study and attack, not the idealized engineering plans often used to manage the systems.

In other words, it is the software actually used in the field that has to be the focus of assurance efforts. So, if libraries are incorporated and deployed by a compiler, or configuration choices undermine design choices, or someone otherwise exposes a weakness, we need to detect this before our adversaries do.

The various human disciplines that relate to engineering (of any sort) have been shaped by our collective historical learning from working in and with the physical world. Within that world, things are essentially repeatable and sufficient description simply comes down to an adequate understanding of the basic physical science's behaviors and properties. Our training from birth is in this physical world, and our expectations are shaped by that predictable world. Consequently, it is no wonder that we subconsciously and consciously expect that same level of consistency to hold in the cyber domains, though without the evidentiary basis for that belief.

The software-based code and logic underpinning cyber capabilities lack the predictability that the physical world follows. The cyber action and interactions due to the mistakes and flaws in our computer languages, software tools, the training or accountability of software developers, and the ingenuity of attackers doesn't follow a nice scientific formulation. Until we can get those who create, build, and support the cyber systems that people depend on to understand that cyber is a man-made and man-defined environment and it will not follow any rules other than those we impose and enforce upon them, almost any aspect of a system can become an avenue of attack that puts the mission and our people at risk.

Non-Benign Environments with Attackers

We operate in a non-benign cyber world with attackers and attack techniques that need to be considered as "motivated forces of nature." In traditional engineering one is often dealing with known hazards and threat agents (e.g., hurricanes, fires, tornadoes), and generally, with the exception of corrosion, these are one time or short term issues, from which one can recover.

In contrast, attackers in cyber space are persistent, target-specific entities that will work to identify weaknesses in their specific targets. They will evolve their tradecraft to better leverage the weaknesses of their ICT targets and users. Attackers are a given fact of life for the cyber environment that software-based systems will have to work in and survive to support their intended missions. Mission resilience within our cyber-assisted world is very difficult without ICT resilience.

Adversary Evolution

The third thing that differs from what most engineering approaches expect is the speed and adaptability of the attacks. This is especially true with regards to the rapid evolution of the attack techniques and ability of attackers to adapt to change and new elements of the offensive and defensive cyber environment. While attacker and defender have always had a race, the speed of that race and the breadth and scope of changes are orders of magnitude quicker and broader in the software-based cyber realm.

As an example, if one builds a tank it is understood that eventually the adversary will come up with a better weapon that will force the development of a new and better tank or way of using tanks. Both the development of the tank and subsequent countermeasures will likely take years, fitting nicely into the traditional acquisition life cycle.

In contrast, the development and evolution of cyber adversary attacks can change in hours or days. Admittedly, some of these are tactical changes but they can still impact the mission. Even if one were to argue it takes weeks to develop a new strategic cyber-attack technique, this is still far faster than the normal acquisition and systems engineering process.

Integrated System Engineering for Hygiene Assurance

While tainted products can be a concern in their own right, the three differences surrounding the engineering of software-based systems from above and the implications that these differences represent in ensuring the systems meet their mission in spite of the intentions of others needs to be threaded throughout the systems engineering activities when the system in question has any cyber components. So rather than fitting into today’s systems engineering process, that engineering activity itself needs to be adjusted to better fit the challenges that software brings to our systems so that the systems deployed, with all their fuzzy edges and unplanned features are what we validate, verify, and gain assurance about just as those systems are what the adversaries reconnoiter and attack.

A deep, broad, strategic approach to evolving systems engineering to better address cybersecurity issues in software-based systems that covers education, research, legal liabilities and expectations, business understanding, and systems development methodologies is needed. Only this reworked approach can make software-based systems as reliable and resilient as they need to be given the role they play in the myriad of governance, business, security, and personal endeavors they support.

ASSURANCE FOR THE MOST DANGEROUS NON-MALICIOUS ISSUES

There are a wide variety of ways software can become exploitable to an attacker, allowing them to make use of the products in ways that the software’s developers and/or those running the software never intended. With this comes the question as to which of these non-malicious issues should an enterprise focus on eliminating or mitigating? Unfortunately, there is no simple answer to that question. Because different organizations can use the same type of software in vastly different ways, the same flaw could be critical to one and a trivial nuisance to the other.

Two questions that each enterprise needs to be ready to answer are the questions of what any particular piece of software is doing for them, and how dangerous would different failure modes of that software be to the enterprise. This is the “fitness for use” consideration that each mission must address before accepting software into its operational environment.

While there may be “over 1,000 different categories of security mistakes that developers can make” [5] in the Common Weakness Enumeration (CWE) [6], the community-developed dictionary of software weakness types such as constructs in code, design, architecture and deployment of software that can lead to exploitation by attackers, there appear to be only eight different consequences or technical impacts [7], as shown in Table 1, to which these failures lead. In other words, if a weakness manifests itself in a product in an exploitable manner and an attacker successfully exploits it, then there will be one of eight technical impacts or consequences from that weakness. With each CWE entry the “common consequences” field lists the “technical impacts” that can result from each weakness in CWE. The technical impact and its translation into an impact to the mission are important criteria within the DoD’s approach to “Threat and Vulnerability Assessments” within program protection planning (PPP) [8, 9] and can be equally useful to any organization that needs to have reasonable assurance that their software-based capabilities due what they are intended and nothing more.

The collapsing of the hundreds of types of errors into a small set of technical impacts offers a simplification to the question of what an organization should focus on to gain assurance in their software-based products. Instead of trying to remove all weaknesses, they could decide which of the eight impacts are either more or less dangerous to them, given what the software product is doing for their organization. For example, a public web site utilizing Akamai to provide information may not worry about weaknesses that lead to resource consumption denial-of-service exploits but could be extremely concerned about weaknesses that can lead to someone modifying the data. Using this approach they could focus their assurance activities on those weaknesses that could lead to this unacceptable failure mode. The eight technical impacts are listed in Table 1.

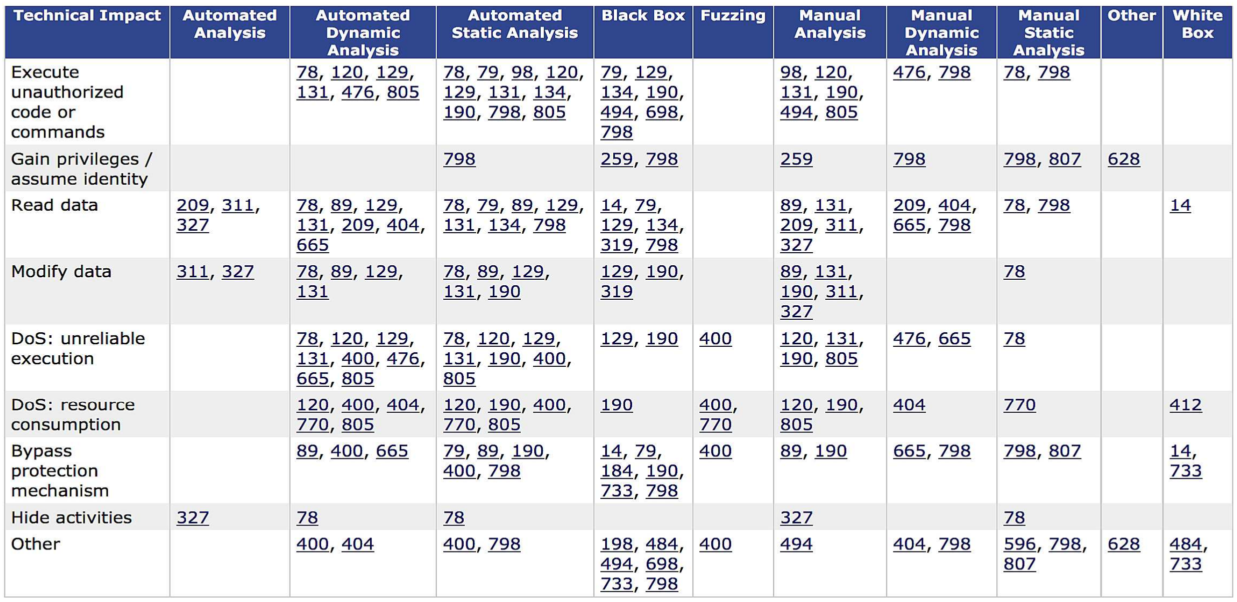
Table 1. Technical Impacts of Software Weaknesses

Technical Impacts of Software Weaknesses:

- Modify data
- Read data
- Denial-of-Service: unreliable execution
- Denial-of-Service: resource consumption
- Execute unauthorized code or commands
- Gain privileges / assume identity
- Bypass protection mechanism
- Hide activities

Similarly, there is a “Detection Methods” field within many CWE entries that conveys information about what types of assessment activities that weakness can be found by. More and more CWE entries have this field filled in over time. The recent Institute of Defense Analysis (IDA) State of the Art Research report conducted for DoD provides additional information for use across CWE in this area. Labels for the Detection Methods being used within CWE at present are: Automated Analysis, Automated Dynamic Analysis, Automated Static Analysis, Black Box, Fuzzing, Manual Analysis, Manual Dynamic Analysis, Manual Static Analysis, Other, and White Box.

This offers a second simplification where stakeholders can now match weaknesses against type of assessment activities, and will thereby gain insights into whether that weakness is still an issue, or whether it has been mitigated or removed. Continuing the example above, using the information in Figure 1, the specific CWEs that can lead to that type of impact can be reviewed and the ones that dynamic analysis, static analysis, and fuzzing can gather evidence about and which ones they can not.



Technical Impact	Automated Analysis	Automated Dynamic Analysis	Automated Static Analysis	Black Box	Fuzzing	Manual Analysis	Manual Dynamic Analysis	Manual Static Analysis	Other	White Box
Execute unauthorized code or commands		78, 120, 129, 131, 476, 805	78, 79, 98, 120, 129, 131, 134, 190, 798, 805	79, 129, 134, 190, 494, 698, 798		98, 120, 131, 190, 494, 805	476, 798	78, 798		
Gain privileges / assume identity			798	259, 798		259	798	798, 807	628	
Read data	209, 311, 327	78, 89, 129, 131, 209, 404, 665	78, 79, 89, 129, 131, 134, 798	14, 79, 129, 134, 319, 798		89, 131, 209, 311, 327	209, 404, 665, 798	78, 798		14
Modify data	311, 327	78, 89, 129, 131	78, 89, 129, 131, 190	129, 190, 319		89, 131, 190, 311, 327		78		
DoS: unreliable execution		78, 120, 129, 131, 400, 476, 665, 805	78, 120, 129, 131, 190, 400, 805	129, 190	400	120, 131, 190, 805	476, 665	78		
DoS: resource consumption		120, 400, 404, 770, 805	120, 190, 400, 770, 805	190	400, 770	120, 190, 805	404	770		412
Bypass protection mechanism		89, 400, 665	79, 89, 190, 400, 798	14, 79, 184, 190, 733, 798	400	89, 190	665, 798	798, 807		14, 733
Hide activities	327	78	78			327		78		
Other		400, 404	400, 798	198, 484, 494, 698, 733, 798	400	494	404, 798	596, 798, 807	628	484, 733

Figure 1: Weakness Technical Impacts by Detection Methods

Understanding the relationship between various assessment/detection methods and the artifacts available over the life-cycle, better enables decision-makers to plan for: specific issue(s) to review; at what point(s) in the effort; using what method(s); and through the use of the coverage claims representations [10] of the various tools and services, which capability(s) could be leveraged, etc. This is depicted in Figure 2.

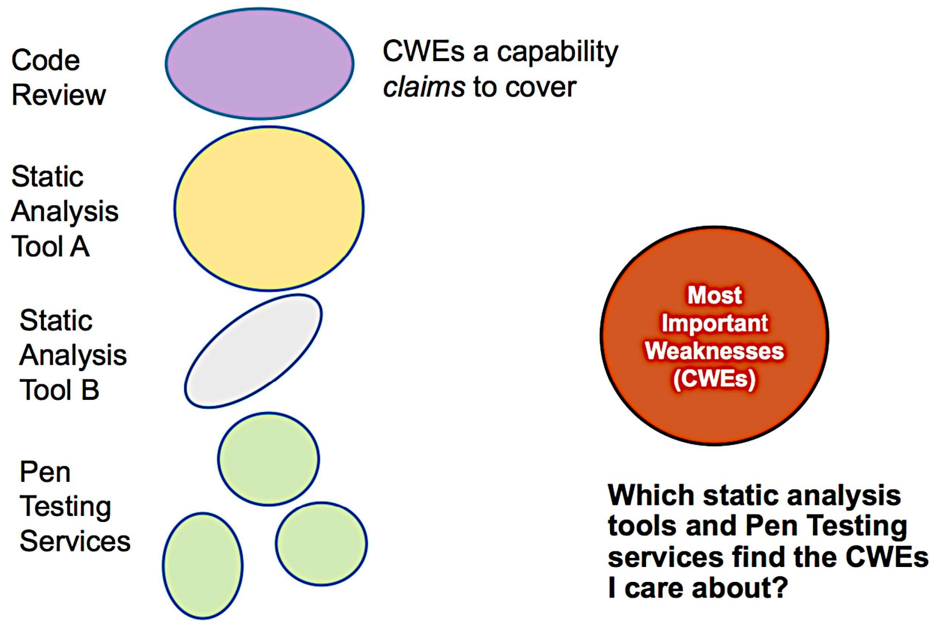

Code Review
CWEs a capability *claims* to cover
Static Analysis Tool A
Static Analysis Tool B
Pen Testing Services
Most Important Weaknesses (CWEs)
Which static analysis tools and Pen Testing services find the CWEs I care about?

Figure 2: Matching Coverage Claims to Your Organization's Needs

This information can assist project staff in planning their assurance activities; it will better enable them to combine the groupings of weaknesses that lead to specific technical impacts with the listing of specific detection methods. This provides information about the presence of specific weaknesses, enabling them to make sure the dangerous ones are addressed.

Figure 1 conveys information associated with the "Software Assurance On-Ramp" portion of the CWE web site. This area of the site is focused on providing help to projects on how they can make use of the information about weaknesses to manage their software security efforts.

Finally, the same type of information in this table could be used to produce an assurance tag for an executable code bundle, leveraging ISO/IEC 19770-2:2009 [11] as implemented for Software Identification (SWID) Tags [12]. SWID Tags can contain assurance information to convey which types of assurance activities and efforts were undertaken against what types of failure modes. The receiving enterprise could then review this tag and match that information against their plan for how they will use the software and what failure modes they are most concerned about. This would be invaluable in determining if sufficient efforts were taken in those areas. [Note: This also supports ISO/IEC 15026 assurance cases.]

MANAGING RISKS ATTRIBUTABLE TO TAINT IN SOFTWARE AND HARDWARE

Hardware follows the physical laws applicable to their composition, electrical characteristics, and construction. Statistical process variations, logical errors of design, or mechanical instabilities may not be originally understood, but can be studied and addressed using general engineering and process improvement methodologies. However, it is clear that software fails from things other than these causes. As discussed above, software follows no laws unless their creators impose them and can fail due to individual implementation mistakes or through the introduction of weaknesses or malicious logic.

Few software developers or systems engineering practitioners have the training and experience to recognize, consider, and avoid these weaknesses. Few (if any) tools or procedures are available to review and test for all weaknesses in a systematic manner. Developers are rarely provided with criteria about what types of problems are possible, and what their presence could mean to the fielded software system and its users.

To manage these risks we cannot just expect to come up with the "right security requirements". We also need to provide a methodology that assists in gaining assurance through the gathering of evidence and showing how that information provides assurance and confidence that the system development process addressed the removal or mitigation of weaknesses that could lead to exploitable vulnerabilities. The changes in revision 4 of National Institute of Standards and Technology (NIST) Special Publication 800-53 [13] directly bring assurance into the security posture equation.

MAKING CHANGE THROUGH BUSINESS VALUE

Key to a successful SCRM strategy (beyond good intelligence about threats) is an approach to engage industry hardware and software developers, manufacturers, and resellers, and not just the "contractors" and "integrators". If necessary appreciation of the problems and requisite risk mitigating behaviors are going to become a core part of the marketplace, then these foundational and ultimate sources of products are where the discipline has to reside. As the software security industry's norms of behavior evolve, those who sell to governments, the critical infrastructures, and the larger global ICT-consuming economy will leverage and adopt these norms in their own operations. All the various facets and aspects of the marketplace have their own business incentives and cost considerations that can be influenced. Given the right set of motivations, we can all benefit from assured software-based products through a sanitized ICT supply chain. That will contribute to the assurance and confidence that products are fit for use in the respective mission and business environments.

Through their own upstream efforts to their parts and component suppliers, and downstream to their customers, the vendor communities must be motivated to control and manage the quality issues for the supply chain going to government as well as to the civilian critical infrastructure and broad consumer markets. This can be in the interest of both the producer industry and the consumers if the right business value proposition can be found. Under that type of market, the government customer can effectively become (almost) a "free rider beneficiary" of these broader supply chain hygiene and assurance changes.

Not to be confused with classical "motivations" for business, aligning the self interests of the business community with the interests of government and industry on concerns such as 'taint' can transform the way everyone conducts their activities. While most community interests cannot drive industry, it is possible to lead industry to a different way of "doing business." Collectively we can show sustained business value propositions to the various participants through either cost avoidance or market dynamics, which reward their behavior in alignment with the collective interests of all participants in our software-based economy and critical infrastructures.

For an example of a behavior change in an industry motivated by a new perceived business value, consider that many of the vendors currently doing public disclosures are doing so because they wanted to include CVE [14] Identifiers in their advisories to their customers. However, they couldn't have CVE Identifiers assigned to a vulnerability issue until there was publicly available information on the issue for CVE to correlate. The vendors were motivated to include CVE Identifiers due to requests from their large enterprise customers who wanted that information so they could track their vulnerability patch/remediation efforts using commercially available tools. CVE Identifiers were the way they planned to integrate those tools. Basically the community created an ecosystem of value propositions that influenced the software product vendors (as well as the vulnerability management vendors) to do things that helped the community, as a whole, work more efficiently and effectively.

Similarly, large enterprises are leveraging CWE Identifiers to coordinate and correlate their internal software quality/security reviews and other assurance efforts. From that starting point, they have been asking the Pen Testing Services and Tools community to include CWE identifiers in their findings. While CWE Identifiers in findings was something that others had cited as good practice, it wasn't until the business value to Pen Testing industry players made sense that they started adopting them and pushing the state-of-the-art to better utilize them.

While motivating business interest usually comes down to incentives and perceptions about market share possibilities, aligning self interests can be an alternate approach to changing things that are both sustainable and win-win for suppliers and customers. These types of symbiotic situations are most certainly available in the various parts of the SCRM challenge space and they present a topic that we collectively need to explore for opportunities and common benefits. Over the past 15 years the community has explored many different ways to influence industry using a wide variety of standards. Community repositories, languages, acceptable usage, or process standards are being considered in terms of the best fit for the variety of different situations faced within the community; and to-date, these have substantively changed the global IT industry in positive and effective ways.

CONCLUSION

The software and hardware fields need to holistically approach the questions around the hygiene and quality activities that provide assurances that products are fit for their intended use. Negative impacts to the mission can be just as deadly and unmanageable in the field from tainted ICT software-based products regardless of intent (from malice or negligence). Within the military, taint considerations can be addressed as part of the 'fitness for use' criteria in program protection planning (PPP) as can the risk based remediation strategies for addressing software vulnerabilities. Use of CWE and the consideration of the technical impacts that software weaknesses can lead to as a guide to reviewing and organization’s hygiene practices, along with the information about which detection methods are best suited for gathering assurance about the presence or absence of exploitable vulnerabilities, can help when managing a project’s assurance activities in a manner that others will understand and can verify. Until both malicious and non-malicious aspects of taint are dealt with in ways that are visible and verifiable there will be a continued lack of confidence and assurance in the delivered capabilities and the supply chain that sourced and services them.

ACKNOWLEDGEMENTS

The summary work contained in this article is based on the discussions with a number of individuals at MITRE and throughout the industry; a special thanks is given for the contributions of the CWE Team members and those working Supply Chain Risk Management, including Joe Jarzombek, DHS Director for Software & Supply Chain Assurance, who provided input to this article. The U.S. Department of Homeland Security (DHS) is acknowledged as the sponsor of this work. The MITRE Corporation operates the Homeland Security System Engineering and Development Institute under contract HSHQDC-09-D-0001 for the Department of Homeland Security.

REFERENCES

[1] “ISO/IEC TR 24772:2013 Information technology -- Information technology -- Programming languages -- Guidance to avoiding vulnerabilities in programming languages through language selection and use”, International Organization for Standardization, (http://standards.iso.org/ittf/PubliclyAvailableStandards/c061457_ISO_IEC_TR_24772_2013.zip)

[2] “National Defense Authorization Act for Fiscal 2013, Public Law 112–239, Section 933”, Jan. 2013 (http://www.gpo.gov/fdsys/pkg/PLAW-112publ239/pdf/PLAW-112publ239.pdf)

[3] “Definition of Taint”, Merriam-Webster Dictionary, Dec. 2013 (http://www.merriam-webster.com/dictionary/taint/)

[4] "Threat-Classification-Glossary", The Web Application Security Consortium (http://projects.webappsec.org/w/page/13246980/Threat-Classification-Glossary)

[5] "Secure Code Starts With Measuring What Developers Know", Information Week, Dec. 19, 2013 (http://www.informationweek.com/security/application-security/secure-code-starts-with-measuring-what-developers-know/d/d-id/1113154)

[6] "The Common Weakness Enumeration (CWE™) Initiative", MITRE Corporation, (https://cwe.mitre.org/)

[7] "Common Weakness Enumeration - Enumeration of Technical Impacts", MITRE Corporation, (https://cwe.mitre.org/cwraf/enum_of_ti.html)

[8] "Requirements Challenges in Addressing Malicious Supply-Chain Threats", Paul R. Popick, and Melinda Reed, INCOSE Insight, July 2013 (http://www.acq.osd.mil/se/docs/ReqChallengesSCThreats-Reed-INCOSE-Vol16-Is2.pdf)

[9] "Program Protection and System Security Engineering", Offic of the Deputy Assistant Secretary of Defense (ODASD) Systems Engineering, January 2014 (http://www.acq.osd.mil/se/initiatives/init_pp-sse.html)

[10] "Common Weakness Enumeration - Coverage Claims Representation", MITRE Corporation, (https://cwe.mitre.org/compatible/ccr.html)

[11] "ISO/IEC 19770-2:2009 Information technology -- Software asset management -- Part 2: Software identification tag", International Organization for Standardization, (http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=53670)

[12] "Software Identification (SWID) Tags", TagVault.org, (http://tagvault.org/swid-tags/)

[13] "NIST Special Publication 800-53 Revision 4 - Security and Privacy Controls for Federal Information Systems and Organizations", National Institute of Standards and Technology, (http://dx.doi.org/10.6028/NIST.SP.800-53r4)

[14] "The Common Vulnerabilities and Exposures (CVE®) Initiative", MITRE Corporation, (https://cve.mitre.org/)