Defining Proactive Software Assurance Practices for Healthier Cyber Ecosystems

Brian Badillo, Harmonia Holdings Group, LLC
Marc Abrams, Harmonia Holdings Group, LLC

Abstract. Distributed security in cyberspace can be performed using many real-time components, from intrusion detection systems to incident management systems. However, these components are reactive rather than proactive. This article will define the space within resilient cyber ecosystems that represents proactive software assurance during the software development lifecycle, from requirements to design to implementation to testing and beyond. Section 1 defines software assurance and provides some examples of current tools and practices in the government that are used for this proactive security practice. Then Section 2 defines the term cyber ecosystem as presented by DHS so that Section 3 can use this concept to explore the space of a software assurance ecosystem. Then in Section 4 we share our experiences in developing a software assurance infrastructure that implements the principals of a software assurance ecosystem and also bridges the gap between proactive and reactive systems. A healthy software assurance ecosystem is critical. The government needs the capabilities to quickly and efficiently certify and accredit systems to minimize vulnerabilities so they can be connected to networks such as the DoD Global Information Grid.

Section 1: How Does the Government Currently Do Software Assurance?

According to the Committee on National Security Systems, software assurance is the, "Level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its lifecycle, and that the software functions in the intended manner" [1]. In this section we mention current software assurance practices and tools used by the DoD, however, many of the principals apply to other agencies and organizations. It is critical that DoD information systems assure software as a proactive security measure before using them in operations. The DoD Information Assurance Certification and Accreditation Process (DIACAP) ensures that risk management practices are put in place during DoD software development. Essentially, DIACAP is a formal framework in which software assurance can take place so that the software assurance process is well documented. DIACAP is a formal, well-defined set of activities, tasks, and management workflow for certifying and accrediting software for the DoD.

There are some software systems in the government that help to enforce workflow in DIACAP including DoD's Enterprise Mission Assurance Support Service and the Air Force's Enterprise Information Technology Data Repository. These systems assist parties undergoing DIACAP by providing management services for workflow among the various roles in the DIACAP process, report generation capabilities for DIACAP requirements, and repositories for generating required reports, and repository capabilities for data pertinent to DIACAP.

In support of secure systems, the Defense Information Systems Agency has introduced Security Technical Implementation Guides that help define specific ways to ensure security in a system. The Application Security and Development STIG is particularly pertinent to software assurance because it defines specific guidelines to be followed by application designers to ensure security (e.g., "The Designer will ensure the application does not display account passwords as clear text" [2].).

The next section summarizes how DHS defines a cyber ecosystem so that we can define a software assurance ecosystem in similar terms in Section 3. Then in Section 4 we share our experiences building a software assurance infrastructure to illustrate the principals.

Section 2: What Is a Cyber Ecosystem?

On March 23, 2011, DHS posted a blog entry with a white paper titled, "Enabling Distributed Security in Cyberspace" [3]. The white paper provides an overview of distributed cyber security approaches and represents the collective vision of 13 federal agencies towards a healthy cyber ecosystem. A cyber ecosystem is defined as the set of diverse participants (which include cyber devices such as computers, software, and communications technologies) that interoperate. However, the ecosystem does not stop at cyber devices, but also includes other participants such as private firms, non-profits, governments, individuals, processes, etc.

The white paper presents three building blocks of cyber ecosystems: Automation, Interoperability, and Authentication. The ecosystem described in the white paper is the operational side of the distributed cyber ecosystem (i.e., running servers, network devices, production software, etc.). The operational side necessitates reactive security measures such as intrusion detection and real-time courses of action. The next section uses the three building blocks to describe a software assurance ecosystem, which is the software development side of cyber ecosystems, and includes requirements, design, implementation, and test stages. In contrast to the operational side, the security in the software development side is proactive in nature and includes security activities such as software assurance.

Section 3: How Does Software Assurance Fit in Cyber Ecosystems?

Software assurance is an important part of any software development project to meet quality, safety, and security requirements. However, in today's software development world, enterprise software assurance capabilities must match the security needs of the growing and diverse cyber ecosystem. For example, many software vendors utilize open source software or acquire COTS software to include in their solution. In this case, each software component is now part of the software assurance ecosystem. Furthermore, the intercommunication between software necessitates standardization of software assurance capabilities to provide a common interface.

The same three building blocks used to describe the secure operations portion of healthy

cyber ecosystems can be used to describe software assurance ecosystems.

Automation

Software assurance tools can aid software developers in making important security, quality, and safety decisions during the requirements, design, implementation, and testing stages. These tools automate parts of the software assurance process by performing much of the brute force work for identifying software weaknesses, which then allows developers to sift through the suspected weaknesses identified by automated tools and decide which weaknesses need further action (later sections describe tool automation as a way to collect evidence for software assurance cases). Software assurance tools can be classified into several analysis approaches and techniques, each of which have specific advantages and identify a specific subset of software weaknesses. A classification of tools is given by NIST Software Assurance Metrics And Tool Evaluation (SAMATE) project <http://samate.nist.gov/index.php/Tool_Survey.html> and includes such tool classes as static analysis, dynamic analysis, pedigree analysis, binary code scanners, disassembler analysis, binary fault injection, fuzzing, etc.

Interoperability

Given the multitude of tools in the software assurance ecosystem (more than 75 listed on the SAMATE project website), standards for interoperability among these tools is a necessity.

One such standard is the Common Weakness Enumeration (CWE) <http://cwe.mitre.org>. CWE is a dictionary of software weakness types developed by MITRE, intended to facilitate communication about weaknesses in software such as code constructs that are prone to memory leaks, susceptible to injection attacks, etc. From person to person, descriptions of these weaknesses can often be inconsistent; the CWE dictionary gives a standardized reference point as well as levels of specificity for these weaknesses. They are organized in a hierarchy, with general weaknesses (e.g., CWE-710: Coding Standards Violation) at the top level, getting increasingly more specific towards the lower levels (e.g., CWE-259: Use of Hard-coded Password). This hierarchy allows weaknesses to be related to each other with parent/child relationships. Some weaknesses also relate to each other with a precede/follow relationship that suggests that one weakness may be caused by another. Each weakness has a self-explanatory title, accompanied by an index. For example, the weakness described as NULL Pointer Dereference has the index of 476. Many tools available today already reference the CWE indices in their output. CWEs are part of a larger initiative called Making Security Measurable <http://measurablesecurity.mitre.org> to standardize system security.

With tools in the software assurance ecosystem using CWEs to represent their output, developer participants in the ecosystem can use a wider array of tools because using each tool that outputs the familiar CWEs will be easier to learn. Using a combination of tools for software assurance in turn leads to more assurance coverage of software. For example, consider a developer who is already familiar with a static analysis tool of their choice for detecting memory management weaknesses in code. Suppose that their familiar tool maps the weaknesses it identifies to CWEs. Since the developer already has knowledge concerning the weaknesses identified by their tool of choice, they are able to easily use and understand other tools that also produce CWE output.

Another advantage of interoperability is the ability to leverage collective bodies of knowledge concerning common assurance cases. An assurance case is defined as claims, arguments, and evidence that support the contention of particular software requirements [4]. In effect, an assurance case builds confidence in a system given evidence found by automated software assurance tools.

The Software Assurance Evidence Metamodel (SAEM) and the Argument Metamodel (ARM) are standardized models for representing parts of an assurance case, both of which were developed by Object Management Group's (OMG) Systems Assurance Task Force (SATF) <http://sysa.omg.org>. Arguments are logic that combines evidence and other asserted claims in a meaningful way to support or refute another particular claim [5]. The most primitive building blocks arguments are premises and conclusions. The argument asserts that if all the premises are accepted as true, then the conclusion must also be accepted. Arguments can be chained together such that the conclusion of one argument can provide the input to a premise in another argument. A CWE could contribute to evidence in a claim. However, an evidence item in SAEM contains additional useful information to be used in an assurance case, such as the evidence collection method used. Information that SAEM might include is the name and version of the tool used to identify the CWE, the time that the CWE was assessed, or the confidence level given to the evidence item. In addition, SAEM represents whether the evidence strengthens or weakens an assertion made by the evidence (which would in turn support an argument which uses the evidence to make a claim).

CWE, SAEM, and ARM are part of a larger Software Assurance Automation Protocol (SwAAP). SwAAP is a protocol composed of many interrelated standards [6].

Authentication

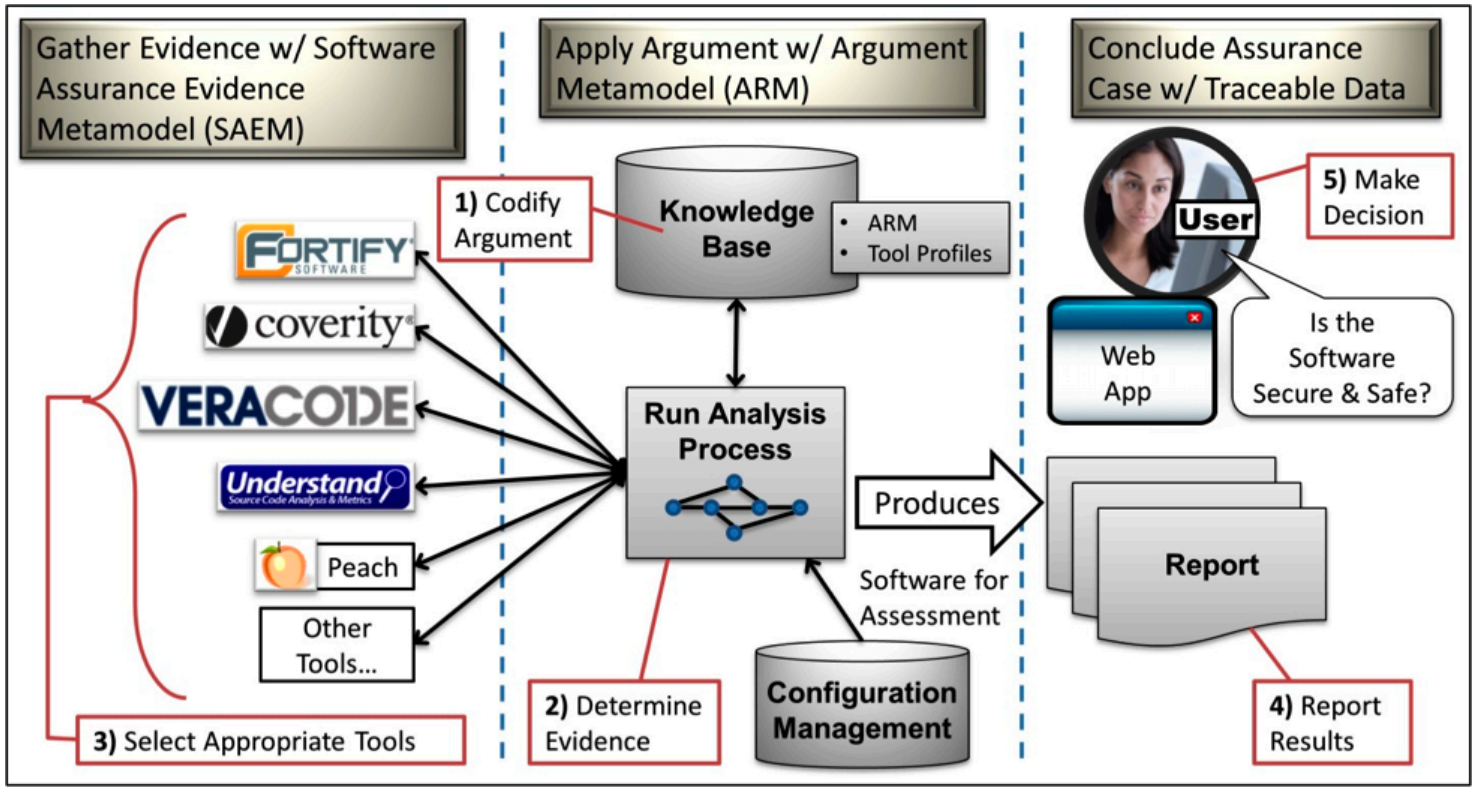
In the operations realm of the cyber ecosystem, authentication means making sure that the users of cyber devices are who they say they are (which includes both human and machine users). However, in the software assurance ecosystem, authentication means making sure that codebases are in fact the ones that have undergone the extensive assurance processes that they say they have. For instance, suppose that a library for protecting against Cross Site Scripting (XSS) is used in the security of a critical web application. The developers of the web application have decided to use this particular library because it has been vetted by Independent Verification and Validation (IV&V). However, when packaging the web application for production use, the library is not authenticated (i.e., the codebase is not checked to be from the expected supplier) and a malicious look-alike library is used in deployment. Now, the web application contains open vulnerabilities for attack.

Software supply chain integrity is another facet of authentication in the software ecosystem. According to SAFEcode not only must codebases be authenticated to make sure that they are the expected software, but the software must be expected to use secure, safe, and quality assurance processes during development [7]. It is therefore important to consider developer pedigree and policy during sourcing, development, and distribution.

Section 4: Building a Software Assurance Infrastructure

In this section, we discuss our experiences in applying the principles above by combining open standards and open source technologies

Figure 1: Infrastructure for a Software Assurance Ecosystem Overview



into an infrastructure. Software tooling that supports the three building blocks of software assurance (automation, interoperability, and authentication) is needed to make pre-incident detection practices during the software development lifecycle a reality. For instance, there are many software assurance tools that can be used to identify weaknesses in code, some of which already conform to the SwAAP standards (e.g., produce CWE output). However, traditional tools perform a single class of analysis approach or technique (i.e., static analysis, dynamic analysis, fuzzing, etc.) that provides them certain strengths and shortcomings. In addition, tools are usually focused on finding weaknesses in code developed in a particular language or for a specific platform. Furthermore, any single tool is subject to generating false-positive findings (e.g., a weakness in code that does not lead to vulnerability). In the rest of the section, we discuss our experiences in implementing the principles of a software assurance ecosystem through a software assurance infrastructure called Conforma.

In our experience while building the Conforma software assurance infrastructure we found that the infrastructure can provide the foundation to combine best-of-breed tools from many tool classes that have overlapping CWE coverage to increase confidence and reduce false-positive findings in software assurance. To accomplish this, the Conforma infrastructure contains a Tool Profile for each third-party tool plugged in to the infrastructure. The profile uses Coverage Claims Representation (CCR) [cwe.mitre.org/compatible/ccr.html] from the CWE standard to express which CWEs each tool claims to uncover. This profile enables Conforma to orchestrate the execution of appropriate third-party tools given a set of evidence that must be found to support an assurance case. While some third-party tools already produce CWEs (e.g., Fortify, Veracode, Klocwork), Conforma must map to a CWE each message generated by tools that do not (e.g., Splint, Peach). With the number of tools available, cross checking between tools using the common CWE output provides a base evaluation of the confidence level regarding the results. A software assurance infrastructure, such as Conforma, computes percentages involving the number of tools that found a certain error. For example, some types of tools reliably find particular weaknesses, but if multiple tools report the same weakness then a user's confidence that the weakness is a valid result, and not a false-positive, increases. In this respect, a software assurance infrastructure harnesses an ecosystem of tools to the advantage of the user by increasing confidence and reducing false-positives.

In Figure 1, there are three columns, which represent the three major parts of our software assurance infrastructure design (and the associated OMG standards that they leverage). The left side of the figure shows a list of software assurance tools that are plugged into the infrastructure. Each of these tools performs some sort of analysis, which produces CWEs that the infrastructure wraps into evidence in the form of SAEM, which is sent back to the infrastructure. The middle of the figure depicts a knowledge base containing rules and workflows that

support ARM and that are executed by the infrastructure. In other words, the rules will model claims in the form of premises that must be satisfied in order for certain conclusions to be made. The right side of the figure shows the users of the infrastructure (human participants in the software assurance ecosystem) using a web application UI to make conclusions about the software under assurance assessment and decide whether the evidence, arguments, and claims made in the assurance cases indicate that the software is ready to become an operating member of a healthy cyber ecosystem.

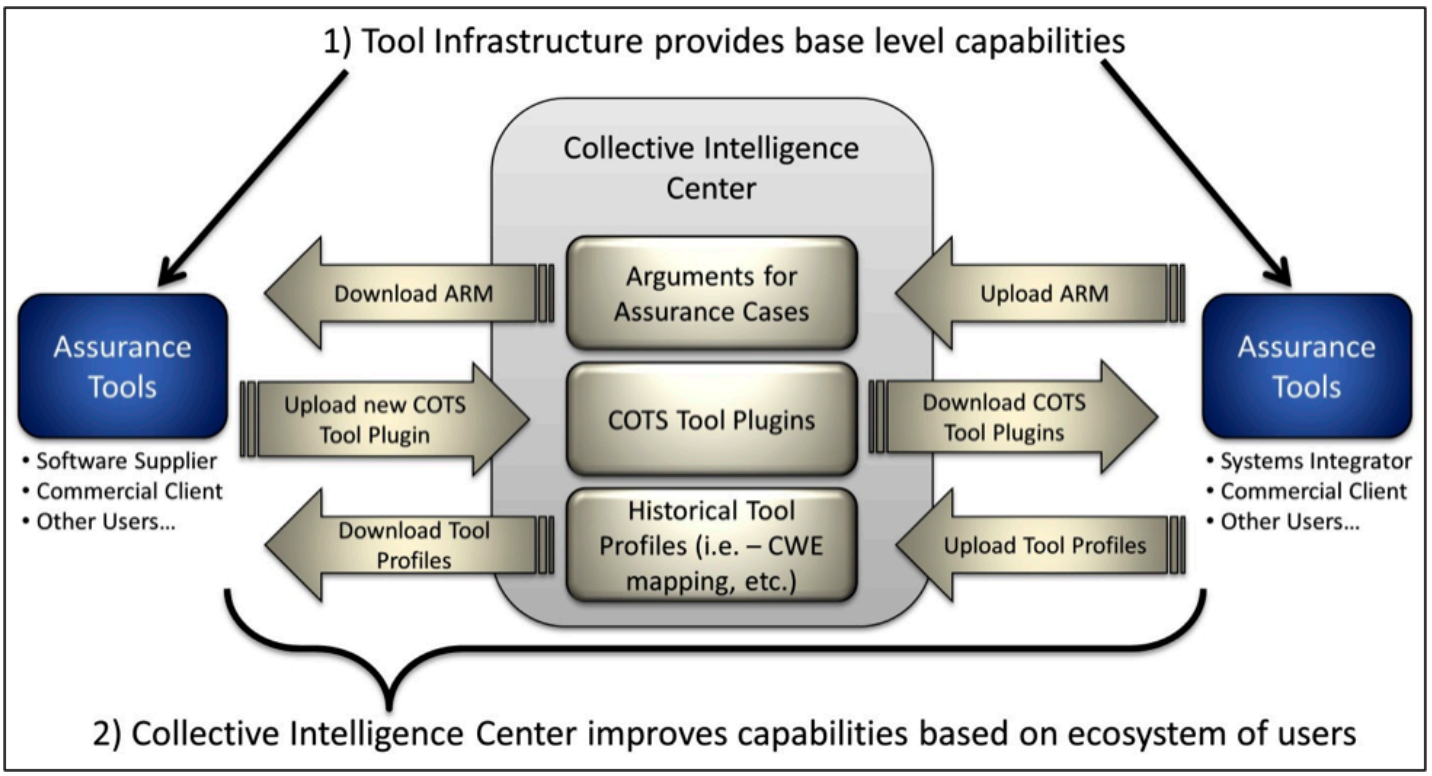
Each of the red boxes in Figure 1, in the order of how each part is used, is described in the list below.

1. **Codify Argument:** Arguments are codified (written in the standard ARM format) and stored in the Knowledge Base.
2. **Determine Evidence:** The codified ARM model is used in the Analysis Process to determine the evidence that is needed for certain claims to be made.
3. **Select Appropriate Tools:** With the evidence identified along with the Tool Profiles stored in the Knowledge Base (which describe tool coverage using CWE Coverage Claims Representation), the appropriate tools are executed by the infrastructure. These tools produce CWEs which should be used to strengthen or weaken evidentiary assertions.
4. **Report Results:** After all tools have been executed and the Analysis Process is complete, the user can initiate the generation of a report. The report shows the resulting claims about the software that can be made using the evidence that has been found using the appropriate tools.
5. **Make Decision:** Finally, the user can answer the question of whether or not the software is secure, safe, and of good quality. They can make a claim that is backed up by the arguments made in conjunction with evidence found by the infrastructure.

An infrastructure such as Conforma can be deployed within an enterprise to support the needs of a single software development house. However, over the course of our work developing a software assurance infrastructure we have learned that the power of the infrastructure is truly realized when deployed on a cloud environment where software assurance community cooperation can be achieved. In a cooperative environment, the infrastructure learns from the software assurance ecosystem participants by continuously expanding its Knowledge Base in real-time, which can then be used across the infrastructure for improved software assurance. This concept is illustrated in Figure 2 where two sets of Assurance Tools (far right and left sides) share through the Collective Intelligence Center some ARM data for assurance cases, Tool Profiles for up-to-date tool data including new CWE mappings and CCR coverage, and COTS Tool Plugins for increased interoperability between tools. The infrastructure deployment depicted in Figure 2 fosters a software assurance ecosystem through knowledge sharing.

Conforma itself is designed to learn how to better assure software when software is in operation in the cyber ecosystem. Conforma reacts to detected vulnerabilities and attacks during operation and learns which parts of the code base were not properly assessed in the assurance process. If there were tools that produced evidence that was originally deemed false-positive, then the tool profile is updated to reflect a different

Figure 2: Community Deployment Overview



level of confidence in that particular tool. For example, suppose a particular tool (with which Conforma associates a high level of confidence) showed that an input field in a user interface was being properly validated to protect against an XSS attack. If a successful XSS attack on that input field is detected, Conforma would promptly lower the level of confidence associated with that particular tool for assuring input field validation. This information would then be shared across the infrastructure to all users in the software assurance ecosystem as part of the tool's profile.

Securing a cyber ecosystem can be divided into two methods: reactive and proactive. In our experience building a software assurance infrastructure, we found that we could complement reactive security with proactive software assurance, and vice versa. Thus, in addition to fostering a tighter software assurance ecosystem, Conforma bridges the gap between the operations and development lifecycle phases of software in cyber ecosystems by using both reactive and proactive security measures. New vulnerabilities and new attacks continue to be identified every day in the cyber world. It is important that the software assurance community learns how to protect against these vulnerabilities. The Conforma infrastructure is designed to improve its own assurance processes by detecting vulnerabilities and attacks during operation of software that was assured within its infrastructure.

REFERENCES

1. United States. Committee on National Security Systems. National Information Assurance Glossary: CNSS Instruction No. 4009. By Richard C. Schaeffer, Jr. 26 Apr. 2010. Web.
2. United States. DISA for Department of Defense. Application Security and Development: Security Technical Implementation Guide. Version 3, Release 4, 28 Oct. 2011. Web. <http://iase.disa.mil/stigs/app_security/app_sec/u_application_security_dev_stig_v3r4_20111028.zip>.
3. United States. Department of Homeland Security. Enabling Distributed Security in Cyberspace. U.S. Department of Homeland Security, 23 Mar. 2011. Web.
4. Software Assurance Evidence Metamodel (SAEM). Publication no. Ptc/2010-08-37. Object Management Group, Inc. (OMG), Aug. 2010. Web. <www.omg.org/cgi-bin/doc?ptc/10-08-37.pdf>.
5. Argumentation Metamodel (ARM). Publication no. Ptc/2010-08-36. Object Management Group, Inc. (OMG), Aug. 2010. Web. <http://www.omg.org/cgi-bin/doc?ptc/10-08-36.pdf>.
6. Jarzombek, Joe. "Public/Private Collaboration Efforts for Enterprise Security Automation." Speech. Software Assurance Forum: Building Security In. Baltimore Convention Center. 27 Sept. 2010. Security Content Automation Protocol. U.S. Department of Commerce, NIST. Web. <http://scap.nist.gov/events/2010/itsac/presentations/day1/Software_Assurance-PublicPrivate_Collaboration_Efforts_for_Enterprise_Security_Automation.pdf>.
7. Reddy, Dan, Brad Minnis, Chris Fagan, Cheri McGuire, Paul Nicholas, Diego Baldini, Janne Uusilehto, Gunter Bitz, Yuecel Karabulut, and Gary Phillips. The Software Supply Chain Integrity Framework: Defining Risks and Responsibilities for Securing Software in the Global Supply Chain. Tech. Ed. Stacy Simpson. SAFECode, 21 July 2009. Web.

ABOUT THE AUTHORS

Brian Badillo, M.S., (Computer Science, Virginia Polytechnic University) is lead software engineer at Harmonia. He successfully completed seven Phase I Small Business Innovative Research (SBIR) topics, bringing three to Phase II, and has been awarded three Phase I SBIR topics. He has used many community efforts such as DHS' "Building Security In," MITRE's "Making Security Measurable," Open Web Application Security Project, and Microsoft's "Security Lifecycle Development." Using this research, he led Conforma development.

Harmonia Holdings Group, LLC
2020 Kraft Drive, Suite 1000
Blacksburg, VA 24060
Phone: 540-951-5900 Ext. 255
Fax: 540-951-5911
E-mail: bbadillo@harmonia.com

Marc Abrams, Ph.D., (Computer Science, University of Maryland; Post Doctoral Study, Stanford University) is Harmonia's President and CTO, providing technical and business leadership and overseeing all technical activities. He has more than 20 years of professional experience in the design, development, deployment, and maintenance of software and information networks, focusing on user interfaces. He architected Harmonia's LiquidApps® tool suite and led its implementation in the Army's ATIA-M project, US Navy's DDG 1000 destroyer, and Tomahawk weapons control system.

Harmonia Holdings Group, LLC
2020 Kraft Drive, Suite 1000
Blacksburg, VA 24060
Phone: 540-951-5901
Fax: 540-951-5911
E-mail: mabrams@harmonia.com