Threat analysis in the software development lifecycle

J. Whitmore
S. Türpe
S. Triller
A. Poller
C. Carlson

Businesses and governments that deploy and operate IT (information technology) systems continue to seek assurance that software they procure has the security characteristics they expect. The criteria used to evaluate the security of software are expanding from static sets of functional and assurance requirements to complex sets of evidence related to development practices for design, coding, testing, and support, plus consideration of security in the supply chain. To meet these evolving expectations, creators of software are faced with the challenge of consistently and continuously applying the most current knowledge about risks, threats, and weaknesses to their existing and new software assets. Yet the practice of threat analysis remains an art form that is highly subjective and reserved for a small community of security experts. This paper reviews the findings of an IBM-sponsored project with the Fraunhofer Institute for Secure Information Technology (SIT) and the Technische Universität Darmstadt. This project investigated aspects of security in software development, including practical methods for threat analysis. The project also examined existing methods and tools, assessing their efficacy for software development within an open-source software supply chain. These efforts yielded valuable insights plus an automated tool and knowledge base that has the potential for overcoming some of the current limitations of secure development on a large scale.

Introduction

The characteristics and the uses of software have evolved and diversified over the past 30 years, along with the methods, tools, and practices employed to design and build the software. Before the advent of distributed computing in the early 1980s, software was developed and maintained by small groups of engineers and computer science professionals in tightly controlled environments. Software from the mainframe or mini-computer era could be reasonably classified as either *industrial software* or *high-assurance software*, where industrial software supported businesses for billing, inventory, accounts receivable, manufacturing, etc., and high-assurance software supported certain critical systems such as government defense and intelligence [1].

Distributed computing changed the face of software and software development with an expanding community of programmers, and easy access to assemblers, compilers, testing tools, new programming languages, new programming models, and a growing body of interoperable software. This movement brought general-purpose software to businesses, governments, and individual consumers. The assimilation of new software products and services led to the Internet of today with mobile computing, personal computing, social computing, business-to-consumer computing, business-to-business computing, academic computing, and computing in support of critical infrastructure. Arguably, in the evolution of information technology, the distinction between general-purpose software, industrial software, and high-assurance software has blurred from the points of view of users, buyers, and developers. There are many contributing factors, including a view that the demand for

Digital Object Identifier: 10.1147/JRD.2013.2288060

function, consumability, and affordability superseded the considerations for security.

The growing dependence on the Internet as the default operating environment for all uses of software, and the rise of Internet-related security risks for all users of computing, has led to a renaissance in awareness of the importance of software assurance. For the purpose of this paper, *software assurance* indicates a "level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its lifecycle, and that the software functions in the intended manner" [2].

Organizing and sustaining a focus on software assurance across the development lifecycle is a complex undertaking. Notable efforts for software assurance include: Microsoft Security Development Lifecycle [3], Building Security in Maturity Model (BSIMM) [4], and the IBM Secure Engineering Framework [5]. These and other similar programs either describe or prescribe a set of practices that promote a concentrated focus on security in all aspects of software development.

This paper provides a view of the research and development associated with secure engineering efforts in IBM and, in particular, the investigation of threat analysis in the development process. In brief, the term *secure engineering* generally refers to a set of practices followed by software and service development teams. While it is outside the scope of this paper to propose a benchmark for software security, our examination strongly suggests that there may be value to both software developers and software consumers in revisiting classifications of software for both assurance and recommended use.

Secure engineering in IBM

There is a long-standing focus on software assurance in IBM, as represented by the Statement of Integrity for the mainframe operating system that is now referred to as z/OS* [6], as well as the Statement of Integrity for the mainframe virtualization software referred to as z/VM* [7]. With the formation of IBM Software Group in 1995 [8] and the subsequent evolution of the distributed software portfolio through organic growth and acquisition, a substantial diversification of the development community ensued. This evolution led a study of software assurance practices by a cross-IBM team of security leaders. The team cataloged best practices for security and produced an internal white paper. The white paper prompted an initiative with the goal of consistent adoption of the best practices across all development teams.

For IBM, software assurance in product development is characterized by maturity of practices in four "pillars" [9]: a Common Development Process, a Secure Engineering Framework, a Continuous Improvement Quality Management Program, and Supply Chain Security. The Secure Engineering Framework pillar is further defined by a set of eight essential practices that are indicators of success in the drive to build secure software. The essential practices are as follows: Education and Awareness, Project Planning, Risk Assessment and Threat Modeling, Security Requirements, Secure Coding, Security Testing, Security Documentation, and Security Incident Response. The promotion, enablement, and support of adoption and execution of these essential practices by development teams across IBM is coordinated the Chief Security Compliance Officer, with accountability to development and business executives.

It is important to note that most contemporary initiatives to increase the focus on security in the software development lifecycle rely on a population of security experts that oversee development projects. Secure Engineering efforts in IBM adopted an alternate philosophy. Because of the size, geographic, and organizational diversity of the development population, the large number of concurrent major development projects per year, plus the role-based makeup of IBM software development teams, we have determined that security in development is best facilitated by making security a shared responsibility of all members within each development team.

Threat analysis in software development

There are three terms that are commonly associated with analysis of threats: threat analysis, threat modeling, and threat assessment. An informal search shows that "threat analysis" has broad applicability for computer security, physical security, and other security contexts, whereas, "threat modeling" is generally associated with computer security. For the purpose of this paper, threat analysis and threat modeling are considered as essentially equivalent. The comparison of threat analysis and threat assessment provides more insight into development practices.

The National Information Assurance Glossary [2] equates threat analysis with threat assessment, defining threat assessment as the "process of formally evaluating the degree of threat to an information system or enterprise and describing the nature of the threat." This treatment of threat analysis and threat assessment raises a question as to whether a threat assessment and threat analysis are equivalent. There are differences in performing a threat assessment on an operational system or enterprise compared with performing a threat assessment on a design or design concept. Operational systems can be analyzed and tested for susceptibility to threats, whereas designs cannot be tested prior to the point at which the design has been built.

It follows that threat analysis in software development examines the architectural representation of the system to

be built, considering available methods and knowledge to reduce the likelihood that the architectural specification and development practices will yield a system that includes vulnerabilities and/or weaknesses. In addition, it follows that the measure of effectiveness for threat analysis is related to the operational resilience and vulnerability history of the deployed system. Four tangible benefits are realized from threat analysis: "First, it will aid prioritizing types of attacks to address. Second, it will help more effectively mitigate risk. Third, it will augment assessments with new potential attack vectors. Finally, it will identify business-logic flaws and other critical vulnerabilities that expose core business assets" [10].

A literature review of threat analysis practices in software development shows a small number of common methods. These include adapting consultative approaches for assessment of security controls and defenses in running systems; applying a taxonomy of security issues in a structured development process; applying patterns and checklists; or associating risks and outcomes to scenarios, interactions, relationships and consequences. In all cases, the goal of the analysis is to identify and assess risks in an application or system, which can then guide subsequent design and coding decisions to mitigate those risks or accept the risk, depending on the priority and impact.

Notable methods for threat analysis in software development include STRIDE [11] (which is an acronym for spoofing identity, tampering with data, repudiation, information disclosure, denial of service, and elevation of privileges), a practice within the Microsoft Security Development Lifecycle that uses bottom-up modeling of the software being built, with consideration for how the STRIDE taxonomy of threats affect components and interactions. A second notable approach is CORAS [12], which is a method for security analysis that relies on consultative workshops and top-down modeling of business and technology interactions to identify risks and consequences. Another method is the Risk Assessment and Threat Modeling Practice within the IBM Secure Engineering effort. An IBM Secure Engineering publication [9] uses the term "threat modeling," describing the general flow of the activity as follows: (a) identify the assets, (b) identify the potential threats, (c) assign an impact for each threat, (d) determine the probability of compromise, (e) rank the risks, and (f) define mitigating counter-measures. The output document is commonly referred to as a Threat Model and Mitigation plan.

Separate from the referenced threat analysis methods, there are a variety of graphical and analytical techniques that support detailed analysis and modeling activities, such as directed graphs [13]; attack trees in the context of smaller scenarios such as web applications [14]; threat trees, misuse cases, and cause and effect diagrams [15]; and formal use of annotated data flow diagrams to enumerate threats [16]. The TAM (threat analysis modeling) process [17] is an example of attack path analysis [18]. Other examples of analytic techniques include flow analysis [19], fuzzy logic [20], and Petri nets [21].

The methods and techniques described above follow the rationale that once the design is appropriately documented, a person that is skilled in the art of recognizing the symptoms and triggers of risk and weakness in design will be able to guide the developers toward mitigation of related threats. This philosophy raises questions about consistency of results for threat analysis from practitioner to practitioner and design to design.

Many companies report that new software engineers arrive from universities with an understanding of the general parameters of security technologies, but little if any understanding of how to apply that knowledge in their coding [22]. Many universities teach the principles of operational security, i.e., network security and security management, without teaching how to develop software that cannot be subverted. Even for experienced software developers, it is often difficult to know how to apply the knowledge of particular vulnerabilities to produce truly secure code; prescriptive information for developers is most helpful, but also appears to be in limited supply. The issue is exacerbated by the fact that students and IT (information technology) professionals generally lack the skill to create and manage complex policies needed for a sustainable security program for a software development environment [23].

There is a growing body of knowledge on commonly occurring threats, vulnerabilities, and weaknesses in software. This information is captured, cataloged, and analyzed in projects such as: IBM X-Force Trend Analysis [24], the Open Web Application Security Project (OWASP) Top 10 Project [25], the SANS Institute Top 25 Dangerous Software Errors [26], the Common Weakness Enumeration data published by MITRE Corporation [27], and the National Vulnerability Database sponsored by the U.S. Department of Homeland Security and US-CERT (United States Computer Emergency Readiness Team) [28].

Investigation of process and tooling for threat analysis

IBM sponsored a project with Fraunhofer Institute for Secure Information Technology (SIT) [29] and the Technische Universität Darmstadt [30] to examine several dimensions of threat analysis in software development. The project spanned two years. The first year focused on research, and the second year focused on development of practical methods and tools.

The project team was challenged to select a method for threat analysis that was flexible and adaptable to a wide

range of project types, development styles, and software deliverables.

The research activity was organized into four work streams: (1) a review of current practices and tools for threat analysis, (2) a review of security education curricula and gaps as related to the IBM software development community, (3) an examination of security in selected open source components, along with (4) an analysis of the changed landscape for product evaluation as information technology moves from packaged software to online services and cloud computing. The development activity expanded upon the research findings, proposing a flexible consultative approach and supporting tools.

The research aspect of the project examined several methods for threat analysis and concluded that there were significant limitations and drawbacks. The popularized tools and techniques involved an assumption of an advanced level of understanding of security concepts and analytical techniques to detect anomalies and make risk-based assessments. Controlled experiments showed a lack of consistent results when the same design problem was presented to multiple analysts. The project determined that there were several shortcomings in the methods when applied to projects using iterative and agile styles of development. The project also concluded that existing threat analysis methods did not scale well for large projects and were potentially irrelevant for teams building small modular reusable components.

Defining a normative approach for threat analysis

As the IBM development community and the software portfolio have grown, the need for a consistent and extensible method for threat analysis has become increasingly important. IBM maintains a large community of computer science and engineering professionals who work on development projects that scale from small components, to enterprise software products, to complex software solutions and services. IBM development teams are staffed by individuals in recognized roles that include project managers, architects, developers, software engineers, build specialists, testers, information planners, and support specialists.

Architects and software engineers routinely develop requirements, produce high-level and detailed designs, and provide guidance to those in other roles on development teams. Enabling new skills for architects and software engineers has the potential to satisfy a recognized shortcoming related to traditional threat analysis such that developers had difficulty converting the output of threat analyses into actionable tasks in coding and build integration. In addition, it was advantageous to define the activity as an extension to the job role and skills of architects and software engineers, who should be familiar with various types of modeling. This thought process is reflected in a set of requirements that drove the development activity:

Requirement 1—The secure engineering approach should be adaptable to developer's skills and project resources available.
Requirement 2—We need to achieve consistency among the results or, at the least, decisions. Therefore, development actions should be explicitly bound to analysis results.
Requirement 3—Our secure engineering approach should start as early as possible in the software development process to avoid expensive remediation costs.
Requirement 4—We should employ available security expert knowledge, in terms of making this knowledge accessible to the developers in the project who may currently lack this expertise.
Requirement 5—Our approach should be suitable or adaptable to various software development paradigms such as waterfall or agile paradigms.
Requirement 6—We should address the different tasks and strategies of the different teams involved in a software development effort, in order to benefit from the established ways in which these teams interact.

Along with these guiding requirements, it is important that threat analysis practitioners have sufficient context and flexibility to support an analysis that is consistent with the function and intended operation of the software being built. The proposed context recognized three perspectives, to be resolved in a set of architectural decisions that would be carried through the development project and delegated to the individuals and roles. These perspectives are:

Software-centric—How does the software design, architecture or implementation allow or support harmful attacks?
Attacker-centric—How might attackers compromise the software, system, or service?
Asset-centric—What information assets are created, stored, updated, or used, and how might they be compromised?

Within this structure, the IBM development team can set a focus that is consistent with what they know about the operational environment for the software. Teams who build small, reusable software components focus on software design and construction issues, along with a small set of operationally relevant use cases and abuse cases. Teams who build more complex integrated solutions and services would incorporate a richer set of use cases and abuse cases, as well as special consideration for protection of information assets for deployers in specific industry segments, such as banking and finance and healthcare. With a consistent approach for all teams, complex offerings can make use

of and aggregate the output of threat analyses from teams' building components and foundational products.

Risk assessment and threat modeling as an essential practice

Factoring in the requirements and constraints described in the previous section, the project team recommended a structured process to incorporate both the Risk Assessment and Threat Modeling aspects of the Secure Engineering Framework into a standard practice. The four-step process was condensed from the OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation) [31] method as documented by the Software Engineering Institute at Carnegie Mellon University: (1) describe the software to be built and the operating environment, (2) identify the in-scope risks from the software-centric, asset-centric, and attacker-centric perspectives described above, (3) examine the risks in relation to known threats and threat vectors, and (4) develop a mitigation plan.

Depending on the type of software being built, a development team has the flexibility to include one or more of the perspectives defined in the previous section. The software-centric view of a threat model is represented by the insight and body of knowledge associated with applying best practices for secure design, secure coding, and security testing. The attacker-centric view is represented by representing good and bad actors and how they interact with the target software in both planned and unplanned ways. The asset-centric view is represented by the enumerated information assets, and their intended and unintended uses. Of these views, the asset-centric perspective is the most familiar to architects and software engineers, since it is common to document business asset lists and use-cases as part of high-level design.

The initial release of the threat modeling process was supported by a set of templates and educational materials to be used with pilot projects. These deliverables were accompanied by a modest, self-contained software application. The purpose of the software application was to provide easy access to reference material for developing the software-centric perspective of the threat analysis. The software application shown in **Figure 1** consisted of downloadable JavaScript** and XML (eXtensible Markup Language) data files that contained the MITRE CWE (Common Weakness Enumeration) database [27]. The user interface provided for filtering and display of CWE entries based on a predefined taxonomy and text search. Each displayed entry included a link to the full CWE content on the MITRE website.

The first experiments with the tool focused on viewing the CWE data using the taxonomy provided in the XML data. These filtering criteria included Language Class, Language, Operating System, Time of Introduction (development project phase), Consequence Scope, and Consequence or Technical Impact, which were supplied with the native CWE data. The application represented these filters with drop-down boxes along the top of the HTML page. We discovered that these filtering criteria had limited value in relating individual weaknesses to one another, or to functional aspects of a software design problem. A more interesting result was achieved by using the text search field to organize subsets of weaknesses based on common terms. **Table 1** shows the results of basic text searches of the CWE data. The search argument "crypt" provides the union of "crypto" and "encryption," plus the search arguments "entropy" and "algorithm" provide valuable insight into the problem space of common usage for cryptography; however, the results are almost totally disconnected from the query space of cryptography and encryption. Taken together, and combined with results from related searches for "certificate" and Secure Sockets Layer (SSL), the search results begin to provide a narrative for consistent guidance on standard uses of cryptography, along with practices to be avoided in design, coding, and integration of modern systems and applications.

The project team worked with several IBM development teams to apply the methods, reference materials and the output from the prototype tool in order to determine whether we could satisfy the requirements listed earlier. We learned that there is a significant concept and terminology gap between security-oriented subject-matter experts and other development roles, including architects, programmers, build specialists and testers. We also learned that for large, geographically diverse developer populations, a centralized application service for threat analysis is a requirement for executive reporting. We also recognized the mixed value of hardcopy threat analysis reports. These documents have value as project artifacts for traceability, but have limited value as input to downstream projects in a development supply chain. Our experiments helped us to identify three additional requirements:

Requirement 7—Develop a bridge between the concepts and terminologies used by Security Subject-Matter Experts and software development practitioners.

Requirement 8—Threat analysis is not an isolated activity that documents the issues and recommendations for software assurance. Instead, threat analysis is an activity that, when most effective, organizes and orchestrates the development project, feeding Security Requirements, Secure Coding, and Security Testing efforts.

Requirement 9—Electronic copies of the input data can be analyzed for purposes of quality control and reused by other teams.

These observations led the team to create extensions to the prototype tool with the goal of providing customized views and reports on Software Weaknesses from the CWE

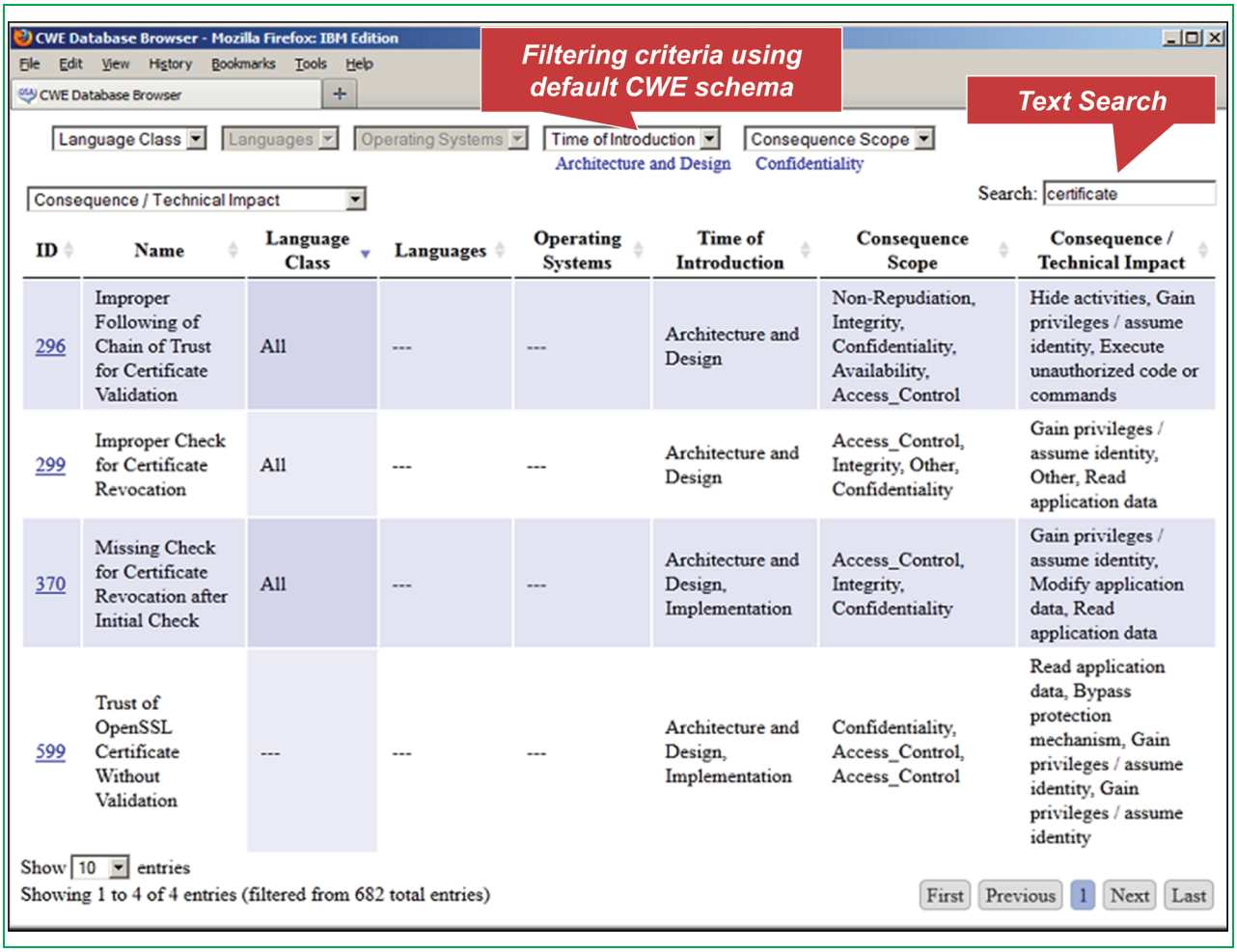

Figure 1

First prototype search tool for Common Weakness Enumeration knowledge base.

database. The resulting tool and process provides what we believe is a suitable and sustainable foundation for the Software aspects of threat analysis in the IBM Software Development process. Furthermore, conversations with development executives and technical leaders from several enterprise customers confirm interest in this approach.

Secure Engineering Design Assistant

The Secure Engineering Design Assistant (SEDA) resulted from the threat analysis prototype activity, with the goal of addressing the requirements described previously and creating a platform for further development. The concept was to mirror the tasks in a standard consultative threat analysis with automation, using the CWE database as a security knowledge base for the Software-centric perspective of threat analysis in **Figure 2**. Implicit in this flow is a collaboration across the development team, where leaders of the threat analysis activity identify a work plan and delegate the mitigation—including design, coding, and integration tasks—to the appropriate members of the project team.

The first working model of the SEDA tool had a single interface designed for end-users. As shown in **Figure 3**, the user service was patterned after a shopping cart, with the existing knowledge base of issues as the inventory. The user's task was to apply filters to the inventory by defining the project parameters, transferring the prescribed security issues to the shopping cart, and proceeding to checkout when the filtering process is complete. The "checkout process" included steps for reviewing and updating the issues list, assigning the issues to one or more roles in the development process, and finally, saving the output and creating the role-specific documentation, with a project level document that included all roles and issues to be resolved. This application provided a way for architects to search for and accumulate CWE entries relevant

Table 1 Example text string searches of unstructured Common Weakness Enumeration (CWE) data. (J2EE**: Java** 2 Platform, Enterprise Edition; RSA: Rivest, Shamir, Adleman.)

Search argument	*Search result*
crypt	• CWE 261 – Weak Cryptography for Passwords
	• CWE 311 – Missing Encryption of Sensitive Data
	• CWE 321 – Use of Hard-Coded Cryptographic Key
	• CWE 323 – Reusing a Nonce, Key Pair in Encryption
	• CWE 325 – Missing Required Cryptographic Step
	• CWE 326 – Inadequate Encryption Strength
	• CWE 327 – Use of a Broken or Risky Cryptographic Algorithm
	• CWE 338 – Use of a Cryptographically weak PRNG (Pseudo-Random Number Generator)
	• CWE 347 – Improper Verification of Cryptographic Signature
	• CWE 005 – J2EE Misconfiguration: Data Transmission without Encryption
	• CWE 649 – Reliance on Obfuscation or encryption without Integrity Checking
algorithm	• CWE 303 – Incorrect Implementation of Authentication Algorithm
	• CWE 327 – Use of a Broken or Risky Cryptographic Algorithm
	• CWE 407 – Algorithmic Complexity
	• CWE 757 – Selection of Less-Secure Algorithm during Negotiation
	• CWE 780 – Use of RSA Algorithm without OAEP (Optimal Asymmetric Encryption Padding)
entropy	• CWE 331 – Insufficient Entropy
	• CWE 332 – Insufficient Entropy in PRNG (Pseudo-Random Number Generator)
	• CWE 333 – Improper Handling of Insufficient Entropy in TRNG (True Random Number Generators)

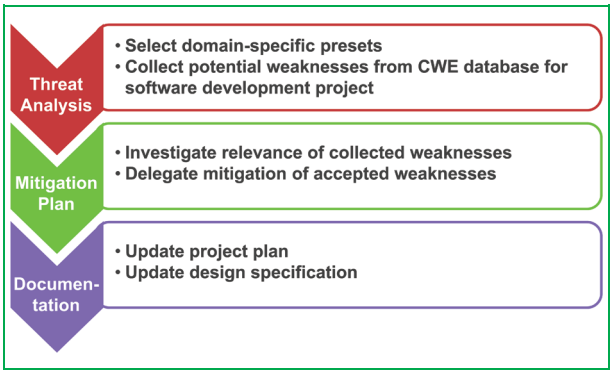

Figure 2

Secure Engineering Design Assistant (SEDA) tool: basic workflow.

to their project. The workflow, and the ability to build a profile for the software, was an improvement; however, the results were limited by the expertise and persistence of the user.

To address Requirement #7, the second iteration of the SEDA application included a separate service for administrators. As shown in **Figure 4**, the tagging service supported the creation and assignment of natural language tags to entries in the CWE database, providing a means to compensate for the unstructured text in the native CWE database. The administrator manually associates one or more natural language tags with selected CWE entries. Once entered, the tags are available to all users as a means to build the Software-centric portion of their threat analysis. The tags reflected common terminology and computing functions that were familiar to developers.

The resulting list included entries that map common taxonomies mentioned earlier in this paper. By filtering the Common Weakness Knowledge base by the tag values, the tool produced output for familiar taxonomies, such as OWASP Top 10, SANS Top 25 dangerous software errors, and STRIDE. Beyond these taxonomies, tags were created for groupings of CWE entries mapped to the following concepts and functions: application programming interface (API), authentication, authorization, confidentiality, cookie, cryptography concepts, digital certificates, exception handling, expiration, hash, PHP (hypertext preprocessor), identity, information disclosure, information protection, integrity, log files, malware, password, permission, privacy, privilege, salt, session, socket, and storage. The tagging application allowed the administrator to group sets of security concerns that were otherwise unrelated in the CWE data. These extended groupings of issues were then aggregated to create broader software profiles. For example, a "database system" tag can be associated with a set of potential security issues related to "storage" and "SQL (Structured Query Language) Injection," "authentication," "authorization," and "privilege."

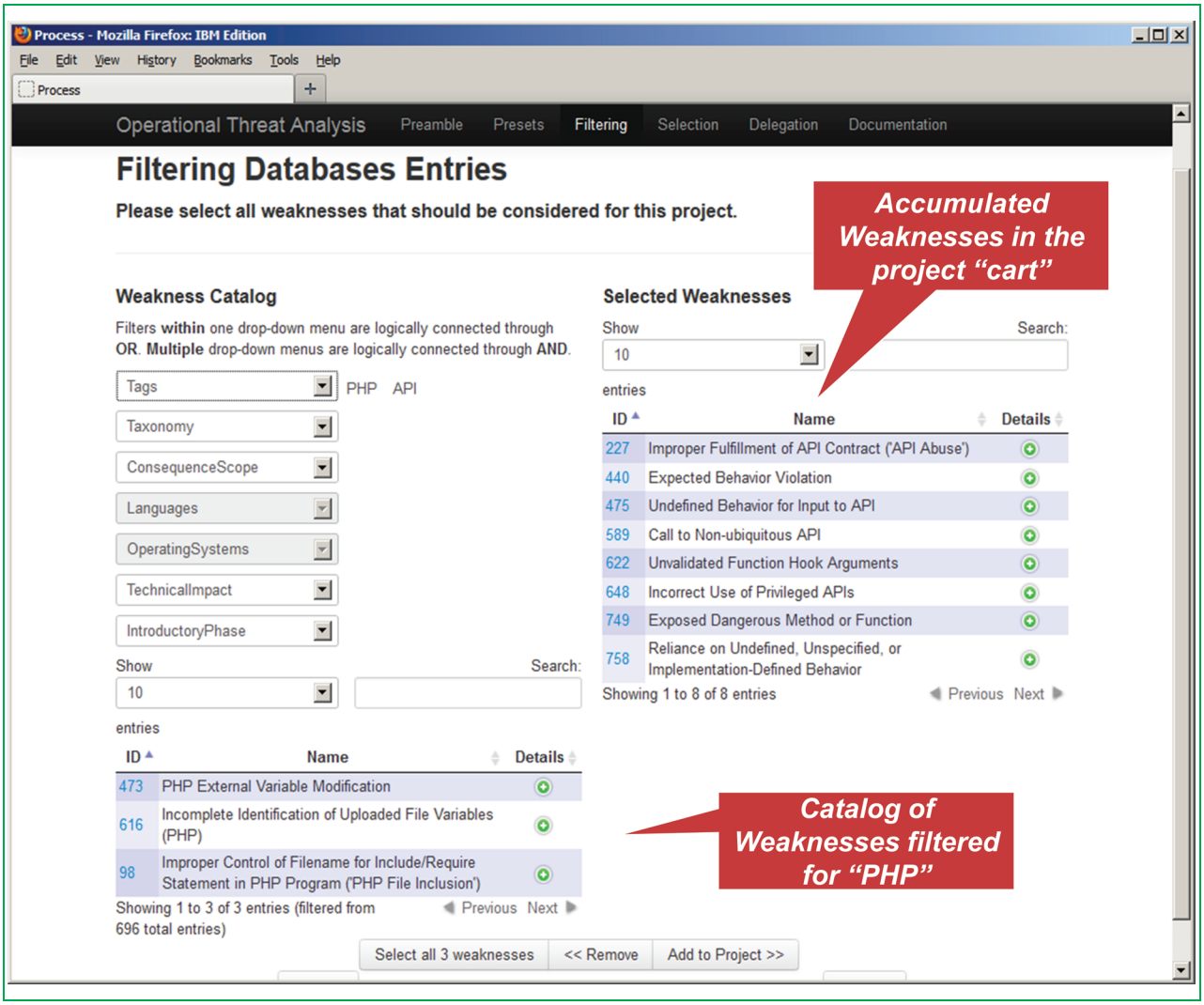

Figure 3

Secure Engineering Design Assistant (SEDA) tool: user interface for filtering task.

The output of the SEDA tool provided some interesting insights about the relationships between CWE entries when they are grouped by concept or by function. We were able to visualize how individual CWE entries are brought into scope—or potentially missed—when different architectural perspectives are applied. We could observe, for example, that a focus on the functional view of software naturally missed many of the weaknesses called out by the OWASP or SANS recommendations. In addition, we determined that the SANS Top 25 issues are a subset of a larger set of more than 65 related weaknesses in categories that we defined as User and Access Management, Privilege Management, Passwords, Privacy and Confidentiality, System Configuration, Middleware Configuration, and Application Coding. We also recognized that important categories and weaknesses were not covered by default, including those related to system integrity, information exposure, cryptography, certificates, storage, session management, and cookies. These insights led to a combined set of more than 130 data points for weaknesses. This may sound unwieldy; however, when grouped, they become part of 16 high-level development requirements that can be reasonably assigned, tracked, and managed. We refer to the second iteration of the tool as SEDA.

SEDA basic operation

The user service in SEDA embodies the following six steps: Preamble, Presets, Filtering, Selection, Delegation, and Documentation. The intention is to create a project that

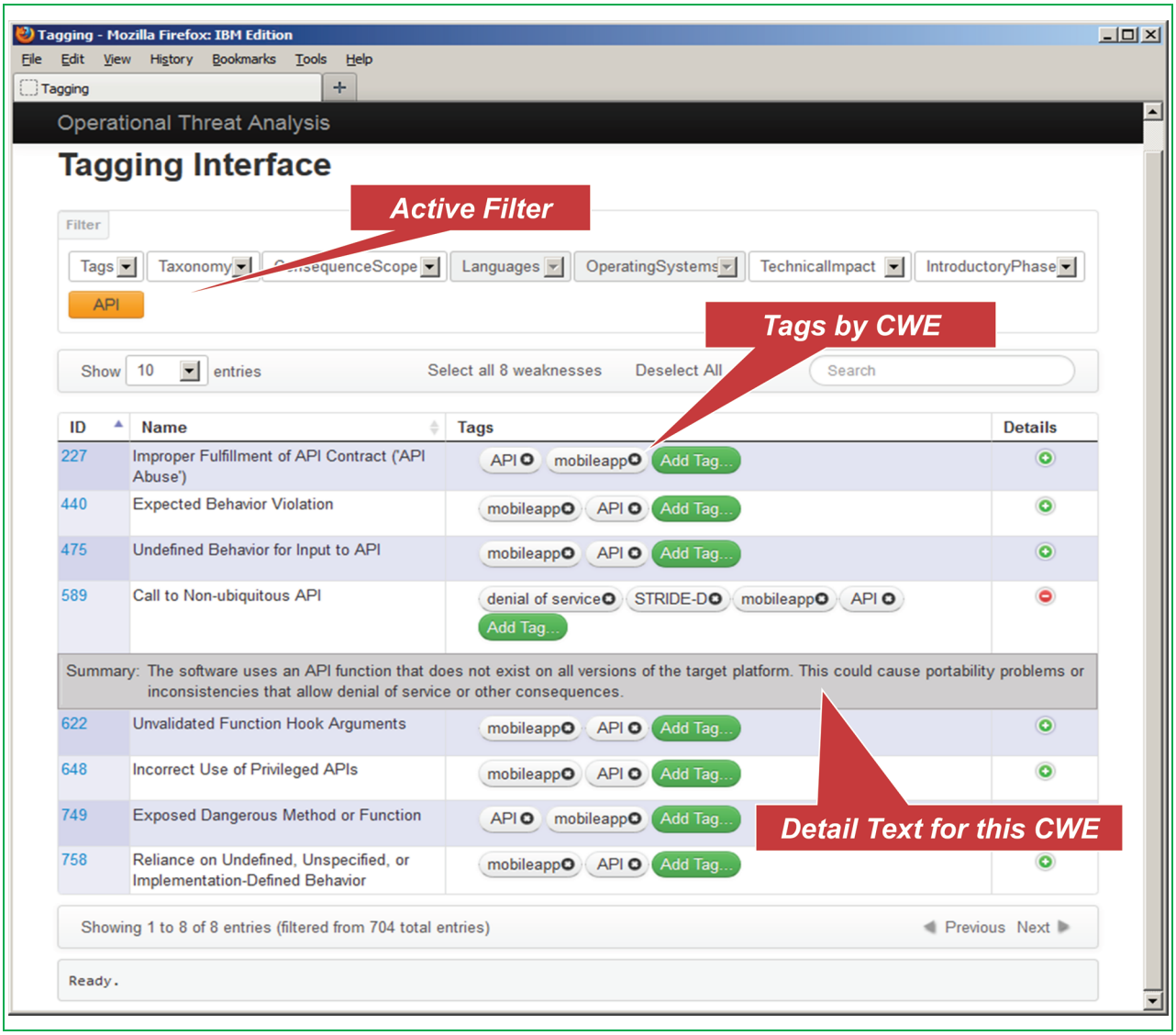

Figure 4

Secure Engineering Design Assistant Version 1 tool: administrator interface for tagging. (API: application programming interface.)

is stored on a server and that can be changed or updated at any time during the development process. We do not want to interfere with the current work paradigms or processes of the developers, but rather support them with the tool. Thus, our six-step process is not mandatory; it is just a recommendation that the developers can override as their workflow requires. The tool is implemented in HTML and JavaScript, which serves as a graphical user interface to the backend database, which currently contains the CWE database but can be extended further with custom databases.

In the preamble step the user can describe the project briefly with a name, version, and application domain. This helps to identify the project and serves as a "cover page" for the resulting documentation.

Presets are topics created by security experts that serve as a marker for relevant database entries for these topics. The purpose of these presets is to give developers with little security background a point from which to start. For example, one could create a preset "web app" that automatically marks all database entries relevant for cross-site-scripting, SQL Injection, and other pertinent design and coding issues.

Filtering is the main step of the tool. It offers a view of the database along with filters to search the database. Developers can search the database via categories such as

Technical Impact" [i.e., gain privileges, DOS (denial of service), etc.], tags created by security experts, or a free text input. Each database entry contains a brief description, and the developer then must decide whether it is relevant for the project at hand. If an entry is relevant, the user adds it to the project. The user can also choose to add all entries of a Preset. An example is shown in Figure 3. In this example, the user had previously selected the tags "PHP" and "API," leading to the three results in the table on the left hand side (entries with IDs: 473, 616, and, 98). The project cart contains the entries with CWE IDs 227, 440, 475,589, 62, 648, 749, 758, and this is shown in the table on the right-hand side.

The Selection step is used to refine the list, focusing on those entries that are relevant to the software that is built in the project. Irrelevant entries can be marked as such, and the developer must give a reason why he or she deems an entry irrelevant. This is important for both the security documentation of the project and for other developers working on the project. Selections that are relevant for the project can be delegated to the developer roles (Designers, Programmers, and Information Planners) in the Delegation step. Each entry can be assigned to one or more roles in the development team. This assignment will result in creation of documentation that describes the issue and provides information and references for mitigation.

The last step provides for automatic generation of documentation and retention of the threat analysis. This step satisfies Requirements #8 and #9. One can choose to either create the documentation for just one role or for all roles, as mentioned above. The documentation in HTML format contains all the database entries added to the project together with all instruction and reason texts.

SEDA tool adoption

The SEDA tool provided an environment to experiment with end user service and to populate the tagging interface. We created a number of tests and built a number of sample reports for commonly occurring sets of issues and software configurations. These reports included Basic Software, with an emphasis on the SANS Top 25 issues; Mobile Applications, with an emphasis on APIs, privacy, and protected storage; Web Applications encompassing the OWASP Top 10, User Management, and Storage security; and Complex Integrated Offering, with more than 150 system and application security issues.

With predefined tags, the reports required only a few minutes to create. We found that the most tedious part of the process was involved with importing the basic HTML into a standard word-processing document and transforming the raw output into a form with Front Matter, Headings, Table of Contents, and Customized Guidance that made the document useable as the starting point for a more formal project artifact. This effort revealed the need for more work, especially in the area of usability, output organization, and report formatting.

The lessons learned about formatting the tool output, the implications of unintentional biasing, and the need to ensure that a minimum core set of issues are considered in-scope led us away from our initial design for a self-service tool toward the decision to provide a set of project-ready, customizable, threat-analysis templates, with the option for teams to engage a security subject-matter expert for a tailored threat analysis documents.

Findings

In our research phase, we determined that the average time required for a security consultant to perform a complete threat analysis was on the order of 160 hours, or four person-weeks. As we began the development phase, we set an objective that an architect or software engineering with between five and ten years of design experience could complete the basic elements of the Software-centric portion of a threat analysis within approximately 16 hours. This goal was not achieved for all users due to the limitations in usability of the SEDA. We did find that this was a reasonable goal for teams that begin with one of the standard threat-analysis templates that were built. Also, when a custom template is needed, a subject-matter expert-assisted threat-analysis activity using the SEDA tool can be accomplished in less than the 16-hour target established for our process.

Working directly with development, we determined that additional guidance was needed to ensure that appropriate rigor was applied to scoping the analysis of threats. This led us to make two amendments to the overall method by (1) asserting a number of risk scenarios that provided context for the threat analysis process, and (2) changing the perspectives for threat analysis.

Development teams need a reference point from which to form opinions about risks and their relevance. We documented a set of risk scenarios as a starting point. The default list of risk scenarios are as follows: (1) vulnerabilities or weaknesses in supply chain components, (2) vulnerabilities or weaknesses in developed code, (3) vulnerabilities or weaknesses in operational systems, (4) non-compliance with policies, (5) compromise of information assets, (6) cybersecurity attack, and (7) insider attack. Architects and designers use these as the basis for architectural decisions, determining what is in-scope and out-of-scope, and assigning a priority to the in-scope issues.

Every development project is expected to consider risk scenarios 1, 2, and 3. Projects that create complex software are expected to additionally consider scenarios 4 through 7. Once the scenarios are selected and the priority set, then the practitioner builds a list of threat vectors that are relevant to each in-scope risk scenario.

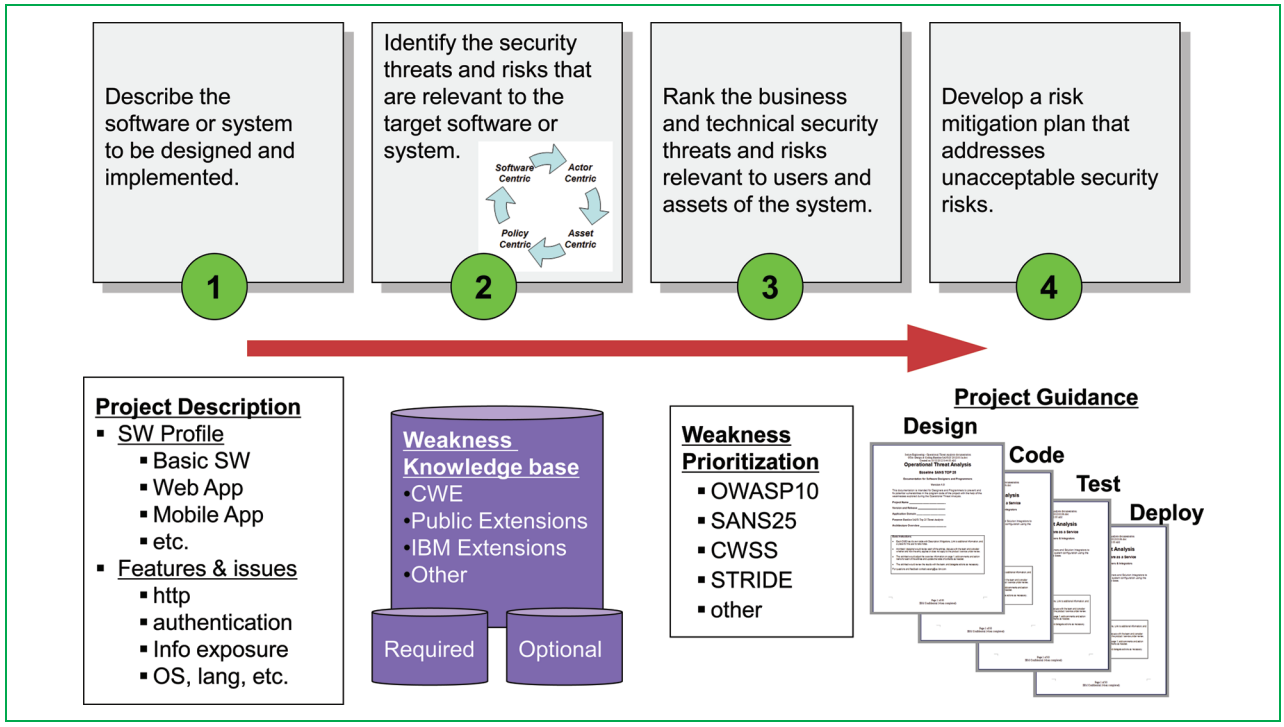

Figure 5

Threat modeling process flow with an automated design assistant. (SW: software; App: Application; RBAC: role-based access control; OS: operating system; lang: language; CWSS: Common Weakness Scoring System.)

The perspectives developed earlier allowed development teams to customize their consideration of threats to the type of software being developed. We found that renaming the attacker-centric view to the actor-centric view encouraged teams to recognize that both normal use cases and abuse cases may contribute to security weaknesses. Furthermore, we determined that it was important to recognize policy as a security aspect of design. Adding a policy-centric perspective provides a means for development teams to consider the applicability of standards or requirements frameworks, such as defined for the finance industry and healthcare industry.

The expanded context for threat analysis within IBM Secure Engineering includes the following:

Software-centric—How does the software design, architecture, or implementation allow or support harmful attacks?

Actor-centric—Who uses the software? How is the software used? How might attackers compromise the software, system, or service?

Asset-centric—What information assets are created, stored, updated, or used, and how might they be compromised?

Policy-centric—This view accounts for enterprise risks associated with non-compliance with internal policies, external regulations, and missed service levels.

Finally, we are conducting experiments to augment the default CWE data with security-related information from other sources, as well as applications of analytics as a means to build a broader knowledge-base and more in-depth treatment of risks and threats.

Conclusion

Threat analysis is a more complex activity than can be achieved through the use of a simple tool. However, the results of our investigation suggest that there is sufficient accumulated knowledge about software-related threats and weaknesses to provide a foundation for knowledge-based threat analysis, which is in contrast to consultative approaches that require security experts on every development team. Our approach also provides the seed for further research and development.

The documented threat-analysis process provides a consistent structure for all development teams to follow to ensure a focus on security early in the development process (**Figure 5**). The approach of expanding the role of architects and designers to include a structured threat-analysis activity is providing value to IBM by encouraging a consistent focus on security throughout the development. The SEDA tool supports the architect's role in articulating the software-centric view of threat

analysis by providing insight on avoiding potential weaknesses and vulnerabilities in design, coding, and build-integration.

In the past several years, our team fielded an increasing number of requests from businesses, government agencies, and industry groups for evidence of assurance for software, computing systems, and computing services. The criteria for assurance varied from request to request. The lack of a standard of performance for secure development leaves all parties with heightened risk exposure: businesses and governments sensing exposure to attack, and information technology providers sensing exposure of intellectual property.

One path forward may be an initiative in the Open Group [32] that is jointly driven by business, government, and IT providers. The Open Trusted Technology Framework (OTTF) Working Group [33] is establishing a standard and an accreditation process for ICT (Information and Communication Technology) providers to demonstrate their ability to follow accepted practices in design, build, deliver, and support of their products and services.

Acknowledgments

We thank the following individuals for their participation, guidance, and assistance in this project and in writing this paper: Andreas Fuchs, Jan Steffan, John Lucassen, Lukas Kalabis, Marco Ghiglieri, Mechthild Stöwer, and Michael Waidner.

References

1. U.S. Department of Defense. (1985). U.S. Department of Defense trusted computer system evaluation criteria, Washington, DC, USA, DoD 5200.28-STD. [Online]. Available: http://csrc.nist.gov/publications/history/dod85.pdf
2. *National Information Assurance (IA) Glossary*, Committee on National Security Systems, Ford Meade, MD, USA, 2010. [Online]. Available: http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf
3. M. Howard and S. Lipner, *The Security Development Lifecycle*. Redmond, WA, USA: Microsoft Press, 2006.
4. *Building Security in Maturity Model*. [Online]. Available: http://www.bsimm.com
5. *Security in Development: IBM Secure Engineering Framework*, IBM Corporation, Armonk, NY, USA. [Online]. Available: http://www.ibm.com/security/secure-engineering/
6. *IBM z/OS Statement of Integrity*, IBM Corporation, Armonk, NY, USA. [Online]. Available: http://www.ibm.com/systems/z/os/zos/features/racf/zos_integrity_statement.html
7. *IBM z/VM System Integrity Statement*, IBM Corporation, Armonk, NY, USA. [Online]. Available: http://www.vm.ibm.com/security/zvminteg.html
8. IBM Corporation. (2001). IBM highlights 1990–1995, Armonk, NY, USA. [Online]. Available: http://www-03.ibm.com/ibm/history/documents/pdf/1990-1995.pdf
9. D. Allan, T. Hahn, A. Szakal, J. Whitmore, and A. Buecker, *Security in Engineering: IBM Secure Engineering Framework*, IBM Corporation, Armonk, NY, USA, pp. 1–22. [Online]. Available: http://www.redbooks.ibm.com/redpapers/pdfs/redp4641.pdf
10. J. Steven, "Threat modeling–Perhaps it's time," *IEEE Security Privacy*, vol. 8, no. 3, pp. 83–86, May/Jun. 2010.
11. *The STRIDE Threat Model*. [Online]. Available: http://msdn.microsoft.com/en-us/library/ee823878%28v=cs.20%29.aspx
12. M. S. Lund, B. Solhaug, and K. Stlen, *Model-Driven Risk Analysis–The CORAS Approach*. Berlin, Germany: Springer-Verlag, 2011.
13. A. Steinberg, "Open interaction network model for recognizing and predicting threat events," in *Proc. Inf. Decision Control*, 2007, pp. 285–290.
14. X. Li and K. He, "A unified threat model for assessing threat in web applications," in *Proc. Int. Conf. Inf. Sec. Assurance*, Apr. 2008, pp. 142–145.
15. P. H. Meland and J. Jensen, "Secure software design in practice," in *Proc. 3rd Int. Conf. ARES*, Mar. 4–7, 2008, pp. 1164–1171.
16. D. Dhillon, "Developer-driven threat modeling: Lessons learned in the trenches," *IEEE Security Privacy*, vol. 9, no. 4, pp. 41–47, Jul./Aug. 2011.
17. *Application Threat Modeling*. [Online]. Available: http://msdn2.microsoft.com/en-us/security/aa570413.aspx
18. Y. Chen, B. Boehm, and L. Sheppard, "Value driven security threat modeling based on attack path analysis," in *Proc. 40th Annu. HICSS*, Jan. 2007, pp. 280a–288a.
19. S. Al-Fedaghi and A. A. Alrashed, "Threat risk modeling," in *Proc. 2nd ICCSN*, Feb. 26–28, 2010, pp. 405–411.
20. A. S. Sodiya, S. A. Onashoga, and B. A. Oladunjoye, "Threat modeling using fuzzy logic paradigm," *Inf. Sci., Int. J. Emerging Transdisc.*, vol. 4, no. 1, pp. 53–61, Jan. 2007.
21. D. Xu and K. Nygard, "A threat-driven approach to modeling and verifying secure software," in *Proc. 20th IEEE/ACM Int. Conf. ASE*, 2005, pp. 342–346.
22. L. Hoffman, D. Burley, and C. Toregas, "Holistically building the cyberSecurity workforce," *IEEE Security Privacy*, vol. 10, no. 2, pp. 33–39, Mar./Apr. 2012.
23. M. Kwon, M. Jacobs, D. Cullinane, C. Ipsenand, and J. Foley, "Educating cyber professionals: A view from academia, the private sector, and government," *IEEE Security Privacy*, vol. 10, no. 2, pp. 50–53, Mar./Apr. 2012.
24. *IBM X-Force Analysis Reports*, IBM Corporation, Armonk, NY, USA. [Online]. Available: http://www.ibm.com/security/xforce/downloads.html
25. *Open Web Application Security Project–Top 10 Project*. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
26. *SANS Institute Top 25 Most Dangerous Software Errors*. [Online]. Available: http://www.sans.org/top25-software-errors/
27. *Common Weakness Enumeration*. [Online]. Available: http://cwe.mitre.org/
28. *National Vulnerability Database*. [Online]. Available: http://nvd.nist.gov/
29. Fraunhofer SIT, Darmstadt, Germany. [Online]. Available: https://www.sit.fraunhofer.de/en/the-institute/fraunhofer-sit/profile/
30. Technische Universität Darmstadt, Darmstadt, Germany. [Online]. Available: http://www.tu-darmstadt.de/index.en.jsp
31. *OCTAVE Operationally Critical Threat, Asset and Vulnerability Evaluation*. [Online]. Available: http://www.cert.org/octave/
32. The Open Group, San Francisco, CA, USA. [Online]. Available: http://www.opengroup.org/
33. *The Open Group Trusted Technology Provider Framework*. [Online]. Available: https://www2.opengroup.org/ogsys/catalog/W113

Received March 20, 2013; accepted for publication April 21, 2013

Jim Whitmore *IBM Software Group, Mechanicsburg, PA 17011 USA (whitmore@us.ibm.com)*. Mr. Whitmore is an Open Group Distinguished Architect and Technical Leader for the Secure Engineering Initiative in IBM. He holds a B.S. degree in electrical engineering from University of Maryland and an M.S. degree from University of Maryland, University College. Mr. Whitmore joined IBM in 1984 with prior experience in computer system and software development. At IBM, he worked as a subject-matter expert in data communications, security design and integration, self-protecting autonomic systems, security architecture, and secure development. His writings have been published in the *IBM Systems Journal* and IEEE Conference Proceedings.

Sven Türpe *Fraunhofer Institute for Secure Information Technology, Security Test Lab, 64295 Darmstadt, Germany (sven.tuerpe@sit.fraunhofer.de)*. Mr. Türpe's research interests include application security, security evaluation and testing, and threat analysis. He has a Diploma in computer science from Universität Leipzig.

Stefan Triller *Fraunhofer Institute for Secure Information Technology, Security Test Lab, 64295 Darmstadt, Germany (stefan.triller@sit.fraunhofer.de)*. Mr. Triller's research focus is security testing and secure software engineering. Regarding software engineering, he was part of the Middle Size League RoboCup team at the University of Kassel and developed a behavior engine to achieve his Master's degree.

Andreas Poller *Fraunhofer Institute for Secure Information Technology, Security Test Lab, 64295 Darmstadt, Germany (andreas.poller@sit.fraunhofer.de)*. Mr. Poller's research focuses on privacy protection, especially in Web 2.0 applications, and empirical secure software engineering. Mr. Poller has a Diploma in applied computer science from the Chemnitz University of Technology.

Christina Carlson *IBM Software Group, Minneapolis, MN 55402 USA (chrisrc@us.ibm.com)*. Ms. Carlson is the Lead Developer for the IBM Secure Engineering Tooling Team. She has a B.S. degree in chemistry from Olivet Nazarene University and an M.S. degree in chemical engineering from University of Illinois at Urbana-Champaign. She has performed spaceflight instrumentation testing and characterization for multiple spaceflight mass spectrometers, including those that flew on the Cassini mission to Saturn. She joined IBM in 2000 and has worked since then in software quality and test, software development methodology, test automation, and secure development.