SD Elements: A Tool for Secure Application Development Management

Golnaz Elahi[1], Tom Aratyn[2], Ramanan Sivaranjan[2], Rohit Sethi[2], and Eric Yu[3]

[1] Department of Computer Science, University of Toronto, Canada, M5S 1A4
`gelahi@cs.toronto.edu`
[2] SD Elements, Toronto, Canada
`Tom,Ramanan,Rohit@sdelements.com`
[3] Faculty of Information, University of Toronto, Canada, M5S 3G6
`eric.yu@utoronto.ca`

Abstract. A major problem in achieving security goals in application development is the overwhelming amount of security-related information, variety of tools, and numerous security risks and vulnerabilities. Software analysts, developers, and testers are not often able to identify relevant security knowledge. Many security tools focus only on detecting vulnerabilities, but the embedded available security guidelines are usually not directly auditable. To fill these gaps, we introduce a new tool, called SD Elements, which focuses on prevention of vulnerabilities as opposed to detection. SD Elements is a centralized security knowledge base that covers different development life cycle phases, so security is built into the application from the early phases of the life cycle. Users are able to specify technologies, platforms, requirements, and programming languages, and SD Elements tailor security guidelines for different projects according to the user specifications. It enables businesses to provide tangible security audit evidence and trace compliance with security standards. The tool is currently being beta tested in varieties of firms, by different roles, and in different development phases.

Keywords: Application security, security requirements, development guidelines, security knowledge, test case.

1 Introduction

The software engineering community is slowly beginning to realize that information security is also important for software whose primary function is not related to security [1]. Since prevention is often more economical than remediation, empirical security knowledge such as common attacks and vulnerabilities are made public and available for practitioners through web-based portals such as NVD [2], CWE [3], OWASP [4]. Security standards, such as PCI DSS [5] or ISO [6] provide high level guidelines and impose several compliance requirements to application developers.

In reality, software analysts, developers, and testers are overwhelmed with the amount of available information and variety of tools they can employ. Analysts are given massive lists of security requirements, guidelines, and standards, and they need to provide tangible and audit-able evidence that the products comply with security guidelines. Employing tools can help application testers. However, security testing tools are usually intimidating and their adoption rate is low, specially because application developers and testers are not often security experts. Testers also need to tailor the testing scripts for the platforms and technologies that they use.

The bottom line for practitioners is finding the relevant body of information to their projects. However, there is a significant gap between the existing body of empirical knowledge collected in (web-based) knowledge portals and actual development demands.

1.1 Contributions

To fill the current gaps, we introduce a secure application development management tool, called SD Elements, which provides a set of core values to application developers, system analysts, and quality assurance teams.

SD Elements is a web-based knowledge repository of security guidelines, empowered by a retrieval tool. SD Elements surveys users to learn about the nature of the project, platform, language, and technologies, and then it tailors security knowledge:

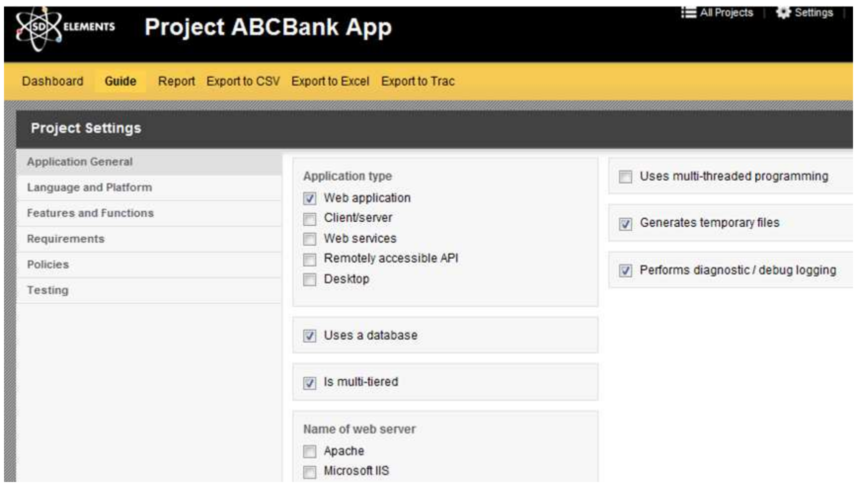
- Generates relevant security requirements.
- Provides tailored guidelines on secure architecture design.
- Provides reusable development standards for different development platforms and technologies.
- Provides sample tested code for implementing the standards.
- Creates a list of security test cases (and check lists) to enable non-expert developers systematically test security requirements.
- Integrates into application life cycle management and bug tracking tools such as Quality Center and trac.
- Ranks the risks related to standards which helps with prioritization of development guidelines and security requirements.

SD Elements focuses on vulnerability prevention instead of detection. It integrates security knowledge into the development life cycle, thus, security is built into the application from the early phases. SD Elements provides a compliance mechanism, i.e., users can trace which guidelines are employed, implemented, and tested. This provides businesses with tangible and traceable evidence for audit purposes. Finally, it provides requirements, implementation, and testing guidelines, in situations that compliance with PCI DSS and HIPPA is needed.

2 SD Elements Architecture

SD Elements users will be software developers such as requirements analysts, programmers, testers, project leaders, and security analysts. For each project, SD Elements surveys the developers about the nature of the project, security features and users of the application, types information being handled by the application, business drivers and policies, platforms, technologies, programming languages, and application interfaces. By collecting these information about the project, SD Elements retrieves a customized set of guidelines and requirements, appropriate for specific projects.

For example, application general questions (Figure 1) uncover the type of application, type of web server, programming languages, platforms, third party technologies and libraries. The survey also enables users to provide more details about the features and functions of the project. For example, developers can specify whether the application being developed involves interactions with operating system, file-upload function, authentication of end users, etc.



Project ABCBank App

Project Settings

Application type
- [✓] Web application
- [ ] Client/server
- [ ] Web services
- [ ] Remotely accessible API
- [ ] Desktop

- [✓] Uses a database
- [✓] Is multi-tiered

Name of web server
- [ ] Apache
- [ ] Microsoft IIS

- [ ] Uses multi-threaded programming
- [✓] Generates temporary files
- [✓] Performs diagnostic / debug logging

Fig. 1. SD Elements Survey (application general questions)

The answer to the questions enable the tool to refine the rest of questions as well as retrieve relevant security guidelines and requirements. These guidelines in the SD elements knowledge base are developed according to:

– Current vulnerabilities for different technologies, platforms, and languages. Each guideline or requirement corresponds to a vulnerability in Common Weakness Enumeration list of vulnerabilities [3].
– Best practices and existing standards such as OWASP [4], WASC threat classification [7]; empirical data about commonly-exploitable applications in web applications based on years of penetration testing; threat models, and source code review; and regulatory compliance including PCI DSS, HIPPA HITECH, GLBA, NERC CIP, and international privacy laws.

2.1 Knowledge Retrieval Engine

SD Elements' knowledge storage and retrieval is based on Boolean logic. The knowledge retrieval engine provides security guidelines, based on what users has specified. For example, if users select a J2EE project, then SD Elements concludes the user will be developing a web application that is probably multi-tiered, etc. Thus it does not require overwhelming efforts for users and project owners to describe the nature of project for receiving useful information. Project managers can get security guidelines as well, without knowing much of technical details.

2.2 SD Elements Knowledge Base Architecture

Figure 2 depicts a high level overview of the SD Elements' knowledge base architecture. The contents of the knowledge base are vulnerabilities, security standards, and implementation of the standards. By answering the survey questions, a set of properties about the project are gathered. Each property entails a set of content. The tool is implemented in Django which allows creating models to generate the data base schemas.

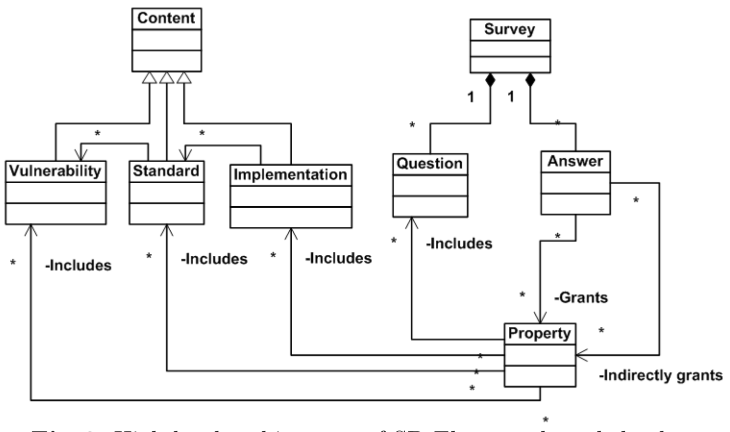


Fig. 2. High level architecture of SD Elements knowledge base

3 Integration with Software Development Life Cycle

SD Elements helps in management of secure application development. It facilitates building security into the application, from the early requirements stage, and considering security in mind at the design and implementation activities. Finally, it provides step-by-step test cases to help non-security experts test relevant cases to their application.

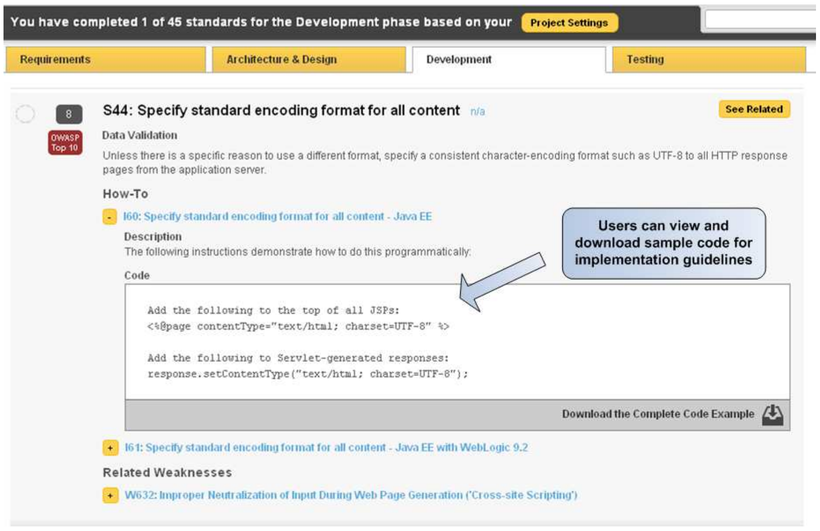
3.1 Requirements Generation

SD Elements supports application security requirements analysis:

– Surveys analysts about the project settings.
– Generates a list of relevant security requirements (in different categories) which are tailored with respect to the project settings.
– Links generated requirements to relevant vulnerabilities.
– Links the generated requirements to a design/implementation guidelines and test cases.
– Lists the requirements criteria of acceptance. These criteria are actually the test cases. Thus, analysts will know from the early stages, how the requirements will be tested.

3.2 Design and Development Guidelines

SD Elements provides project-specific implementation guidelines. Suggested guidelines correspond to the list of security requirements and settings of the project. Sample (and tested) code, whenever applicable, is provided as part of the content. For example, for the HTML encoding for JSPs, SD Elements provide a sample code as depicted in Figure 3. Figure 4 shows an example list of development guidelines in the authentication category.


You have completed 1 of 45 standards for the Development phase based on your Project Settings

Requirements | Architecture & Design | Development | Testing

8 OWASP Top 10

S44: Specify standard encoding format for all content n/a

See Related

Data Validation

Unless there is a specific reason to use a different format, specify a consistent character-encoding format such as UTF-8 to all HTTP response pages from the application server.

How-To

- I60: Specify standard encoding format for all content - Java EE

Description

The following instructions demonstrate how to do this programmatically:

Code

```
Add the following to the top of all JSPs:
<%@page contentType="text/html; charset=UTF-8" %>

Add the following to Servlet-generated responses:
response.setContentType("text/html; charset=UTF-8");
```

Download the Complete Code Example

+ I61: Specify standard encoding format for all content - Java EE with WebLogic 9.2

Related Weaknesses

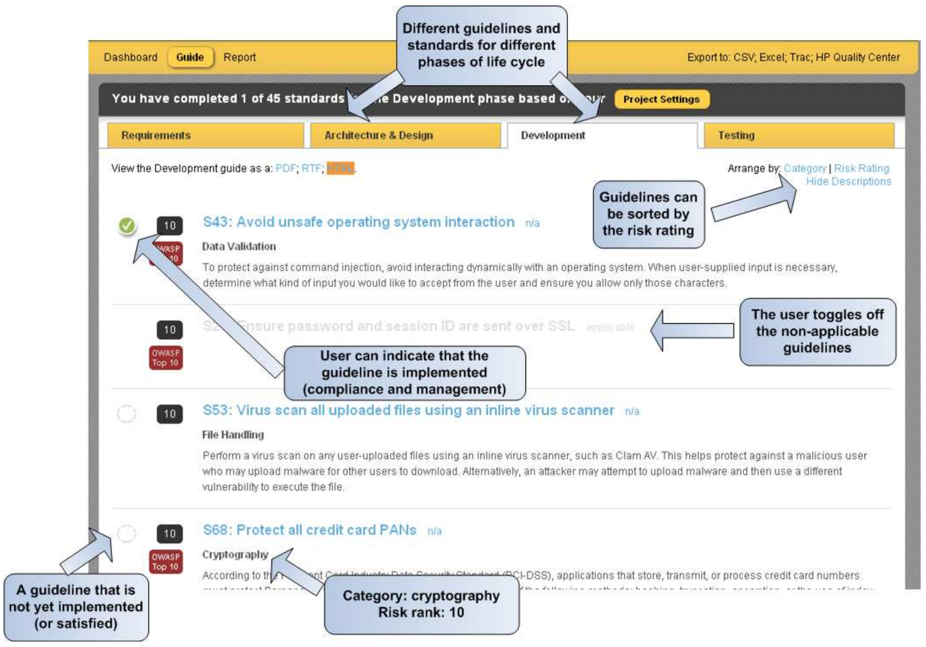
+ W632: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Users can view and download sample code for implementation guidelines

Fig. 3. Standard encoding format, sample code

3.3 Application Testing

SD Elements generates step by step test cases, that include failure conditions, sample scripts (if needed), guidelines about testing tools, and guiding videos. By the time the users get to the testing phase, SD Elements has provided proper security requirements and implementation guidelines. Thus, although test cases are provided, the emphasis is on preventing vulnerabilities instead of detection. Test cases are linked to security requirements, and user can specify a test is passed. Then, the requirements status is changed (to satisfied) as well. Figure 5 shows a screen shot of a sample test case.


Different guidelines and standards for different phases of life cycle
Guidelines can be sorted by the risk rating
The user toggles off the non-applicable guidelines
User can indicate that the guideline is implemented (compliance and management)
A guideline that is not yet implemented (or satisfied)
Category: cryptography Risk rank: 10
Dashboard
Guide
Report
Export to: CSV; Excel; Trac; HP Quality Center
Project Settings
Requirements
Architecture & Design
Development
Testing
OWASP Top 10

You have completed 1 of 45 standards e Development phase based o ur

View the Development guide as a: PDF; RTF; HTML.

Arrange by: Category | Risk Rating
Hide Descriptions

10 S43: Avoid unsafe operating system interaction n/a

Data Validation

To protect against command injection, avoid interacting dynamically with an operating system. When user-supplied input is necessary, determine what kind of input you would like to accept from the user and ensure you allow only those characters.

10 S2 Ensure password and session ID are sent over SSL applicable

10 S53: Virus scan all uploaded files using an inline virus scanner n/a

File Handling

Perform a virus scan on any user-uploaded files using an inline virus scanner, such as Clam AV. This helps protect against a malicious user who may upload malware for other users to download. Alternatively, an attacker may attempt to upload malware and then use a different vulnerability to execute the file.

10 S68: Protect all credit card PANs n/a

Cryptography

According to th CI-DSS), applications that store, transmit, or process credit card numbers

Fig. 4. A sample development guideline generated by SD Elements tool for a banking application

3.4 Umbrella Processes

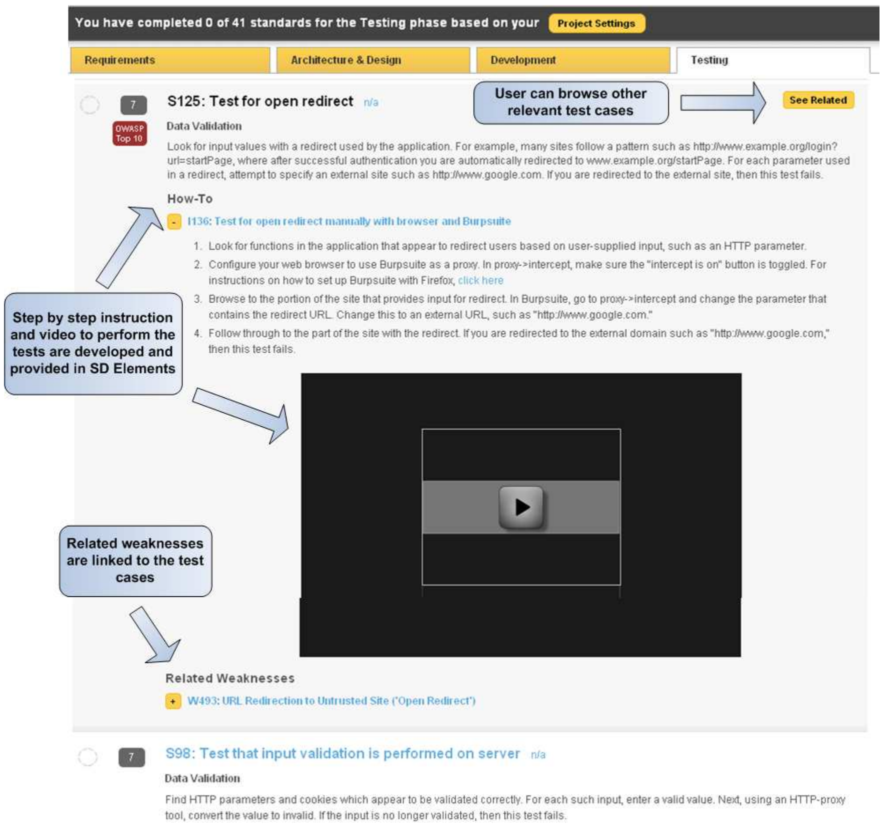
Incremental survey: The survey section of the tool can be answered gradually and incrementally, i.e., the more questions answered by the user, the more specific and accurate guidelines are retrieved by SD Elements.

A Traceability System: Users can trace which guidelines they have applied, which ones are not applicable, and which guidelines are outstanding. Applicable guidelines and implementation standards can be added to bug tracking systems and generated requirements can be imported to general requirements documents such as PDF and doc files.

4 Beta Test Plan

SD Elements will be in beta test in a variety of sectors, such as energy, independent software vendors, healthcare, and financial services. It will be mostly championed by application security managers and development team leaders. The beta tests will help us investigate various aspects of real world application of SD Elements. By the end of the beta test phase, we will have concrete data to evaluate usefulness and usability of SD Elements in real world practice.

1. How and in what logical path users browse the standards.
2. How the standards and requirements are applied in different phases of development and by what roles.


Project Settings
Requirements
Architecture & Design
Development
Testing
User can browse other relevant test cases
See Related
OWASP Top 10
Step by step instruction and video to perform the tests are developed and provided in SD Elements
Related weaknesses are linked to the test cases

You have completed 0 of 41 standards for the Testing phase based on your

7 S125: Test for open redirect n/a

Data Validation

Look for input values with a redirect used by the application. For example, many sites follow a pattern such as http://www.example.org/login?url=startPage, where after successful authentication you are automatically redirected to www.example.org/startPage. For each parameter used in a redirect, attempt to specify an external site such as http://www.google.com. If you are redirected to the external site, then this test fails.

How-To

- I136: Test for open redirect manually with browser and Burpsuite

1. Look for functions in the application that appear to redirect users based on user-supplied input, such as an HTTP parameter.
2. Configure your web browser to use Burpsuite as a proxy. In proxy->intercept, make sure the "intercept is on" button is toggled. For instructions on how to set up Burpsuite with Firefox, click here
3. Browse to the portion of the site that provides input for redirect. In Burpsuite, go to proxy->intercept and change the parameter that contains the redirect URL. Change this to an external URL, such as "http://www.google.com."
4. Follow through to the part of the site with the redirect. If you are redirected to the external domain such as "http://www.google.com," then this test fails.

Related Weaknesses

+ W493: URL Redirection to Untrusted Site ('Open Redirect')

7 S98: Test that input validation is performed on server n/a

Data Validation

Find HTTP parameters and cookies which appear to be validated correctly. For each such input, enter a valid value. Next, using an HTTP-proxy tool, convert the value to invalid. If the input is no longer validated, then this test fails.

Fig. 5. A sample SD Elements test case

3. Whether users treat guidelines only as an after the fact check list or developers use guidelines in daily development tasks.
4. Whether SD Elements help developers actually prevent the introduction of potential vulnerabilities into the code.
5. Whether users find SD Elements intuitive and usable.
6. Whether users need additional security knowledge sources in addition to SD Elements.

5 Related Work

Various web-based software vulnerability knowledge bases provide a shared and standard way for identifying, specifying, and measuring software weaknesses and vulnerabilities. Common Weakness Enumeration (CWE) [3] provides a unified, measurable set of software weaknesses for enabling effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems. National Vulnerability Database (NVD) portal provides a search engine over the Common Vulnerability and Exposure (CVE) and Common Configuration Enumeration (CCE) databases. However, these knowledge portals usually lack specific guidelines to prevent the introduction of vulnerabilities. The burden of identifying relevant vulnerabilities in the massive lists of these portals is on developers. SD Elements solves these issues by providing customized guidelines for preventing CWE vulnerabilities.

The need for security guideline customization has been addressed in another tool called TeamMentor [8]. TeamMentor only provides two layers of guidelines filtering: 1) technology and 2) role of the user. Thus, still a massive list of guidelines without specific categorization is provided to the user. A link between the suggested guidelines for different phases of life cycle is not considered, thus traceability is not possible. Project specific features and functionalities are not used to generate the list of guidelines, and still software developers can become overwhelmed with the large list of guidelines.

6 Conclusions and Future Work

SD Elements is a web-based security knowledge base that provides security guidelines for different development life cycle phases. The main goals of SD Elements is to build security into the application, from the early phases of the life cycle. The outstanding contribution of SD Elements is tailoring the guidelines according to project description. SD Elements helps businesses provide tangible security audit evidence and trace compliance with security standards.

The results of the beta tests will help us investigate whether by applying SD Elements, more vulnerabilities are actually prevented. In future releases, SD Elements will be customizable to different domains and businesses. End user administrators will be able to add their own questions, answers, and content items so that they can support any technology stack. Also, we will continuously add more content, thus SD Elements will work as a subscription service rather than a single tool.

References

1. I. A. Tondel, M. G. Jaatun, and P. H. Meland, "Security requirements for the rest of us: A survey," *IEEE Software*, vol. 25, pp. 20–27, 2008.
2. National Vulnerability Database. http://nvd.nist.gov/.
3. Common Weakness Enumeration. http://cwe.mitre.org/.
4. OWASP. http://www.owasp.org/.
5. PCI Secutity Standard Council, Data Security Standards (PCI DSS). https://www.pcisecuritystandards.org/.
6. M. J. Kenning, "Security management standard — iso 17799/bs 7799," *BT Technology Journal*, vol. 19, no. 3, pp. 132–136, 2001.
7. Web Application Security Consortuim Threat Classification v2.0. http://projects.webappsec.org.
8. Secure Development Standards. http://securityinnovation.com/products/teammentor/.