2009 International Conference on Availability, Reliability and Security

An architectural foundation for security model sharing and reuse

Per Håkon Meland*, Shanai Ardi†, Jostein Jensen*, Erkuden Rios‡,
Txus Sanchez‡, Nahid Shahmehri† and Inger Anne Tøndel*
*SINTEF ICT, Software Engineering, Safety and Security, NO-7465 Trondheim, Norway
Email: {per.h.meland,jostein.jensen,inger.a.tondel}@sintef.no
†Department of computer and information science, Linköpings universitet, SE-58183 Linköping, Sweden
Email: {shaar,nahsh}@ida.liu.se
‡European Software Institute, Corporacion Tecnológica Tecnalia, E-48170 Zamudio, Spain
Email: {erkuden.rios,jesus.sanchez}@esi.es

Abstract

Within the field of software security we have yet to find efficient ways on how to learn from past mistakes and integrate security as a natural part of software development. This situation can be improved by using an online repository, the SHIELDS SVRS, that facilitates fast and easy interchange of security artefacts between security experts, software developers and their assisting tools. Such security artefacts are embedded in or represented as security models containing the needed information to detect, remove and prevent vulnerabilities in software, independent of the applied development process. The purpose of this paper is to explain the main reference architecture description of the repository and the more general tool stereotypes that can communicate with it.

1. Introduction

As our society's reliance on software increases, so does the importance of information and software security [1]. A recent report by ENISA [2] states "*The direct cost to Europe of protective measures and electronic fraud is measured in billions of euros; and growing public concerns about information security hinder the development of both markets and public services, giving rise to even greater indirect costs.*"

Within the field of software security we have yet to find efficient ways to learn from past mistakes. As stated by Noopur Davis [3]:"*...over 90 % of software security vulnerabilities are caused by known software defect types. [...] the top ten causes account for about 75 % of all vulnerabilities.*" This is further supported by OWASP Top 10 [4] that describe the main security problems of web applications. We believe that for most systems it is possible to improve security substantially by focusing on common security problems that can be solved in similar ways in most, or maybe all, software. However, software developers are usually not security experts, and thus rarely have the required knowledge of and focus on security issues [1]. Information on known vulnerabilities is available in several online repositories, but in general they are more directed towards system administrators than developers [5].

The EU project SHIELDS [6] seeks to increase software security by: 1) bridging the gap between security experts and software developers by providing means for sharing software security expertise, and 2) providing this expertice in a reusable form that can be effectively used by developers and the tools they use. The heart of SHIELDS is a *Security Vulnerability Repository Service* (SVRS) and its contents. Security expertise is embedded in or represented as security models containing the needed information to detect, remove and prevent known vulnerabilities in software, independent of the applied development process. Security experts collaborate on creating these models, which can contain information to be read and used by humans, but also tool specific input. The SVRS is intended to allow a variety of tools to take advantage of and contribute to the content through open interfaces, either by creating new or modifying existing tools.

The purpose of this paper is to explain the main reference architecture description of the SVRS and the accompanying tool stereotypes. We explain the method that has been used to create the architecture description, followed by a description of the main components, information objects and stakeholders. Then we exemplify utilisation of the SVRS, before we discuss our contribution compared to related work. Finally the paper is concluded with remarks on further work.

2. Method

SHIELDS is a collaborative research and development project with eight partners from all across Europe, varying from academic institutions to software industry. Because of a diversity in background and expertise, the initial activity was for all partners to create a set of practical use case scenarios or user stories describing how they envisioned the SVRS to improve secure software development. This process resulted in 56 initial scenarios, which were refined and combined into 13 final scenarios describing functional aspect. The scenarios were used to establish a common understanding of concepts, behavior and to extract functional requirements. Furthermore, relevant actors, workflows, security requirements and interaction outcomes could be deduced.

The architectural description of the SVRS is developed according to the recommended practices from MAFIIA framework [7] and the IEEE standard 1471-2000 [8]. Different views [9] are used to describe structural and behavioral properties, as well as possible configurations and guidelines relevant when making detailed design and implementation decisions. A view consists of one or more models that describe and present different aspects related to structure and behavior.

In the next section we show central excerpts from the views related to the context and components of the SVRS architecture description. More comprehensive information is to be found in the D1.X and D3.X reports at the SHIELDS Website [6].

3. The SHIELDS SVRS

The SVRS architecture is built on the concept published by Ardi et al. [5], that is a centralised repository that enables people and *several* types of tools to exchange security information in order to stay up-to-date.

This first level architectural decomposition of the SVRS is shown in Figure 1, which also shows what we consider to be the *environment* of the system. This component view of the SVRS is actually loosely based on the Flickr architecture, which is a successful photo and video management and sharing application [10], [11]. Flickr does belong to another domain, but shares much of the same functionality with SHIELDS, also being a centralised, but highly scalable repository offering content access through a Web interface as well as interfaces for third party applications (submit and download functionality).

The *Application Logic* is where most of the actual functionality is realised, and it can access three types of information sources: 1) The *Repository storage*, which is optimised for security models and associated information. 2) A general purpose *Database* for storing information about users, statistics, session information, etc. 3) *External information sources* found in the environment, that typically need an *Adapter* to convert their data before use.

The *API* and *Endpoints* components make it possible for the *Modelling* and *Development* tools found in the environment to access the application logic. The API only provides a relevant subset of the total functionality of the application logic to the environment tools. The endpoints are interface libraries. We can add more endpoints to support more tools, or tools can adapt to the existing endpoints. So, there is one API, but many endpoints.

The *Page Logic*, *Templates* and *Web User Interface* form the direct Web entrance for the *Users*. The page logic generates the dynamic content, realised by a Web programming language. Templates are more static information related to the layout and appearance of the Web pages, such as CSS and image files. The Web user interface is what the human users interact with (note that we have not explicitly modelled a Web browser component, as this is considered to be something that only displays the Web User interface).

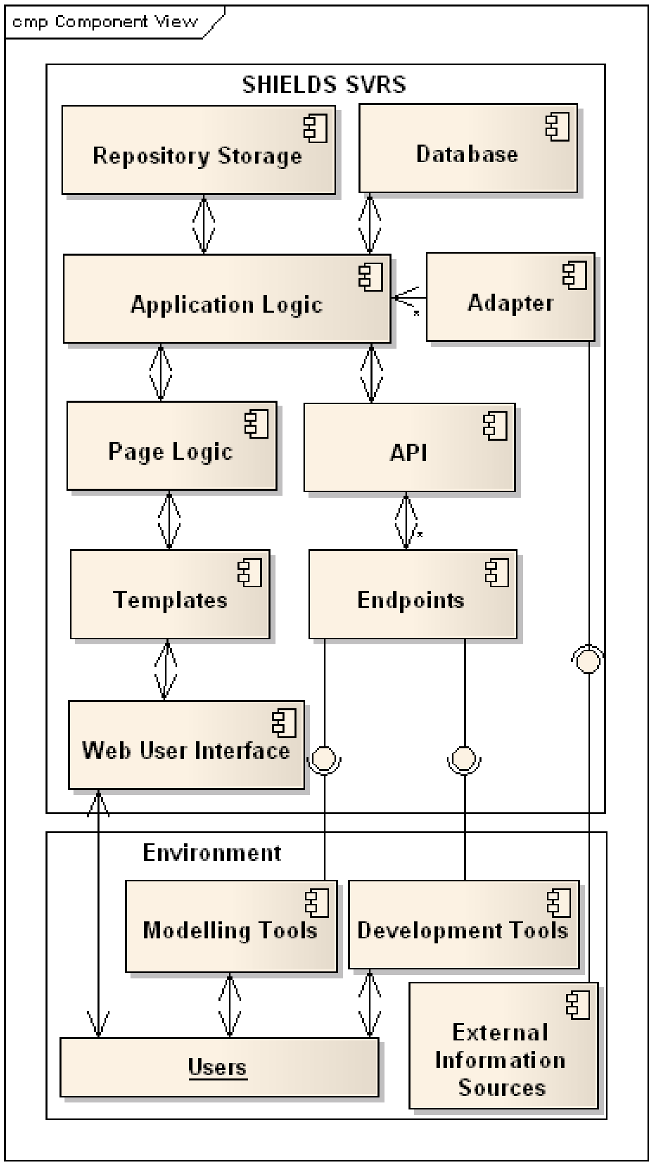

Figure 1. SHIELDS and its environment

In the environment we find users that either write (provide) or read (subscribe to) content. In addition to the already mentioned Web user interface, this is done through tools that belong to the *Modelling tools* or *Development tools* category. While the SVRS is a unique system, these tools found in the environment are more of a generic type. In the next section we present a set of stereotypes that specific tools can realise in order to interact with the SVRS.

3.1. Tool stereotypes

Modelling tools help SVRS content providers analyse and model formalised security models, such as software vulnerability information and how to mitigate them using security activities. This is largely a manual task requiring human expertise. Modelling tools aid the generation and submission of content to the SVRS, but should also be able to find and retrieve models from the SVRS in order to view, modify or reuse existing work.

The upper part of Figure 2 shows a stereotype variation where the modelling tool uses a local model storage. The reason for having this is that the security expert is not yet ready to upload the model (i.e. work in progress), or that some of the models are regarded as confidential and should therefore only be kept within the organisation. In order to synchronise with the SVRS, a dedicated component, *Remote sync* is controlled by the user interface and accesses the local storage.

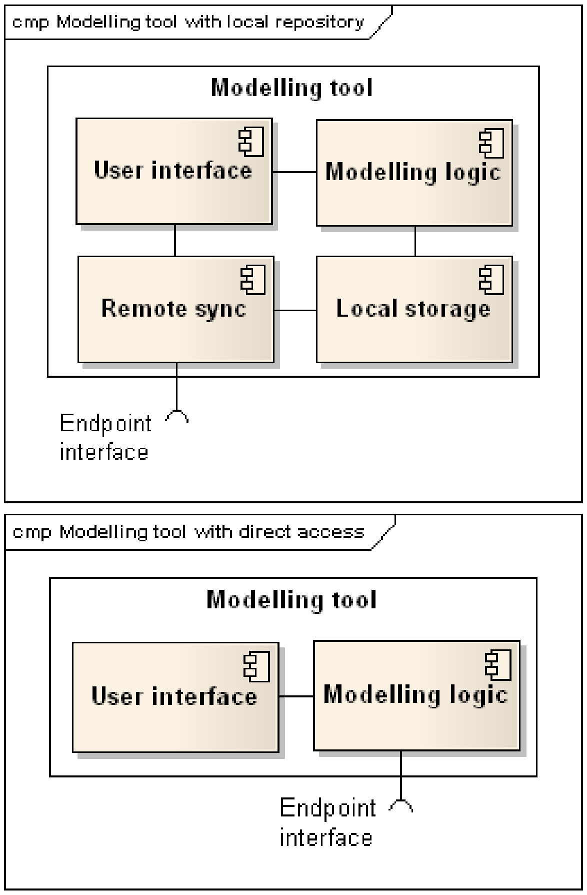

Figure 2. Modelling tool stereotypes for SHIELDS.

The lower part of Figure 2 shows another variation where the modelling logic accesses the SVRS directly (no local storage). Here it is important to avoid conflicts between tools, and we have to impose transaction (ACID[1]) properties.

Development tools connect and retrieve content from the SVRS, thereby staying constantly up-to-date with the latest security knowledge and providing developers and quality assurance staff with the latest information at their fingertips. These are general development tools used for design and implementation, or more specialised stand-alone security tools or IDE-extensions that in one way or another are able to utilise content from the SVRS. Development tools are mainly subscriber tools, but they might also provide information back to the SVRS, such as informing about the effectiveness of security activities and real statistics on vulnerability occurrences (in a controlled and protected manner). The upper part of Figure 3 shows a variation with direct access to the SVRS through the development logic.

The middle part of Figure 3 shows a special case where the development tool needs an adapter between the development logic and the endpoint interface. This is relevant when the provided interface is not fully compliant or when the content itself needs some kind of local conversion before it can be used. Gamma et al. [12] defines an adapter pattern that is suited for this purpose.

The lower-most option of Figure 3 is another special case where the development tool can use SVRS content through a dedicated plugin. This plugin is typically invoked from the user interface. Many development tools already support some kind of plugin technology as a way to extend their functionality.

3.2. Concepts and information model

The information in the SVRS is tied together through a set of central concepts that we need to explain: A *Vulnerability* can be modelled either as a general *Vulnerability Class* or a specific *Vulnerability Instance* (an instance is a realisation of a class). A vulnerability can lead to a *Security Breach Incident* (modelled as threats or attacks), and vulnerabilities might prevent *Security Goals* from being achieved, while *Security Activities* help us fulfill the security goals. Vulnerabilities, security goals and security activities can be modelled by respectively *Vulnerability Models*, *Security Goal Models*, and *Security Activity Models*. In a security activity model we can indicate *Security Activity Artefacts*, which can either be of the type *Security Tool* (using a security tool or usable input to a tool) or *Security Instruction* (documents to be read and used by humans in order to perform a certain security activity).

The information model shown in Figure 4 gives a simplified view on how we actually store this information. The central object is the general *Resource*, which *CoreElement*,

1. ACID is a typical database abbreviation for Atomicity, Consistency, Isolation, Durability.

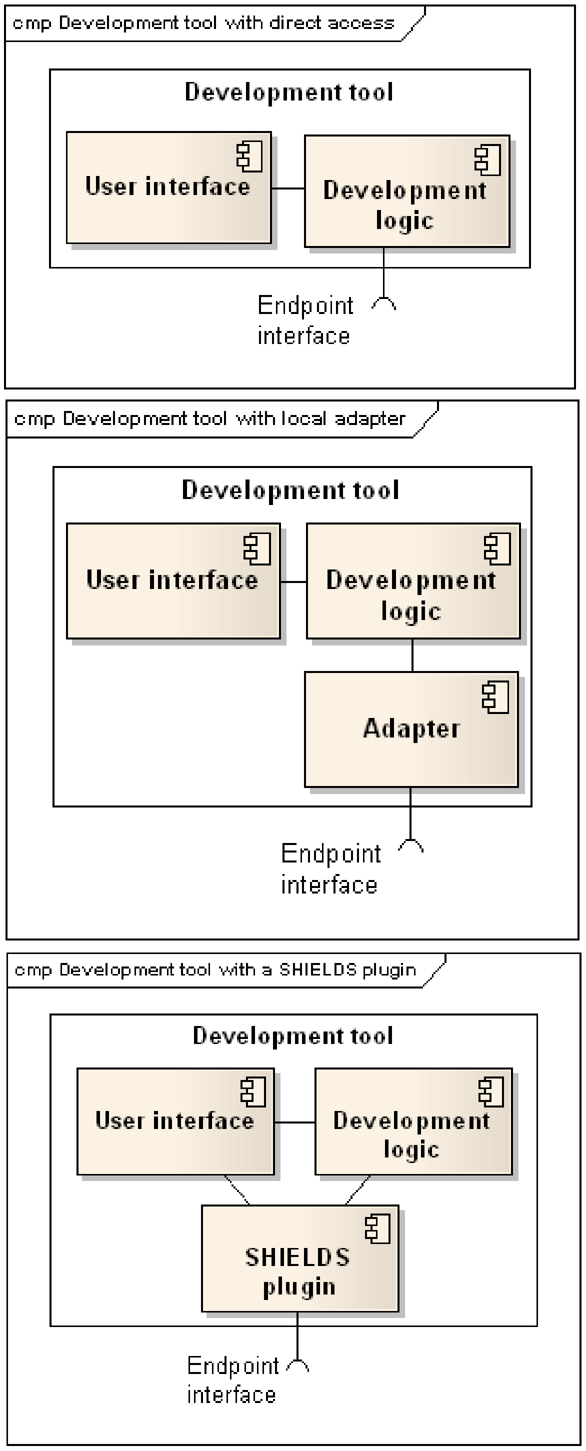

Figure 3. Development tool stereotypes for SHIELDS.

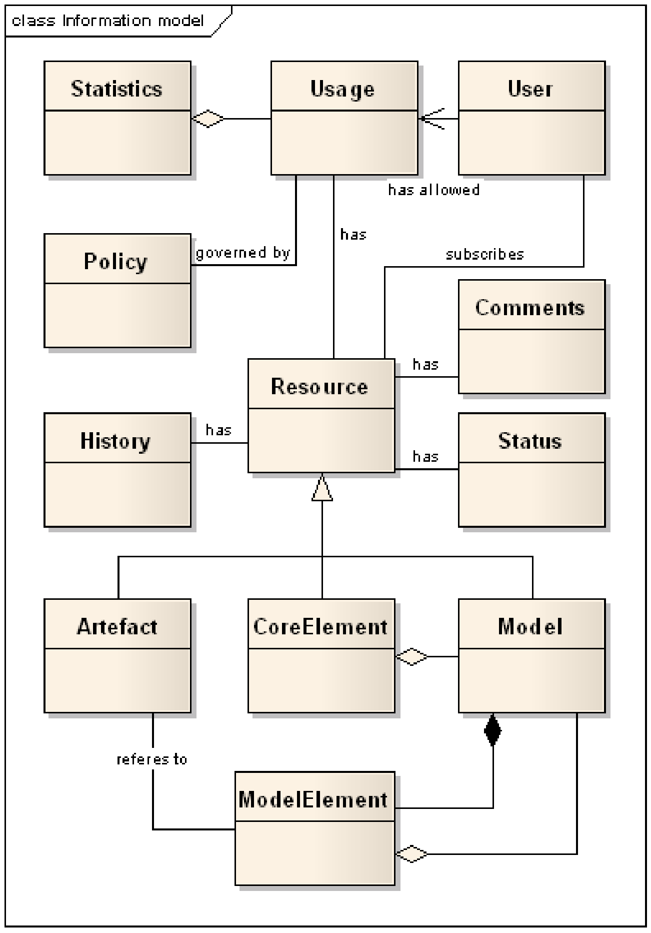

Figure 4. High-level information model in SVRS

Model and *Artefact* inherit from. CoreElements are common concepts for all formalisms and models in SHIELDS, and represent security knowlegde, such as vulnerabilities, vulnerability causes, security activities and security goals. A model consists of one or more *ModelElements*, which can again contain other models. ModelElements are able to refer to different types of Artefacts, which are resources representing some kind of information stored within or outside the SVRS. *History* and *Status* are used to support versioning and endorsement of the content of the SVRS respectively, and *Comments* are used for feedback. Users of the SVRS are either individuals or organisations, and their allowed *Usage* is governed by one or more *Policies*. Usage *Statistics* provide valuable information on the popularity and effect of the various resources. For concrete examples on what relevant security models look like in practice, see the already mentioned work by Ardi et al. [5].

3.3. Stakeholders

Based on the work of Landwehr et al. [13], the SVRS has three main types of stakeholders:

1) A *provider* is typically a *security expert* who creates, modifies, endorses or deletes SVRS content. We also have *commercial providers* that creates content that requires some kind of fee to be accessed.

2) A *subscriber* is typically a *software developer*, either using the Web interface directly or tools that access the SVRS. The SVRS also provides information aimed towards *security researchers*, i.e. trends and statistics related to vulnerability occurrences and effectiveness of security activities.

3) In the *SVRS management team* there are several roles related to the operation of the SVRS, e.g. quality assurance, user management, maintenance and comment moderation.

Additional stakeholders are the *tool developers*, that create or enable modelling/development tools to utilise the SVRS, and the indirect *software users* who benefits from improved software security.

4. Example of use

SQL Injection is a vulnerability class typically found in Web applications. It allegedly caused exposure of 3000 records of a US online bank in October 2007, the UN Web site to be defaced in August 2007, 18 million customer records to be stolen from a Korean auction site in February 2008 and the loss of 226 000 client records uf a US financial firm in February 2008, just to mention a few examples [14].

A lot of knowledge exists about the causes and threats related to SQL Injections. For example, we know that proper server-side input validation is a very efficient remedy, but still this is considered to be one of the main threats to Web applications because the same development mistakes are made over and over again.

So, how does the SVRS architecture help us improve this situation? We have modified two existing security modelling tools according to the stereotype with a local repository shown in Figure 2. Security experts are able to use these tools to find and describe causes and relations between causes, and then create intuitive models that systematically describe different types and levels of mitigation activities. As several security experts are able to cooperate on these models through the SVRS, these models quickly get to a mature and usable state.

One of the modelling tools can also be imported as a plugin into the Eclipse IDE [15], hence realising the lowermost development tool stereotype of Figure 3, and used by the software developers to locate, retrieve and view relevant models from the SVRS. In the SQL Injection case, these models will typically contain information that educates the developers on the causes to the problem (e.g. only client side input validation), what are the possible misuse cases (e.g. use of escape characters), how to perform manual inspection (e.g. look for dynamic SQL queries), find a relevant design pattern (e.g. InterceptingValidator [16]) and so on. The point is to be as concrete and specific as possible, and make the information easily available from tools developers are already familiar with.

Similarly, we are working on modifying a static analysis tool and three testing tools (both active and passive) so that they realise the other two development tool stereotypes. As analysis and testing rules are specialised and embedded into the specific models for the relevant vulnerability, we will suffer less from false positives, and the tools can be specifically instructed to look for newly discovered threats.

5. Discussion

The architecture of the SVRS supports interaction with security modelling tools, storing the models so that they are openly available and related to each other, and letting development tools take advantage of the content. It conforms to the requirements proposed by Ardi et al. [5] for new security improvement tools that aim to "*contribute to the creation of more secure software and improve the security awareness of software developers.*"

Landwehr et al. [13] suggest three alternative architectures for a vulnerability archive: centralised, federated, and distributed search-engine-controlled. The SVRS is a centralised repository, but the tools can use a local repository that also enables the federated approach. This is necessary because some businesses might be very reluctant to share information about vulnerabilities in their products, and are in some cases not permitted to share certain information [5].

The tools we are currently using to interface the SVRS according to the stereotypes are project internal and not widely used in the industry. However, in order to obtain a crucial mass of both content and users we need other tools to interface the SVRS as well. A *SHIELDS compliant* programme will therefore be developed to facilitate this.

Two of our major challenges are to further develop the internal information structure so that the users easily find the right information and tackling issues related to performance when the content amount grows.

As Ardi et al. [5] already have described, there are several existing security related repositories available on the Internet, both commercial and open-content based. The initiative that resembles SHIELDS the most is the Information Security Automation Program (ISAP) together with the Security Content Automation Protocol (SCAP) [17], sponsored by the U.S. Department of Homeland Securitys Computer Security Division and hosted as a part of the National Vulnerability Database (NVD). We do not consider ISAP and SCAP to be direct competitors to the SHIELDS SVRS as they mainly seek to improve security monitoring and configuration of software, not actual development. In contrast, we consider these to be potential *external information sources* as they provide complementary knowledge that the SVRS can either provide links to or import information from (see Section 3).

Other related work is the Open Risk Model Repository (ORI-MOR) [18], which similar to SHIELDS since it is an open source repository of security information with an accompanying security tool, but on a very different level. SOMAP is focused on high-level risk management while SHIELDS is focused on practical software development.

6. Conclusion and further work

We have presented a reference architecture for a Security Vulnerability Repository Service (SVRS) that supports sharing and reuse of security artefacts, and which will hopefully help bridge the gap between security experts and software developers. We are currently validating this approach by modifying various tools to interact with the SVRS, and gathering feedback from the software industry on the their usability and perceived usefulness of the SVRS contents.

Aknowledgements

The research leading to these results has received funding from the European Communitys Seventh Framework Programme (FP7/2007-2013) under grant agreement no 215995. We would also like to aknowledge the valuable help from the rest of the SHIELDS Consortium.

References

[1] H. Mouratidis, P. Giorgini, and G. Manson, "When security meets software engineering: a case of modelling secure information systems," *Information Systems*, vol. 30, no. 8, pp. 609–629, 2005.

[2] R. Anderson, R. Bhme, R. Clayton, and T. Moore, "Security economics and the internal market," The European Network and Information Security Agency (ENISA), Tech. Rep., January 31st. 2008.

[3] N. Davis, "Developing secure software," *Software Tech News*, vol. 8, no. 2, pp. 3–7, July 2005.

[4] OWASP Foundation, "Owasp top 10, the ten most critical web application security vulnerabilities, 2007 update," http://www.owasp.org/index.php/Top_10_2007 (accessed November 1st 2008).

[5] S. Ardi, D. Byers, P. H. Meland, I. A. Tøndel, and N. Shahmehri, "How can the developer benefit from security modeling?" in *The first international workshop on secure software engineering (SecSE'07)*. Vienna, Austria: IEEE Computer Society, 2007, pp. 1017–1025.

[6] SHIELDS Consortium, "SHIELDS - Detecting known security vulnerabilities from within design and development tools," http://www.shieldsproject.eu/ (accessed November 1st 2008).

[7] U. Johansen, E. Stav, and S. Walderhaug, "The mafiia handbook - an architectural description framework for information integration systems," SINTEF ICT, Tech. Rep. STF90 A05139, 2003.

[8] IEEE, "Recommended practices for architectural descriptions of software-intensive systems," IEEE, Tech. Rep. Std 1471-2000, 2000.

[9] P. Kruchten, "The 4+1 view model of architecture," *IEEE Software*, vol. 12, no. 6, pp. 42 – 50, 1995.

[10] Yahoo! Inc, "Flickr - photo sharing," http://www.flickr.com (accessed April 18th 2008).

[11] Hoff, Todd, "Flickr architecture - high scalability," http://highscalability.com/flickr-architecture (accessed April 18th 2008).

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[13] C. E. Landwehr, B. C. Gabrielson, and J. D. Humphrey, "Requirements and approaches for a computer vulnerability data archive," in *Invitational Workshop on Computer Vulnerability Data Sharing*, Gaithersburg, MD, 1996.

[14] L. Clark, "Businesses must keep back door locked to hackers," *ComputerWeekly*, vol. 2008, no. March 20th, 2008.

[15] The Eclipse Foundation, "Eclipse.org home," http://www.eclipse.org/ (accessed November 1st 2008).

[16] C. Steel, R. Nagappan, and R. Lai, *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Prentice Hall, 2005.

[17] NIST and OSD and DHS and NSA and DISA, "The information security automation program and the security content automation protocol," http://nvd.nist.gov/scap.cfm (accessed October 31st 2008).

[18] SOMAP.org, "Security officers management & analysis project," http://www.somap.org/ (accessed November 1st 2008).