

# On a Classification Approach for SOA Vulnerabilities

Lutz Lowis and Rafael Accorsi

*Department of Telematics, Albert-Ludwig University of Freiburg*

*Freiburg, Germany*

*Email: {lowis,accorsi}@iig.uni-freiburg.de*

**Abstract**—Vulnerabilities in operating systems and web applications have been and are being put into various classifications, leading to a better understanding of their causes and effects, and to improved vulnerability management tool support. In a service-oriented architecture (SOA), additional vulnerabilities exist in the implementations of new standards such as BPEL and SOAP. Attackers can exploit these vulnerabilities to interfere with the business processes, which are executed as orchestration of services. We describe our approach and ongoing work of creating a SOA vulnerability classification.

**Keywords**-Vulnerability Classification, Vulnerability Management, Security, SOA

## I. INTRODUCTION

Many companies hesitate to implement a service-oriented architecture (SOA) because of security concerns [1]. While some SOA vulnerabilities have already been identified [2], it is obviously impossible to predict all of them. However, observing over time new entries of vulnerability databases, such as the National Vulnerability Database (NVD) [3], confirms the presumption that completely new types of vulnerabilities are rare, yet existing types are constantly adjusted to changing technologies. Consequently, a logical and tangible question is which of the previously and currently observed vulnerabilities, such as they can be found in public vulnerability databases (e.g., [3], [4], [5]), will play a role in a SOA in either their current form or a form adjusted to new technologies such as the Business Process Execution Language (BPEL) and SOAP.

While much effort is directed towards “classical” vulnerabilities [6], [7] (e.g., buffer overflows) and web application vulnerabilities [8], [9] (e.g., cross-site scripting), SOA vulnerability analysis has not yet received the same attention. Considering its potential regarding improved tool support for avoiding, finding, fixing, and monitoring vulnerabilities, a SOA vulnerability classification is essential. Our immediate goal is to create such a classification of SOA vulnerabilities by analyzing existing vulnerabilities regarding their “survivability” in a SOA.

Considering the amount of previous classifications and the fact that they all suffer from a certain degree of ambiguity (cf. Section III), we are not interested in creating “yet another classification”. Instead, our goal is to create a vulnerability classification as an intermediate step on the way

to improved SOA security tool support. Such tools typically work with vulnerability patterns (cf. Section II), which can be derived from classifications. In order to aim the tools at the vulnerabilities with the biggest impact and the highest exploitability, we first put special emphasis on the impact the corresponding exploits could have on the business process, which, in a SOA, is executed as orchestration of services. In this paper, we present our approach of extrapolating SOA vulnerabilities from previous vulnerability collections and classifying the resulting vulnerabilities according to their impact on the business process.

## II. TERMS AND DEFINITIONS

Vulnerabilities are “design or implementation errors in information systems that can result in a compromise of the confidentiality, integrity, or availability of information stored upon or transmitted over the affected system” [10]. In the following, we differentiate between “classical” and “web application” vulnerabilities. Classical vulnerabilities are those which can be exploited without any web technology being involved (e.g., buffer overflows). Web application vulnerabilities require the use of web technology to be exploited (e.g., cross-site scripting).

Vulnerability management comprises avoiding, finding, fixing, and monitoring vulnerabilities. Facilitation of these tasks is the goal of vulnerability analysis (cf. Bishop [11, chapter 20]), which typically leads to vulnerability classifications. In turn, the definition of patterns for each vulnerability class directly lends itself to tool support. For example, format string vulnerability detection [12] or the static code analysis tools discussed in [13], [14] all need a precise formalization of the vulnerabilities they are supposed to find.

We refer to a SOA as a business process execution environment according to the OASIS SOA reference model and architecture [15], [16]. The differentiation between visibility, interaction, and real world effect of services will be followed during our vulnerability analysis as, according to OASIS, these are the three key concepts of the SOA paradigm. A SOA vulnerability is a vulnerability present in such an environment. Vulnerabilities can exist in any of the SOA layers; Figure 1 shows our view on SOA layers, following, amongst others, Krafzig et al. [17] and IBM’s reference architecture [18].

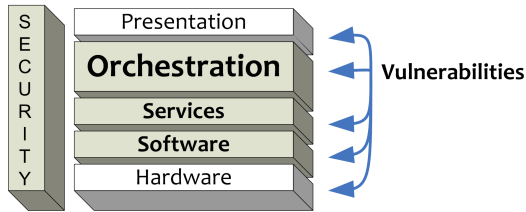


Figure 1. Layers of a service-oriented architecture

Our focus lies on the orchestration layer, as it represents the biggest change a SOA brings along, yet so far has received little attention in vulnerability analysis terms. The orchestration layer contains the business process logic, typically implemented as an execution engine calling specific services in the order and with the parameters specified in a BPEL document. Vulnerabilities which allow an attacker to modify either the activities or their sequence in a business process are our primary topic of interest. An example of such a vulnerability is SOAP action spoofing [2], in which an attacker alters a SOAP message so that an action other than the predefined is executed.

Services are self-contained functions offering a WSDL description and communicating through SOAP. The software layer will increasingly be implemented using managed code, meaning a decrease of buffer overflows and similar vulnerabilities. Nevertheless, previous vulnerabilities might still be present in legacy software, so we share the view and classification of Yu et al. [19] (discussed below) for both the software and service layer.

### III. PREVIOUS VULNERABILITY CLASSIFICATIONS

Various vulnerability classifications of classical and web application vulnerabilities are available. Examples focussing on classical vulnerabilities are the RISOS study [20], the protection analysis study [21], the NRL taxonomy [22], the classifications by Aslam [23], Krsul [24], Tsipenyuk et al. [25], Thompson et al. [26], Howard et al. [27]. A mix of classical and web application vulnerabilities can be found in top vulnerability lists such as SANS Top 20 [6], the Open Source Vulnerability Database (OSVDB) [4] (which does not only collect vulnerabilities in open source software, although its name might seem to suggest otherwise), the Open Web Application Security Project (OWASP) [8], and the NVD with Common Vulnerability and Exposure (CVE) entries [3] and Common Vulnerability Scoring System (CVSS) [28] data. Some of these collections focus on actual vulnerabilities rather than vulnerability classes but nevertheless have an underlying classification. The creation of these classifications has resulted in pattern definitions and, thus, improved tool support for vulnerability management.

Parrend et al. [29] created a classification of Java vulnerabilities by considering the progress from stand alone to component-based systems to Service Oriented Programming

Input Manipulation	65.53
Information Disclosure	16.62
Denial of Service	12.19
Authentication Management	1.67
Race Condition	1.35
Cryptographic	1.28
Misconfiguration	1.04
Hijacking	0.18
Infrastructure	0.14

Table I

OSVDB CLASSIFICATION WITH PERCENTAGE OF OSVDB ENTRIES

(SOP) platforms. Besides describing how previously known vulnerabilities can be exploited in Java SOP, Parrend et al. identify new Java component and SOP vulnerabilities. Considering the progress from operating system to web application to SOA vulnerabilities, our view is very similar. The main difference is that while Parrend et al. classify Java vulnerabilities on the service and software layer, our focus is on the orchestration layer with its vulnerabilities from, e.g., BPEL and SOAP implementations.

Two major problems with vulnerability classifications are the level of abstraction and the point of view, as Bishop argues in [11, chapter 20]. Switching either usually makes the classification ambiguous. Of the above classifications, only Krsul [24] avoids this trap to a large degree. In order to prevent switching and the resulting ambiguity, both the level of abstraction and the point of view should clearly be stated when a classification is given.

We have performed a brief analysis of two of the major vulnerability databases, OSVDB and NVD, in order to provide an example of existing classifications and the actual distribution of vulnerability types in current databases.

Table I shows the OSVDB classification with the according percentage of entries. The total number of vulnerabilities in OSVDB was 52510, of which we considered 44375, as the remainder either had not yet been fully classified or was of the type “other” or “unknown”.

Table II shows the same information for the Common Weakness Enumeration (CWE) [30] classification applied to NVD entries. The total number of vulnerabilities in NVD was 35776, of which we considered 9612, as the remainder either had not yet been fully classified or was of the type “other”, “not in CWE”, “insufficient information”, or “design error” (these types are not mapped in NVD). Both tables are sorted in descending order of percentage, and values have been rounded to the second decimal place.

The web services software vulnerability classification by Yu et al. [19] is a promising step in the SOA direction. However, besides lacking a clear specification of its level of abstraction and point of view, the presented fishbone chart contains *only* the web *application* vulnerabilities listed in OSVDB [4], accompanied with the note that “Web Services inherit almost all kinds of traditional Web application vulnerabilities while opening door to new type of attacks at

SQL Injection	17.85
Cross-Site Scripting (XSS)	14.58
Buffer Errors	12.88
Permissions, Privileges, and Access Control	9.04
Input Validation	7.98
Code Injection	7.8
Path Traversal	6.79
Resource Management Errors	4.97
Information Leak Disclosure	3.91
Numeric Errors	3.06
Authentication Issues	2.72
Link Following	2.26
Cross-Site Request Forgery (CSRF)	1.47
Credentials Management	1.32
Configuration	1.12
Cryptographic Issues	0.97
Format String Vulnerability	0.59
Race Conditions	0.53
OS Command Injections	0.16

Table II  
CWE CLASSIFICATION WITH PERCENTAGE OF NVD ENTRIES

application or data level as its technology is relying on SOAP and XML messages” [19, section 4.2]. While we generally share that view, the web application vulnerabilities which do not apply to a SOA should be filtered out or at least augmented with the assumptions under which they could still be exploited. For example, “signal handling flaws” such as OSVDB’s vulnerability 9737 (<http://osvdb.org/9737>) only apply to a SOA if there are vulnerable service components (in this case, FTP software) reachable through direct, non-SOAP TCP traffic. This assumption of an improperly configured firewall and, potentially, unmanaged code should be made explicit to show under which circumstances a vulnerability is exploitable in a SOA. Even so, it remains unclear which of the vulnerabilities in the fishbone chart are applicable on the orchestrational layer; regarding the services and software layer, the OSVDB based classification of Yu et al. will nevertheless be a valuable input when compiling the SOA vulnerability classification.

As a side note: Yu et al. report that based on OSVDB data, the top three attack types DoS, information disclosure, and input manipulation account for roughly 85 percent of vulnerabilities from July 2003 to October 2004. Based on our analysis of current (February 2009) OSVDB data as shown in table I, the value is 95 percent, but taking the types “other” and “unknown” into consideration to reflect Yu’s approach results in 80 percent. This decrease in prevalence of the top three attack types is surprising given the large number of recently found input manipulation vulnerabilities. A comparison of this OSVDB percentage with the according percentage of other databases would be interesting in terms of attack trends and differences between databases regarding the types of vulnerabilities being reported to them. An exemplary question is how the distribution of vulnerabilities differs between databases such as OSVDB and NVD. However, in this case NVD data does not lend itself to direct

comparison as the top three NVD types SQL injection, XSS, and buffer errors do not unequivocally map to OSVDB’s attack types.

#### IV. CREATING A SOA VULNERABILITY CLASSIFICATION

The typical approach to creating a SOA vulnerability classification would be to wait until a considerable amount of vulnerabilities has been collected, and then look for similarities based on which classes could be defined. However, the idea of creating the classification *before* actual attacks happen seems promising as it would allow to at least monitor, maybe even avoid many of the vulnerabilities in SOA implementations. The general idea of defining vulnerability characteristics can be found in [31], and transferring such characteristics to new technologies is one way of vulnerability prediction. One such approach is described in [32], using a web service scenario as an example. Based on the observation that completely new types of vulnerabilities are rare, but an ongoing adjustment of existing types to changing technologies is typical, we are proceeding as follows.

Our approach is to examine existing vulnerabilities regarding their “survivability” in a SOA. In order to do so, we define the above target SOA and then analyze existing vulnerabilities regarding the technical assumptions under which a vulnerability would be exploitable in a SOA. Such technical assumptions are, e.g., the use of managed code and SOAP- or even BPEL-capable firewalls (see [33] for an example). We then record those SOA exploitability assumptions for each of the vulnerabilities. Referring to [31], these assumptions can be seen as characteristics. Also, we check whether the vulnerability could have a SOA successor, that is, a similar vulnerability based on the new environment of a SOA. The new classification will then be created by grouping the SOA vulnerabilities, especially with regard to the layers of the SOA model (cf. fig. 1) and service visibility, interaction, and effect.

To avoid the classification ambiguity discussed by Bishop (see above), we define the classification’s level of abstraction and point of view. Our level of abstraction is the orchestrational layer, where services are called in a specific order to implement a business process. Our point of view is that of the business process which is to be executed as service orchestration.

We started our analysis with the OSVDB based vulnerability classification of Yu et al. Going through the classes, we check each for its survivability in a SOA, that is, we ask two questions: Under which assumptions could this vulnerability be exploited in a SOA environment? And: Could this vulnerability manifest itself in a new, SOA-style form? Three examples, one for each of the main attack types input manipulation, DoS, and information disclosure, illustrate our approach.

Input manipulation is the largest attack type reported by OSVDB, with more than half of the entries attributed to that

```

<Envelope> <Body> <Employee>
  <Name>Honest Ed<ID>772</ID></Name>
  <ID>2312</ID>
</Employee> </Body> </Envelope>

```

Figure 2. XML injection example

type. SQL injection vulnerabilities are a prominent subtype and continue to be found in web applications. Their SOA equivalent are XML injection vulnerabilities. Our example in figure 2 follows the one of Gruschka et al. [2], who showed that a .NET web service can be tricked into assigning wrong parameter values by inserting one parameter into another. As message alteration is in our threat model defined above, inserting an additional parameter into a SOAP message is within the attacker’s reach. In terms of service visibility, interaction, and effect, this vulnerability can have an impact on both interaction and effect. On the one hand, the attacker can change values in records, thus altering the service’s effect, on the other hand, she can influence the interaction by changing control parameters. Regarding the orchestration layer, this vulnerability could allow an attacker to change the business process flow to his liking by influencing the parameters the process logic makes use of.

DoS can be achieved by drowning a destination in requests, and in some scenarios this cannot be prevented. In a SOA, an attacker can hijack the workflow engine, which typically is equipped with powerful resources, by spoofing the WS-Addressing return endpoint URL (see Gruschka et al. [2] for details). The attacker can then direct a DoS attack against a freely chosen target, requiring little effort for message alteration but causing substantial impact through the hijacked engine. In terms of service visibility, interaction, and effect, this vulnerability can hinder or stop interaction and thereby prevent desired effects from occurring. From an orchestrational point of view, this vulnerability enables an attacker to harm the availability of the whole business process or parts of it, depending on which target is chosen.

Information disclosure is often caused by lacking input validation, e.g., of HTTP requests, database inputs, or browser inputs (see the fishbone chart by Yu et al. [19]). SOAP messages offer a new venue for attack, as Gruschka et al. [2] have demonstrated with their SOAPAction spoofing. By changing the SOAP action either in the HTTP header or in the SOAP envelope, an attacker can bypass SOAP or HTTP filters and invoke an action. Figure 3 shows a SOAP message which will return an employee’s name if the envelope’s action is invoked, but if the HTTP header’s action overrides the envelope, the employee’s roles will be disclosed. In terms of service visibility, interaction, and effect, this vulnerability lets the attacker diverge the interaction. On the orchestration layer, this vulnerability gives an attacker the possibility to invoke actions other than the ones specified

```

POST /Demo.asmx HTTP/1.1
[...]
SOAPAction: "getEmployeeRoles"
<Envelope> <Body> <getEmployeeName>
  <EmployeeID>123</EmployeeID>
</getEmployeeName> </Body> </Envelope>

```

Figure 3. SOAPAction spoofing example

in the executed BPEL script. In addition, this possibly leads to a business process interruption as the execution engine might stop the process after the original action times out because it was never invoked by the original SOAP message.

Figure 4 shows how these exemplary vulnerabilities fit into the orchestration layer of the SOA vulnerability classification, and how the other layers of the classification will be filled.

Orchestration				
Vuln. / Attrib.	XML injection	WS-Addressing spoofing	SOAPAction spoofing	...
Class	Input Manipulation	Denial of Service	Information Disclosure	
Exploitability assumptions	Incomplete schema validation	Unverified endpoint URL	Either HTTP or SOAP unfiltered	
Impact on service visibility	/	Only affected if registry is targeted	/	
Impact on service interaction	Changed control flow	Slower/no interaction	Interaction is rerouted	
Impact on service effect	Changed data	Effect fails to appear	/	
...				

<b>Services</b>	vuln. classes to be derived from Yu et al., OWASP, etc.
<b>Software</b>	vuln. classes to be derived from Krsul, SANS, etc.

Figure 4. Sketch of the SOA vulnerability classification

## V. NEXT STEPS

Currently, we are in the process of analyzing the OWASP-based classification by Yu et al. [19], i.e., deriving and collecting SOA vulnerabilities. We will then use the additional vulnerability information sources CAPEC [34], NVD [3], SANS [6], and Web Application Security Consortium [9] to obtain and analyze more vulnerability types. CAPEC (Common Attack Pattern Enumeration and Class, [34]) is a listing of about one hundred attack types containing real-world examples and related vulnerabilities. NVD is a vulnerability database, SANS has compiled a list of top

(classical and web application) vulnerabilities, and the Web Application Security Consortium has compiled a list of web application attack classes. Fully acknowledging that it is impossible to compile a complete list of vulnerabilities, we will stop the analysis after covering the above sources which we consider a representative collection. Our next step will be to group the derived SOA vulnerabilities together and classify them according to the SOA model with its layers mentioned above, paying special attention to each vulnerability's impact on service visibility, interaction, and effect. Grouping by the SOA exploitability assumptions we will have recorded for each vulnerability, further analysis will become possible regarding effective and efficient ways of monitoring or even avoiding SOA vulnerabilities.

Besides classification, the challenge will be to find a formalization of vulnerability patterns that allows tool support. Such vulnerability patterns could then be used to search for and monitor vulnerabilities. CAPEC definitions are one option, however, their "probing techniques" element often asks for manual action, whereas we aim at highly automated support, for which a more precise description is necessary. Another option are Open Vulnerability and Assessment Language (OVAL) [35] definitions, extended by additional elements to incorporate SOA aspects such as web service descriptions.

## VI. CONCLUSION

Compared to classical and web application vulnerabilities, SOA vulnerabilities have not yet been thoroughly analyzed. With service orchestration being the biggest change a SOA brings along, the question is which vulnerabilities a business process, executed as such an orchestration, can have. Additional questions concern the "survivability" of previous vulnerabilities in a SOA, and the business process effects an exploit can have. We are creating a classification of SOA vulnerabilities to answer these questions.

Vulnerabilities on the orchestration layer can lead to various changes in the business process, for example, services other than the specified being called or an undesired execution order. A classification of such vulnerabilities regarding their impact on the business process supports risk identification approaches such as [36] and compliance monitors such as presented by Accorsi et al. [37]. Also, it can help to identify possible leaks in the information flow within business processes, an analysis pursued by, e.g., Accorsi et al. [38]. Security test case derivation is a further use of such a classification in SOA testing, for which Canfora et al. [39] have identified the main challenges. Finally, from a compliance point of view, a classification of SOA vulnerabilities would help implement and adhere to frameworks and standards such as ISO 27001, ISO 27002, CobiT [40], and the new Consensus Audit Guidelines (CAG) [41]. Model-based compliance engineering, such as described by Höhn et al. [42], or the identification of policy violations for

audits, such as Accorsi et al. [43] do for privacy, benefit from vulnerability patterns as they can be derived from vulnerability classifications.

## ACKNOWLEDGEMENTS

We thank Meiko Jensen, co-author of [2], for sharing his SOA expertise and discussing attack scenarios with us.

## REFERENCES

- [1] G. Trub and L. Olski, "Global report on soa/web services security initiatives," GMG Insights, Tech. Rep., 2008.
- [2] N. Gruschka, M. Jensen, R. Herkenhöner, and N. Luttenberger, "Soa and web services: New technologies, new standards - new attacks," in *Proceedings of the 5th IEEE European Conference on Web Services (ECOWS), Halle(Saale), Germany, 2007*.
- [3] National Institute of Standards and Technology, "National vulnerability database," <http://nvd.nist.gov/>, 2009. [Online]. Available: <http://nvd.nist.gov/>
- [4] Open Security Foundation (OSF), "Open Source Vulnerability Database (OSVDB)," <http://osvdb.org>, 2009. [Online]. Available: <http://osvdb.org>
- [5] SecurityFocus, "Securityfocus vulnerability database," 2009. [Online]. Available: <http://www.securityfocus.com/vulnerabilities>
- [6] SANS, "Sans top 20 security risks," <http://www.sans.org/top20/>, 2007. [Online]. Available: <http://www.sans.org/top20/>
- [7] MITRE Corporation, "Making security measurable," <http://measurablesecurity.mitre.org/>, 2009. [Online]. Available: <http://measurablesecurity.mitre.org/>
- [8] The OWASP Foundation, "Open web application security project (owasp)," <http://www.owasp.org/>, 2009. [Online]. Available: <http://www.owasp.org/>
- [9] Web Application Security Consortium, "Web application security consortium threat classification," 2009. [Online]. Available: <http://www.webappsec.org/projects/threat/>
- [10] D. Turner, M. Fossi, E. Johnson, T. Mack, J. Blackbird, S. Entwisle, M. K. Low, D. McKinney, and C. Wueest, "Symantec global internet security threat report: Trends for july to december 2007," Symantec, Tech. Rep., 2008.
- [11] M. Bishop, *Introduction to Computer Security*. Addison-Wesley, Pearson Education, 2004.
- [12] U. Shankar, K. Talwar, J. S. Foster, and D. Wagner, "Detecting format string vulnerabilities with type qualifiers," in *In Proceedings of the 10th USENIX Security Symposium*, 2001, pp. 201–220.
- [13] B. Chess and G. McGraw, "Static analysis for security," *IEEE Security and Privacy*, vol. Nov/Dec, pp. 32–35, 2004.

- [14] M. Martin, B. Livshits, and M. S. Lam, "Finding application errors and security flaws using pql: a program query language," in *20th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, San Diego, California, USA, 2005.
- [15] OASIS, "Reference model for service oriented architecture v1.0," OASIS, 2006. [Online]. Available: [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=soa-rm](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm)
- [16] —, "Reference architecture for service oriented architecture v1.0," OASIS, 2008. [Online]. Available: [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=soa-rm](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm)
- [17] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA*. Prentice Hall, 2004.
- [18] A. Arsanjani, L.-J. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah, "Ibm developer works: Design an soa solution using a reference architecture," 2007. [Online]. Available: [http://www.ibm.com/developerworks/architecture/library/ar-archtemp/index.html?S\\_TACT=105AGX20&S\\_CMP=EDU](http://www.ibm.com/developerworks/architecture/library/ar-archtemp/index.html?S_TACT=105AGX20&S_CMP=EDU)
- [19] W. D. Yu, D. Aravind, and P. Suphaweek, "Software vulnerability analysis for web services software systems," in *Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC)*, 2006, pp. 740–748.
- [20] R. Abbott, J. Chin, J. Donnelley, W. Konigsford, S. Tokubo, and D. Webb, "Security analysis and enhancements of computer operating systems. technical report nbsir 76-1041," ICET, National Bureau of Standards, Washington, DC 20234, Tech. Rep., 1976.
- [21] R. Bisbey II and D. Hollingworth, "Protection analysis: Final report. technical report isi/sr-78-13," University of Southern California Information Sciences Institute, Marina Del Rey, CA, Tech. Rep., 1978.
- [22] C. Landwehr, A. Bull, J. McDermott, and W. Choi, "A taxonomy of computer program security flaws," *Computing Surveys* 26, vol. 3, pp. 211–254, 1994.
- [23] T. Aslam, I. Krsul, and E. H. Spafford, "Use of a taxonomy of security faults," in *Proceedings of the 19th National Information Systems Security Conference*, pp. 551-560, 1996.
- [24] I. V. Krsul, "Software vulnerability analysis," Ph.D. dissertation, Purdue University, May 1998.
- [25] K. Tsipenyuk, B. Chess, and G. McGraw, "Seven pernicious kingdoms: A taxonomy of software security errors," *IEEE Security and Privacy*, vol. 3, no. 6, pp. 81–84, 2005.
- [26] H. H. Thompson and S. G. Chase, *The Software Vulnerability Guide*. Charles River Media, 2005.
- [27] M. Howard, D. LeBlanc, and J. Viega, *19 Deadly Sins of Software Security*. McGraw-Hill Osborne Media, 2005.
- [28] Forum Of Incident Response And Security Teams (FIRST), "Common vulnerability scoring system 2.0," 2007. [Online]. Available: <http://www.first.org/cvss>
- [29] P. Parrend and S. Frénot, "Classification of component vulnerabilities in Java service oriented programming (SOP) platforms," in *Conference on Component-based Software Engineering (CBSE'2008)*, ser. LNCS, vol. 5282/2008. Springer, October 2008.
- [30] MITRE Corporation, "Common weakness enumeration," <http://cwe.mitre.org>, 2009. [Online]. Available: <http://cwe.mitre.org/>
- [31] M. Bishop, "Vulnerabilities analysis," in *Proceedings of the Second International Symposium on Recent Advances in Intrusion Detection*, 1999, pp. 125–136.
- [32] C. V. Berghe, J. Riordan, and F. Piessens, "A vulnerability taxonomy methodology applied to web services," in *Proceedings of the 10th Nordic Workshop on Secure IT Systems (NordSec)*, Helger Lipmaa, Dieter Gollman, Ed., 2005.
- [33] N. Gruschka, M. Jensen, and N. Luttenberger, "A Stateful Web Service Firewall for BPEL," in *Proceedings of the International Conference on Web Services (ICWS)*, 2007, pp. 142–149.
- [34] MITRE Corporation, "Common Attack Pattern Enumeration and Classification (CAPEC)," <http://capec.mitre.org/>, 2009. [Online]. Available: <http://capec.mitre.org/>
- [35] —, "Open Vulnerability and Assessment Language (OVAL)," <http://oval.mitre.org/>, 2007. [Online]. Available: <http://oval.mitre.org/>
- [36] L. Lewis, "Towards automated risk identification in service-oriented architectures," in *Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI)*, 2008, pp. 253–254.
- [37] R. Accorsi, Y. Sato, and S. Kai, "Compliance monitor for early warning risk determination," *Wirtschaftsinformatik*, vol. 50, no. 5, October 2008.
- [38] R. Accorsi and C. Wonnemann, "Detective information flow analysis for business processes," in *Business Processes, Services Computing and Intelligent Service Management*, ser. LNI, vol. 147. GI, 2009, pp. 223–224.
- [39] G. Canfora and M. di Penta, "Service-oriented architectures testing: A survey," *Springer LNCS: Software Engineering: International Summer Schools, ISSSE 2006-2008, Salerno, Italy, Revised Tutorial Lectures*, vol. 1, pp. 78–105, 2009.
- [40] IT Governance Institute, "Cobit 4.1," 2007.
- [41] J. Gilligan, "Consensus audit guidelines," <http://www.sans.org/cag/>, 2009. [Online]. Available: <http://www.sans.org/cag/>
- [42] S. Höhn and J. Jürjens, "Rubacon: automated support for model-based compliance engineering," in *Proceedings of the 30th International Conference on Software Engineering (ICSE)*. New York, NY, USA: ACM, 2008, pp. 875–878.
- [43] R. Accorsi and T. Stocker, "Automated privacy audits based on pruning of log data," in *Proceedings of the International Workshop on Security and Privacy in Enterprise Computing*. IEEE Computer Society, 2008.