Vulnerability Effect Propagation in Service-Oriented Architectures

Lutz Lowis, Sebastian Höhn, and Maike Gilliot
Albert-Ludwig University of Freiburg
Institute of Computer Science and Social Studies, Department of Telematics
Friedrichstrasse 50, D-79098 Freiburg, Germany
{lowis, hoehn, gilliot}@iig.uni-freiburg.de

Abstract: Software vulnerabilities put automated business processes at risk. In service-oriented architectures (SOA), where business processes are implemented as potentially highly complex service compositions, the exploit of a software vulnerability can have far reaching effects on the confidentiality, integrity, and availability of business processes. In this paper, we report on our ongoing work which combines business process models with vulnerability information to automatically determine those effects. This determination is required in the identification phase in risk management.

1 Introduction

Implementing business processes by composing services of other services and applications is one of the main concepts in service-oriented architectures (SOA). Software vulnerabilities impose a security challenge on these compositions, because a vulnerable service or application can have the effect of making all dependent services vulnerable. In terms of the SOA layers displayed in figure 1, a software vulnerability in the application layer vertically cuts through the integration layer and affects the dependent services on the service layer. There, the effects can propagate horizontally between services, because services call each other and service compositions can appear as services (in figure 1, the service circle size reflects the degree of composition). Finally, the effects on the service layer influence the workflow. Concentrating on sofware vulnerabilities in the application layer and their effects on the service and orchestration layer, we ignore the infrastructure and presentation layer for now.

For a SOA-based business process, which *is* the workflow in SOA terms, this means that a single vulnerability can have effects on the confidentiality, integrity, and availability in various locations among each of these layers. In order to determine these security effects for a given vulnerability, two steps have to be taken (cf. figure 1). First, the vulnerability's precise location within the application layer (step 1a in figure 1) and the service layer (step 1b in figure 1) has to be identified. Second, the possible propagation paths of that vulnerability's exploitation have to be identified (step 2 in figure 1). These two steps ideally should be performed automatically, because the flexible and frequent changes a SOA allows within its possibly complex service compositions make the manual identification

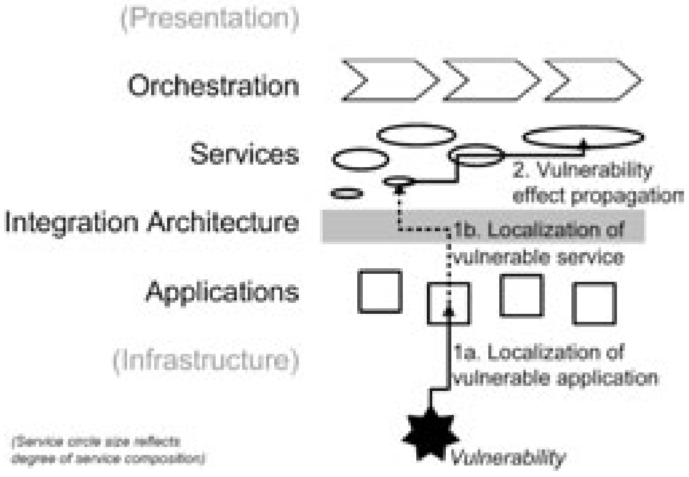

(Presentation)
Orchestration
Services
Integration Architecture
Applications
(Infrastructure)
2. Vulnerability effect propagation
1b. Localization of vulnerable service
1a. Localization of vulnerable application
Vulnerability
(Service circle size reflects degree of service composition)

Figure 1: SOA layers and vulnerability effect determination

too time consuming. With the proposed vulnerability effect identification approach, the steps in the identification phase of risk management can be automated through the combination of vulnerability descriptions (e.g., from vulnerability databases), business process models, and algorithms for determining the risk a vulnerability imposes on a business process. This allows frequent updates and, thus, risk management decisions based on current identification results.

The scenario we are considering is that of a company which implements an internal SOA. In that scenario, the company acts as service provider and service consumer (these and related terms are used as in [OAS06]) at the same time, and performs the analysis on the basis of its knowledge about the applications, services, and business process models. If the service provider and service consumer do not share that knowledge with each other, only a third party can determine the vulnerability effect propagation.

In the following section, we point out some technical challenges regarding the localization of an application vulnerability within a business process, and suggest a solution. The third section introduces the vulnerability effect propagation within a SOA in detail. Then, we motivate and present our method of determining the vulnerability effect propagation, a model-based identification approach. After discussing our approach using a propagation example, work related to our approach is discussed in section five. We conclude in section six and point out our future work.

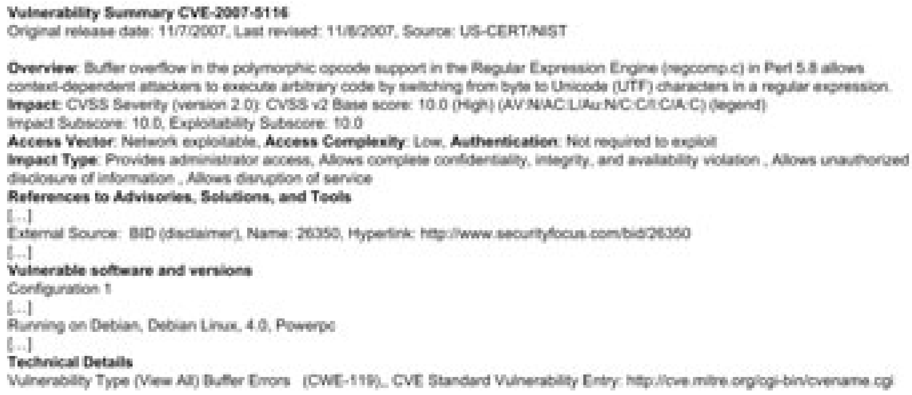
2 Locating vulnerabilities in a SOA

A few selected vulnerability databases and vulnerability description standards are presented, showing where and in which form vulnerability information can be obtained. While introducing some mapping challenges, which have to be overcome in the course of locating a vulnerability in a SOA, the types of security effects a vulnerability's exploit can have within a SOA are identified. Both the sources and mapping parts are put together by suggesting an automated localization solution.

2.1 Sources of software vulnerability information

Sources of vulnerability information such as the NIST National Vulnerability Database (NVDB, with US-CERT CVE entries) [Nat07], ISS X-Force [XF07], Security Focus [Sec07], and the Open Source Vulnerability Database (OSVDB; note that despite its name, the Open Source Vulnerability Database contains vulnerability descriptions of both open source and proprietary software) [Ope07] are updated daily and reflect the set of published, widely known vulnerabilities. Although there always might be unknown vulnerabilities waiting for their discovery, in the following we only consider published vulnerabilities, because we are not trying to find new vulnerabilities but to identify the effects of a given vulnerability's exploit.

Figure 2 shows a typical entry in the NIST NVDB; other databases use a very similar format. Besides the self-evident unique vulnerability id, common fields are a textual description, accessibility information (remote and/or local), as well as cross references to the according vulnerability entry in other databases. Without these references, it would be an error prone and time consuming task to find information on a certain vulnerability in different databases, simply because many databases use a custom naming scheme.


Vulnerability Summary CVE-2007-5116
Original release date: 11/7/2007, Last revised: 11/8/2007, Source: US-CERT/NIST

Overview: Buffer overflow in the polymorphic opcode support in the Regular Expression Engine (regcomp.c) in Perl 5.8 allows context-dependent attackers to execute arbitrary code by switching from byte to Unicode (UTF) characters in a regular expression.
Impact: CVSS Severity (version 2.0): CVSS v2 Base score: 10.0 (High) (AV:N/AC:L/Au:N/C:C/I:C/A:C) (legend)
Impact Subscore: 10.0, Exploitability Subscore: 10.0
Access Vector: Network exploitable, **Access Complexity**: Low, **Authentication**: Not required to exploit
Impact Type: Provides administrator access, Allows complete confidentiality, integrity, and availability violation , Allows unauthorized disclosure of information , Allows disruption of service
References to Advisories, Solutions, and Tools
[...]
External Source: BID (disclaimer), Name: 26350, Hyperlink: http://www.securityfocus.com/bid/26350
[...]
Vulnerable software and versions
Configuration 1
[...]
Running on Debian, Debian Linux, 4.0, Powerpc
[...]
Technical Details
Vulnerability Type (View All) Buffer Errors (CWE-119), CVE Standard Vulnerability Entry: http://cve.mitre.org/cgi-bin/cvename.cgi

Figure 2: An actual NIST NVDB entry

The Common Vulnerability Scoring Standard (CVSS, [RF07]) defines a set of metrics to score a vulnerability. These metrics describe not only the accessibility mentioned above, but also the confidentiality, integrity, and availability impact of a vulnerability on the affected application. It is important to note that this does not equal the impact on the business process which depends on the service composition. Some databases, for example, the NIST NVDB and ISS X-Force, include CVSS values in their entries (compare figure 2). Should the vulnerability description at hand not contain CVSS values, sometimes the cross references can be used to obtain these values from other databases.

Many vulnerability databases are online accessible and offer email notifications or RSS feeds (e.g., NIST NVDB and OSVDB). This provides the opportunity to always receive information on newly published vulnerabilities in a machine readable format.

2.2 Mapping challenges

Given a vulnerability from the sources and with the information described above, the question is not only *where* this vulnerability is located within the SOA, but also *what* its effects are on the confidentiality, integrity, and availability of the immediately affected service. When using web services, the business process is often modeled in UML, as Business Process Modeling Notation (BPMN) [Obj06] diagram, or described in a Business Process Execution Language (BPEL) [Org07] script. Both contain *services*, not applications. Since vulnerability descriptions typically refer to *applications*, there is a technical challenge of mapping vulnerabilities in applications to vulnerabilities in services. Referring to figure 1, a logical link between the applications, the integration architecture, and the services must be created, which allows to unambiguously identify the location of the vulnerability in both the application layer and the service layer. Implementing a solution to this challenge is one of our current tasks.

Another challenge lies in the mapping of a vulnerability's effect on the confidentiality, integrity, and availability of an application to the according effect on the dependent services and their outgoing messages (compare figure 3). Following a conservative estimation, the effects can be mapped one-to-one. This is certainly appropriate for availability, because if an application is not available, the dependent service will not be available, and the dependent service will usually not generate any messages while it is not available. However, regarding confidentiality and integrity, the mapping can become more complex. For example, harming the confidentiality of an application by reading a database's content does not necessarily imply insight into the dependent service (harming the confidentiality of the service), or being able to read all messages the service generates (harming the confidentiality of the messages). We are currently developing a mapping scheme for this challenge. For the rest of this paper, we assume the conservative one-to-one mapping. Please note that this mapping is different from the effect propagation discussed later, because the effect mapping takes place between one application and the directly affected services, whereas the effect propagation happens between a service and all dependent services.

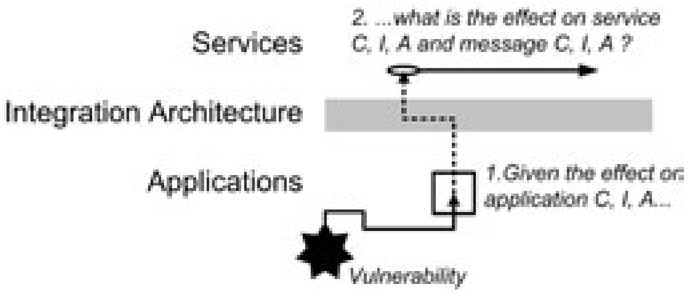

Services
2. ...what is the effect on service C, I, A and message C, I, A ?
Integration Architecture
Applications
1.Given the effect on application C, I, A...
Vulnerability

Figure 3: Mapping vulnerability effects to services and messages

2.3 Towards automated vulnerability localization

Based on the mentioned vulnerability information on the one hand, and using business process models with their web service composition description on the other, we suggest the approach depicted in figure 4, which is a refined version of the first step in figure 1. First, the affected application and the security effects on that application are extracted from a vulnerability description. Then, that precise application is localized within the application layer by using, e.g., application name and version number to differentiate between applications. In the next step, the vulnerability can be mapped through the integration layer to the service layer by means of a deployment diagram or similar information (grey area in figure 4). Besides the pure location in the service layer, the security effects are mapped according to a mapping scheme, which allows for a flexible mapping rather than the static one-to-one mapping.

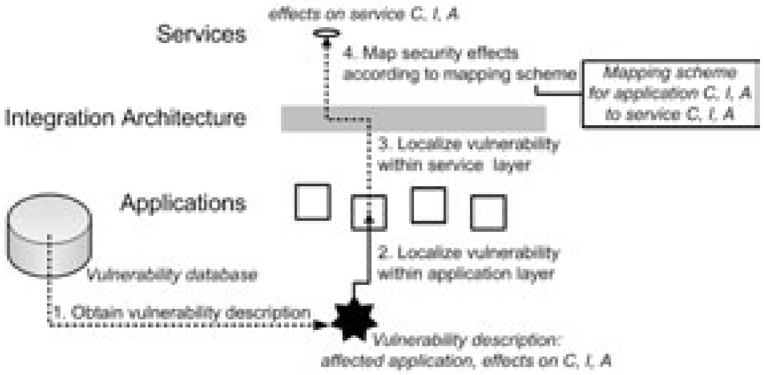

effects on service C, I, A
Services
4. Map security effects according to mapping scheme
Mapping scheme for application C, I, A to service C, I, A
Integration Architecture
3. Localize vulnerability within service layer
Applications
2. Localize vulnerability within application layer
Vulnerability database
1. Obtain vulnerability description
Vulnerability description: affected application, effects on C, I, A

Figure 4: Detailed localization approach

3 Model-based determination of vulnerability effect propagation

We argue for a model-based approach of identifying the vulnerability effects and their propagation. A propagation matrix is presented, allowing the derivation of a vulnerability's effect propagation paths, and a suitable model is developed, which allows to compute propagation paths based on a vulnerability's localization.

3.1 Utilizing SOA modeling advantages

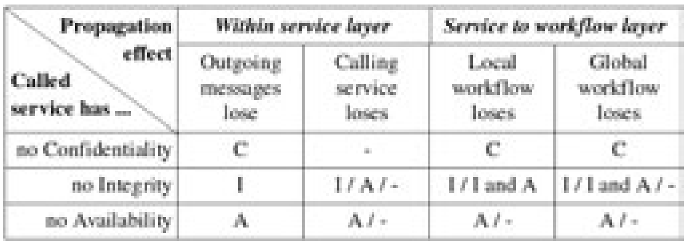
The effects a vulnerability's exploit has can be determined by penetration testing [Bis03] the production system. Because this can easily interrupt the production and incur high costs for restoring the production system, penetration tests on a duplicate system might seem to be a better solution. However, this still incurs costs for creating a duplicate system. Completely duplicating all web services might simply be infeasible, and separately testing single services does not allow an automated determination of the vulnerability effect propagation. A different solution is needed. Following the SOA approach, a model of the business process and its services is essential. If this model (e.g., in BPMN or BPEL) or a derivation thereof could be used to identify the business process impact of a vulnerability's exploit, this would allow for an inexpensive, realistic, and complete approach. Inexpensive, because the model is already available. Realistic, because the model itself will be executed in terms of BPEL scripts. Complete, because the whole composition would be examined rather than separate services. For these reasons, we suggest a model-based approach.

3.2 Vulnerability effect propagation matrix

Recalling the SOA layers in figure 1, at first vulnerability effects manifest in the application and service layer in a single location only. Taking into account the service composition, it shows that many services can be affected by a vulnerability in a single service. Also, vulnerable services can make the workflow vulnerable (orchestration layer in figure 1). To support the automated determination of a vulnerability's business process impact, it must be formalized how breaches of confidentiality, integrity, and availability propagate between and within the service and orchestration layers. We first define these effects on services, messages, and workflows, then suggest an according propagation matrix (see figure 5).

- Confidentiality (C).
  For services: the attacker does not know the exact inner workings of the service. Note that this implies that the service has not been replaced with an attacker's own version of the service.
  For messages: the attacker cannot read the message.

- Integrity (I).
  For services: the attacker cannot make the service produce arbitrary results.
  For messages: the attacker cannot change the message.
- Availability (A).
  For services: the attacker cannot stop the service from working.
  For messages: the attacker cannot delete the message.



Propagation effect / Called service has ...	***Within service layer***		***Service to workflow layer***	
	Outgoing messages lose	Calling service loses	Local workflow loses	Global workflow loses
no Confidentiality	C	-	C	C
no Integrity	I	I / A / -	I / I and A	I / I and A / -
no Availability	A	A / -	A / -	A / -

Figure 5: Vulnerability effect propagation matrix

The first column of figure 5 contains the effects a vulnerable service has on its outgoing messages: if a service loses its confidentiality, integrity, or availability, the messages that service creates are affected in the same way.

In the second column, the effects on the calling service are shown, which are none regarding confidentiality, because reading the input to a service does not tell the attacker how the service works. A breach of message integrity does only propagate if the calling service cannot detect that breach. If the breach is detected but cannot be fixed, the calling services availability is harmed. If the calling service is able to detect the integrity breach and fix it, the effect is none. Regarding availability, the effect on the calling service is none only if the breach is detected and a substitute service can be called.

The effects on the local and global workflow are displayed in column four and five, respectively. While confidentiality breaches propagate unconditionally, integrity and availability breaches only propagate as long as there is no mechanism which detects these breaches and either restore integrity or call substitute services to maintain availability.

Please note that the propagation matrix in figure 5 is based on very conservative assumptions, showing the worst case effect a vulnerability could have. In an actual SOA, the propagation depends on the system environment, meaning that the same vulnerability might have an effect in certain environments and none in others.

3.3 Model-based propagation determination

With the results of the vulnerability localization described in 2, the business process model and its service compositions given in the form of UML diagrams, and the above vulnerability effect propagation matrix, there is only one piece missing before the vulnerability effects and their propagation can automatically be determined. This missing piece is the information required to solve the case differentiation in the integrity and availability rows of the propagation matrix. To this end, we use parameterized UML stereotypes. BPMN diagrams or BPEL scripts can automatically be transformed to UML diagrams, which is why we opt for UML modeling

Currently, we use three tagged values, which can be attributed to services. “i:detect” means a service can detect if the integrity of incoming messages is harmed, “i:fix”-services can even fix changed messages. “a:maintain” means a service can maintain availability in spite of a single unavailable service by calling a substitute service.

Based on the vulnerability localization as presented in section 2.3, the effect propagation can now automatically be determined by creating a bottom-up propagation path. Starting with a service which is vulnerable according to the vulnerability localization, this service is assumed as “called service” and dependent services are regarded as “calling services” (cf. figure 5). Now, for each of the three possible effects (loss of confidentiality, integrity, and availability), the propagation matrix is combined with the value of the propagation stereotype, and the calling services are tagged correspondingly. Then, the procedure is repeated for every calling service in its new role as called service. In the end, all services which directly or indirectly depend on the vulnerable service have been rated. The resulting model then shows which services and workflow parts lose their confidentiality, integrity, or availability, and where. In a SOA, this reflects the business process impact of the examined vulnerability. Figure 6 is a refined version of the second step in figure 1, showing the propagation determination steps in detail.

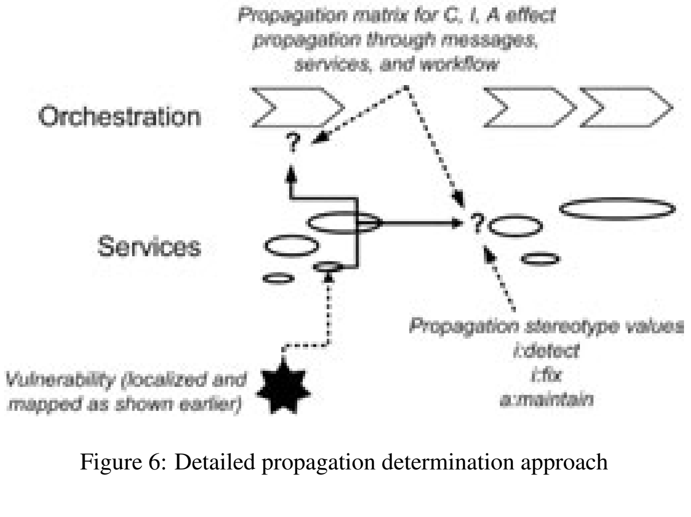

Propagation matrix for C, I, A effect propagation through messages, services, and workflow
Orchestration
?
Services
?
Propagation stereotype values
i:detect
i:fix
a:maintain
Vulnerability (localized and mapped as shown earlier)

Figure 6: Detailed propagation determination approach

4 Discussion

An exemplary SOA instance is presented and used to discuss some selected aspects of the approach introduced above.

4.1 SOA-based business process example

In a SOA-based business process, services are used by multiple other services to implement specific parts of the business process. Assuming the buffer overflow vulnerability from figure 2 is present in the database application of an order service (cf. figure 7), an attacker could execute arbitrary code, harming the confidentiality, integrity, and availability of the application. Following the conservative application-to-service mapping discussed in section 2.2, the dependent order service could lose its confidentiality, integrity, and availability as well. The shipping service and the inventory management service, which also use that order service, would be affected as described by the vulnerability effect propagation matrix in section 3.2. Depending on whether the attacker harmed the confidentiality of the database application, the shipping and inventory management service would lose their confidentiality, too. Integrity and availability breaches might not propagate all the way up to the workflow, because a subsequent service can detect and restore them, which is not the case for confidentiality.

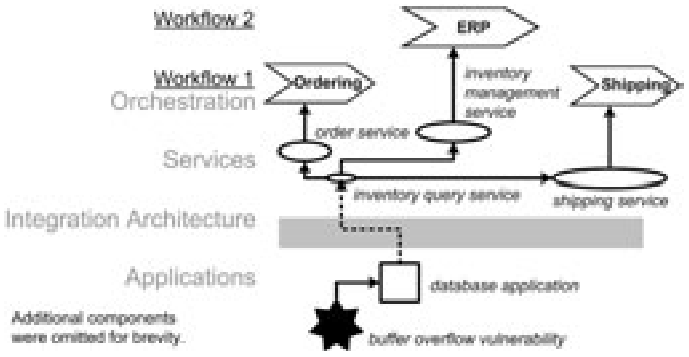


Figure 7: Vulnerability effect propagation example

4.2 Selected aspects

Recall that the CVSS values regarding the vulnerability impact type (cf. section 2.1, especially figure 2) describe the vulnerability effect on an *application* and not on services, let alone the workflow. When mapping the effects from the application to the service layer, and later from the service to the workflow layer, a distinction must be made between two different analysis types. The determination of the more theoretical, conservatively estimated effects of a specific vulnerability's exploitation shows what (in the worst case) *could* happen. Trying to find the most probable effects, i.e., taking specific details of the IT environment into consideration, is supposed to show what (in the most likely case) *would* happen. While for the first kind of analysis, the one-to-one mapping mentioned in section 2.2 is sufficient, the second kind requires a mapping which accounts for conditions the IT environment or even the execution context sets.

Just like the actual effect of an application vulnerability on the depending service, the effect of vulnerable services on other services or the above workflow might be weaker or stronger depending on the IT environment. This increases the complexity of the analysis, because a single, unconditional propagation matrix (see figure 5) might be too coarse to yield exact results. While it is theoretically possible to define a special propagation matrix for each IT environment, in practice this would severely hinder the analysis because of the high number of different IT environments and the frequency with which the IT environment might change (e.g., due to updates and patches).

5 Related Work

Vulnerability taxonomies such as [Krs98] and [BRP05], depending on their focus, offer a comprehensive explanation of how vulnerabilities come into existence, which different vulnerability types exist, and which attacks can be performed through them. While a suitable taxonomy of web service vulnerabilities could be used to develop a vulnerability effect mapping scheme, to our knowledge at this time no such taxonomy of web service vulnerabilities and their effects on web services exist.

Vulnerability description standards such as the Application Vulnerability Description Language [Org04], VulnXML [Ope02], and the [Cor07], focus on describing *how* a vulnerability can be exploited. This is useful when testing for or trying to fix a vulnerability. However, regarding the *effect* of a vulnerability's exploit, the Common Vulnerability Scoring Standard values mentioned in section 2.1 are more relevant.

Powerful analysis approaches and tools exist to check source code or running applications for vulnerabilities (e.g., [MLL05], [CM04], [WKP05]). The vulnerabilities found through such analysis tools, after being published in a vulnerability database, serve as input to our approach.

The attack graph generation approach in [OBM06] can be used to find new attacks in enterprise networks. Again, the attacks found through this approach, or rather the involved vulnerabilities, provide the vulnerability information input to the business process impact determination we have presented.

6 Conclusion and Future Work

Business processes implemented in the notion of service-oriented architectures (SOA) are workflows on – possibly highly complex – service compositions. Therefore, breaches of the confidentiality, integrity, and availability of the underlying services can affect these business processes. The underlying services can in turn be affected by vulnerabilities in the applications they are based on. Identifying the security impact of such software vulnerabilities on business processes is required for the identification phase in risk management approaches, or vulnerability management as in [Gor07]. With the approach at hand, the security effects a given vulnerability has on services, and also how these effects propagate through the service composition to the global workflow, can automatically be determined. In SOA terms, the global workflow *is* the business process, thus our approach will identify the security impact of software vulnerabilities on SOA-based business processes.

To achieve this goal, we will at first implement the application identification described in section 2.2. For a given vulnerability, the affected applications within a SOA will thus be identified. Then, the application-to-service mapping scheme also mentioned in section 2.2 will have to be defined. At the same time, we will examine which type of (UML) model is best suited for the localization and propagation determination steps introduced in sections 2.3 and 3.3, respectively. Currently, activity diagrams and deployment diagrams seem to be the best choice. Also, the propagation matrix in figure 5 has to be extended to allow for a more detailed view on vulnerability effect propagation. Finally, we will check whether including the infrastructure and presentation layer (cf. figure 1) is a valuable widening of the approach presented here or not.

References

[Bis03] Matt Bishop. *Computer Security*. Addison-Wesley, Pearson Education, Boston, USA, 2003.

[BRP05] Chris Vanden Berghe, James Riordan, and Frank Piessens. A Vulnerability Taxonomy Methodology applied to Web Services. In Helger Lipmaa, Dieter Gollman, editor, *Proceedings of the 10th Nordic Workshop on Secure IT Systems (NordSec 2005)*, 2005.

[CM04] Brian Chess and Gary McGraw. Static Analysis for Security. In *IEEE Security and Privacy*. IEEE Computer Society, 2004.

[Cor07] MITRE Corporation. Open Vulnerability and Assessment Language (OVAL), 2007.

[Gor07] Vlad Gorelik. One Step Ahead. In *ACM Queue*. ACM, 2007.

[Krs98] Ivan Victor Krsul. *Software Vulnerability Analysis (Ph. D. Thesis)*. Purdue University, 1998.

[MLL05] Michael Martin, Benjamin Livshits, and Monica 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*, San Diego, California, USA, 2005.

[Nat07] National Institute of Standards and Technology. National Vulnerability Database, 2007.

[OAS06] OASIS. Reference Model for Service Oriented Architecture 1.0, 2006.

[Obj06] Object Management Group (OMG). Business Process Modeling Notation (BPMN), 2006.

[OBM06] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A Scalable Approach to Attack Graph Generation. In *Proceedings of the Conference on Computer and Communications Security*, Alexandria, Virginia, USA, 2006. ACM.

[Ope02] Open Web Application Security Project (OWASP). VulnXML, 2002.

[Ope07] Open Source Vulnerability Database (OSVDB). Open Source Vulnerability Database (OSVDB), 2007.

[Org04] Organization for the Advancement of Structured Information Standards (OASIS). Application Vulnerability Description Language (AVDL), 2004.

[Org07] Organization for the Advancement of Structured Information Standards (OASIS). Business Process Execution Language (BPEL), 2007.

[RF07] Forum Of Incident Response and Security Teams (FIRST). Common Vulnerability Scoring System, 2007.

[Sec07] SecurityFocus. SecurityFocus Vulnerability Database, 2007.

[WKP05] Sam Weber, Paul A. Karger, and Amit Paradkar. A Software Flaw Taxonomy: Aiming Tools At Security. In *Software Engineering for Secure Systems*, St. Louis, Missouri, USA, 2005. ACM.

[XF07] IBM Internet Security Systems X-Force. Alerts and Advisories, 2007.