A method for analyzing security of SOA-based systems

Qifei Lu and Zhishun Wang

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden
This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Qifei Lu
E-mail: qifei.lu@gmail.com

Author(s):
Zhishun Wang
E-mail: willeye@gmail.com

University advisor(s):
Dr. Ludwik Kuzniarz
School of Engineering

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : + 46 457 271 25
Abstract

SOA-based systems offer high degree of flexibility and interoperability. However, the securing of SOA-based applications is still a challenge. Although some related techniques have been proposed and presented in academia and industry, it is still difficult to check SOA quality in security aspect from an architecture view.

In this thesis project, a method for security analysis in SOA is introduced and investigated. The method intends to be used for analyzing security of SOA-based systems on architecture level. To demonstrate the method, a prototype supporting the method is introduced and implemented. And the method and prototype are also evaluated respectively based on Technology Acceptance Model. The evaluation result shows that the prototype supporting the method is a promising inspection tool to detect software vulnerability.

Keywords: Software Security, SOA, Security Analysis
Contents

1 Introduction ................................................. 7
  1.1 Aims and Objectives .................................... 7
  1.2 Research Questions ................................... 8
  1.3 Research Methodology ................................. 8
  1.4 Structure of the Thesis .............................. 10

2 Related Foundations ..................................... 11
  2.1 SOA Security ........................................... 11
    2.1.1 Security Taxonomy ................................ 11
    2.1.2 SOA Security Standards .......................... 14
    2.1.3 SOA Security Patterns ............................ 14
  2.2 Technology for Security Analysis ..................... 15
    2.2.1 Static Analysis Techniques ....................... 16
    2.2.2 Dynamic Detection Techniques ................... 16
    2.2.3 SAVE Tool ....................................... 17

3 Introduction of the Method ................................ 18
  3.1 Introduction ........................................... 18
    3.1.1 Extraction Process ............................... 19
    3.1.2 Knowledge Base Model ............................ 20
    3.1.3 Identification Process ........................... 22
    3.1.4 Analysis Process ................................ 23
  3.2 Preliminary Reflection Results ....................... 24

4 Prototyping .................................................. 26
  4.1 Requirement ........................................... 26
    4.1.1 Purpose ........................................... 26
    4.1.2 Elicitation Method ............................... 26
    4.1.3 Functional Requirements ......................... 27
    4.1.4 Non-Functional Requirements .................... 27
  4.2 Used example: Restaurant Application ................ 35
  4.3 Two Possible Solutions ................................ 35
  4.4 Design and Implementation ........................... 37
    4.4.1 System Model ..................................... 37
    4.4.2 Knowledge Base Model ............................ 38
    4.4.3 Identification Process Model ..................... 39
  4.5 Testing ................................................ 40

5 Evaluation .................................................. 44
  5.1 Aims and Method ...................................... 44
  5.2 Evaluation Instruments ................................ 44
  5.3 Population Sampling ................................... 45
  5.4 Execution and Data Collection ....................... 46
6 Discussion of Evaluation Results
   6.1 Main Results
   6.2 Validity Threats
      6.2.1 Internal Validity
      6.2.2 External Validity
      6.2.3 Construct Validity
      6.2.4 Conclusion Validity
   6.3 Further Research

7 Conclusions

8 Discussion and Future Work
   8.1 Discussion
   8.2 Future Work

References

Appendices
   A Source Code of Library Sample
   B Questionnaire
   C Source Code of Prototype

List of Figures
1 Research methodology
2 Taxonomy of SOA information security
3 Method description
4 Extraction process
5 Example of security tag - stateful service
6 Identification process
7 Development process: rapid prototyping
8 Library example in EMF
9 Code generation for library example in EMF
10 Restaurant Sample based on SCA
11 Approach A – Extraction Process
12 Approach B – Extraction Process
13 System model
14 Knowledge base model
15 Identification process model
16 An excerpt of the system model
17 A tag example of Knowledge base
18 An excerpt of the running results
19 Identified results of restaurant example
List of Tables

1. Research questions and research methodology ........................................ 9
2. Industry standards for SOA security .................................................. 14
3. Message protection pattern ............................................................... 15
4. Weighting scale ................................................................................... 24
5. Initial functional requirements ................................................................ 27
6. Comparison of two solutions for the method ......................................... 37
7. Questionnaire used ................................................................................ 45
8. Overview of participants ......................................................................... 47
9. Questionnaire results for the method .................................................... 47
10. Questionnaire results for the prototype ............................................... 47
11. Summary of usefulness and ease of use for the method and prototype .... 48
12. Statistics of PEOU & PU for the method and prototype ....................... 48
1 Introduction

In recent years, to meet the demands of the business and the market, Service-Oriented Architectures (SOA) [16] are widely applied in various areas. However, security is important to any software system, even more critical in SOA-based system due to the SOA inherent characteristic of interoperability and flexibility [16]. The current situation is that on one hand, SOA systems are required to achieve the goals of higher flexibility, interoperability and reusability; on the other hand, software system become more and more complicated and easy to lead security problems as more third-party services were introduced into the SOA systems. Therefore building secure SOA system and checking security quality of the system become increasingly important as they have tremendous influence to company, organization, infrastructure, etc. From the developer’s perspective, there is also a need to ensure code security, which means the code implemented in the repository does not diverge from their intended architecture. Unfortunately, checking quality of security aspect of software system is still not an easy task [12] [13] [27] [30] [56].

In order to address the problems of checking quality of security aspect in SOA-based system, this thesis project was planned to investigate a method for analyzing security in SOA, and it is executed at a research institute A (Since the method proposed by researchers from the research institute is not published yet, the name of the institute and method will not be revealed and the research institute will be referred to as institute A). The method is proposed within European 7th Research Framework Program MASTER [17], which intends to provide methodologies and infrastructure that facilitate monitoring, enforcement, and auditing of security compliance in SOA. Institute A has a worldwide reputation in empirical software engineering, and they use established security and reliability analysis techniques to support the system designer in designing reliable and secure software-based systems. This thesis introduces, develops on and discusses the method for detecting security aspects of existing service oriented systems on architectural level.

1.1 Aims and Objectives

The main aim of the thesis project is to introduce, demonstrate, prototype and evaluate the method.

To fulfil the aims, the following activities are performed:

• Carrying out a literature research to identify the existing security methods and criteria in SOA-based systems

• Identifying and demonstrating the detailed process of the method

• Implementing a prototype that supports the security identification process of the method

• Performing evaluation of the method and prototype
1.2 Research Questions

The research is planned to be carried out according to the following research plan. The first step aims at achieving a good understanding of the SOA paradigm, security analysis criteria in SOA context and the proposed method itself. Next, a prototype of the method for the identification process is implemented. After that, an evaluation of the method and prototype is performed.

According to the research plan and steps, the following questions are proposed:

- **Understanding phase:**
  - RQ1: What is the current state of SOA security?
  - RQ2: What are the preliminary reflection result of a method for security analysis in SOA?

- **Prototyping phase:**
  - RQ3: How to implement a prototype supporting the security identification process of the method?

- **Evaluation phase:**
  - RQ4: How to evaluate the method and prototype?
  - RQ5: What are the evaluation results of the method and prototype?

The expected outcomes of the thesis project are:

- An understanding of security in SOA context
- An introduction of the method and preliminary reflection results
- A prototype for supporting the security identification process of the method
- An evaluation and evaluated outcome of the method and prototype

1.3 Research Methodology

According to the research plan and questions, the research methodology is structured in table 1 as follows: First, a detailed and comprehensive literature study is carried out to gather material related to SOA security for RQ1. A field study in institute A provides preliminary results to answer RQ2. To answer RQ3, a literature study is performed and a prototype is developed in institute A. After that, an evaluation is designed for RQ4, and the result of evaluation is presented and discussed to answer RQ5.
Table 1: Research questions and research methodology

<table>
<thead>
<tr>
<th>Research Questions</th>
<th>Research Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1: What is the current state of SOA security?</td>
<td>Literature review/study. A detailed and comprehensive literature search is carried out to gather material. The literature study encompasses the material as written down in articles, books and web references.</td>
</tr>
<tr>
<td>RQ2: What are the preliminary reflection results of the method?</td>
<td>Field study in Institute A. Literature study on the method</td>
</tr>
<tr>
<td>RQ3: How to implement a prototype supporting the security identification process of the method?</td>
<td>Field study and prototyping in Institute A</td>
</tr>
<tr>
<td>RQ4: How to evaluate the method and prototype?</td>
<td>Survey (Questionnaire)</td>
</tr>
<tr>
<td>RQ5: What are the evaluation results of the method and prototype?</td>
<td>statistical analysis and reflection</td>
</tr>
</tbody>
</table>

Figure 1 is the simplified research flow chart, which shows the relationships between research questions and research methodology.
1.4 Structure of the Thesis

The rest of thesis is structured as follows. Section 2 is an overview of background and related foundations, and section 2 introduces the method in detail and presents the preliminary reflection results. Then section 4 implements a prototype supporting the method. After that, section 5 conducts an evaluation for the method and prototype and section 6 presents and discuss the evaluation results of the method and prototype. Finally, section 7 gives a conclusion and section 8 directs the further work.
2 Related Foundations

This section explains how security is understood in SOA context. It discusses the related work in this research field, i.e. security taxonomy, SOA security patterns/standards, existing technologies for security analysis. As SOA is mainly driven by big industry giants, i.e. SAP, IBM, Oracle, Microsoft, etc, and different companies have their own preference of definition or terminologies in SOA domain, It is important to identify a commonly accepted SOA security taxonomy. There are also numerous SOA standards and patterns proposed by industry and academia, it is also important to summarize from security perspective. After that, popular technologies to analyze security is presented and discussed.

2.1 SOA Security

This section firstly introduces the SOA security taxonomy needed for further research and discussion. Based on the taxonomy, the common security goals are identified and linked to security attributes in the taxonomy. To achieve the security goals, two major categories of security solutions (SOA security standard and pattern) are explored and discussed.

2.1.1 Security Taxonomy

The terminology related to information security is confusing; depending on the background and perspective of a person the meaning of terms differs. So there is a need to identify and aggregate a terminology commonly accepted by the academic and industry. The first comprehensive information security model was presented by John McCumber [38]. The author introduced Security services, Information states, Security countermeasures as the basic three aspects of information assurance, and claimed that information should satisfy the three main goals of security, which were confidentiality, integrity, and availability. According to the description, McCumber’s work is a security framework on a high-level abstraction.

The work of McCumber was extended and refined by Maconachy in [36] and Abe Usher in [53], which was intended to create a taxonomy that graphically depicts the relationships of these three base aspects from [38] and provide a taxonomy which is sufficiently detailed for application with real world problems. The authors embodied authentication and non-repudiation as security services, besides confidentiality, integrity, and availability.

Based on thesis previous work, the authors in [44] introduce a taxonomy of information security, intended for the use of software architects of service oriented systems. This taxonomy is more appropriate compared with the ones mentioned in [36] [38] [53] as it fits well in SOA context. The security taxonomy for SOA includes five facets: security assets, security attributes, security threats, security solutions, and security metrics, which are illustrated in Figure 2.
• **Security assets** are resources to be protected. It could be private data like configuration/personal information, or communication messages between communication channel, source code of some security-critical system, etc.

• **Security attributes** are abilities to avoid unauthorized users accessing a system or prevent the system from attack and keep in secure state. They can be categorized as confidentiality, integrity, availability, non-repudiation, and accountability, which are shown on level 2 of figure 2.

  – **Confidentiality** is the ability to prevent information from unauthorized disclosure.

  – **Integrity** is the ability to prevent information from unauthorized or illegal modification.

  – **Availability** is the ability to keep the system operable and available, i.e. prevention of unauthorized withholding of information.

  – **Accountability** is the responsibility to provide certain specific functionality.

  – **Non-Repudiation** is the ability to provide the proof for the origin and integrity of data.

• **Security threats** are failures, errors and faults of services. Fault can cause error, either internal (i.e. vulnerabilities) or external (i.e. attacks). Errors can lead to failures which prevent services from providing correct functionalities. Matt Bishop and David Bailey in [4] presents a precise definition and classification of vulnerability taxonomy based on studies in [3] [31]. The work in [2] presents an overview of software vulnerabilities, and different security threats modeling techniques are proposed and discussed, i.e. “Security Goal Indicator Trees” (SGIT) [41], “Vulnerability Cause Graph” (VCG) [5], “Security Activity Graph” (SAG) [6], “Attack Trees” [45], “Software security analysis and assessment model” [57], etc.
• **Security solutions** provide countermeasure to security threats. They are categorized into three kinds: *control mechanisms*, *control properties*, and *failure resistance*. Control mechanisms are security management practices, like identification, authentication, access control, etc. Control properties are standard approach to implement countermeasures, such as key formats (i.e. public key infrastructure X.509 [24]), protocols (i.e. SSL), and credentials (i.e. user name/password logic and certificates). Failure resistance can be prevention, tolerance, removal and forecasting of faults.

• **Security metrics** are metrics used to measure the soundness of countermeasures to security threats. As different systems have different security goals, there is no one-fit-all security metrics. According to the work from [43] [54] [40], security metrics depends on security goals, organization context, and practical needs, they could either be quantitative or qualitative, objective or subjective. For example, Common Criteria [26] is a security metric measuring the capability of a product from specification to implementation. Based on the measurement, the product vendors can claim how well do their products in fulfilling specific security attribute.

As far as we know, the taxonomy of information security is the initial SOA security ontology. The security taxonomy contributes to establish a uniform understanding of SOA security. In addition, this security taxonomy is also appropriate for software architects to specify security requirements and design secure software architecture [44]. According to the security attributes defined in the taxonomy, the SOA security goals can be defined as how to achieve these security attributes in certain degree. As security attributes are not independent (for example confidentiality attribute may have an effect on integrity or Non-Repudiation attribute), the main goals of SOA security are categorized as follows:

• **Authentication** is to allow service’s access only to the intended application that invokes the service. In traditional security approaches, this can be realized as a standard credential based security such as a login username/password pair, certificates etc.

• **Authorization** is to grant service’s access only to a defined set of services. This is the classical role-based security mechanism to restrict access to a subset of functions.

• **Integrity** is to check the integrity to ensure that the data is not changed or corrupted.

• **Confidentiality** is to ensure that service is accessible only to those authorized to have access.

• **Federation** an extension of authentication that helps the service provider to establish trust between the provider’s security domain and an external domain. When a service requires authentication against another external system, federation is used.
2.1.2 SOA Security Standards

As SOA is driven mainly by industry, there are a number of standards for SOA security. Different industrial companies may have different goals and their specific understanding and standard for SOA. The SOA security standards commonly accepted and used are summarized in table 2 inspired by [16].

Table 2: Industry standards for SOA security

<table>
<thead>
<tr>
<th>SOA Security Goal</th>
<th>Standards</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>WS-Security</td>
<td>A standard to specify information like username/encrypted password in XML-based communication protocol like Simple Object Access Protocol (SOAP)</td>
</tr>
<tr>
<td></td>
<td>WS-Trust</td>
<td>An extension of WS-Security by providing additional information like security token</td>
</tr>
<tr>
<td></td>
<td>WS-Secure Con-</td>
<td>An extension of WS-Security by creating a secure conversation context</td>
</tr>
<tr>
<td></td>
<td>versation</td>
<td></td>
</tr>
<tr>
<td>Authorization</td>
<td>XACML</td>
<td>XACML (eXtensible Access Control Markup Language) is an XML schema for specifying authorization policy</td>
</tr>
<tr>
<td>Confidentiality/Integrity</td>
<td>XML Encryption</td>
<td>A W3C standard for encrypting XML contents</td>
</tr>
<tr>
<td></td>
<td>XML DSig</td>
<td>A W3C standard for digital signatures</td>
</tr>
<tr>
<td>Availability</td>
<td>WS-ReliableMess-</td>
<td>A standard to enable reliable message exchange between communication partners</td>
</tr>
<tr>
<td></td>
<td>aging</td>
<td></td>
</tr>
<tr>
<td>Federation</td>
<td>SAML</td>
<td>SAML (Security Assertion Markup Language) is an XML-based schema for authenticating across various domains</td>
</tr>
</tbody>
</table>

2.1.3 SOA Security Patterns

A pattern is regarded as certain commonly accepted best practice/strategy for certain type of problem and packaged in a reusable manner. It can be revised and improved over time. A security pattern is considered as a solution to a
recurring security problem [19] [22]. Security patterns provide a convenient way to reuse expert knowledge, and many methods/tools/frameworks to build secure SOA systems are based on security patterns, i.e. a pattern-driven security process for SOA applications [11], “A methodology to build secure systems using patterns” [18], “A Service-Oriented Framework for Quantitative Security Analysis of Software Architectures” [34], “A pattern-driven security advisor for service-oriented architectures” [46].

There are 218 existing security patterns summarized by Thomas Heyman in [21], i.e. “access controller pattern”, “message authentication pattern”, “Boundary defense pattern”, “Message protection pattern”, etc. The following part gives an example of the Message protection pattern. There are two important aspects, one is “Data confidentiality”, the other is “Data origin authentication and integrity”. In each aspect the goal/problem and possible solutions are defined accordingly. The detail is shown in the table 3 from [23]:

| Goal | “How can a service verify that a message originates from a known sender and that the message has not been tampered in transit?” |
| Problem Description | Processing layers may expose sensitive data when security is limited to point-to-point protocols |
| Solution | A message can be digitally signed by using digital signature algorithm |
| Implementation Guidelines/example | . . . |
| Impacts | Use of cryptographic techniques can add to performance requirements and the choice of digital signing algorithm can affect the level of security actually achieved. |

| . . . | . . . |

2.2 Technology for Security Analysis

This section discusses the existing technologies for security analysis. Although there are a number of different techniques existing, they can be mainly categorized as static analysis techniques and dynamic detection techniques. In addition, a tool developed by institute A is introduced. It can be used to detect architecture violation, and is extended to support features like security analysis in the future.
2.2.1 Static Analysis Techniques

Static techniques are those applied directly to the source code without running the application. The objective is to evaluate or get specific information directly from the source code without executing it. There are different techniques to perform static inspection, i.e., Guided Checklists [15], Pattern Matching, Lexical Analysis, Parsing Type Qualifier, Data Flow Analysis and Model Checking [58].

Software quality of security can be improved by many tools according to source code inspection, i.e., “CP-Miner” [32], “MOPS” [7], “ESC/Java” [9], “Checkstyle”, “FindBugs”, “Programming Mistake Detector (PMD)” etc [52]. PMD is a popular and commonly used one in open source community. It can also be integrated to other development environment, i.e., Eclipse. PMD is a static analysis tool for detecting bugs in the source without running the system. First of all, a rule set is defined in a factory. These rules can be used to detect the bugs and vulnerabilities in the source code. New rules can be added to the factory and old rules can also be tailored to fulfill the specific needs of user. There are already many public Rule Sets defined in [52] based on best practices, i.e., design rules, naming rules, type resolution rules, security checking rules etc.

Static inspection provides an approximation of runtime behavior. However, the static solution may suffer from imprecision: some of the statically derived answers may be false positives, despite the effort to improve static analysis precision [35]. Another problem of static analysis is scalability. A person auditing an application by hand often quickly gets lost given the number of possibilities they need to consider. As a result, while simple errors will possibly be detected, some of the more complex errors stemming from sources located “far apart” in the program may remain unnoticed.

2.2.2 Dynamic Detection Techniques

In order to dynamically detect vulnerabilities it is necessary to execute the program code, and then analyze the runtime behavior of the system and give a verdict. There are different techniques to perform runtime analysis, i.e., Fault Injection, Fuzzing Testing, Dynamic Taint, Sanitization, etc [58].

There are several benefits of runtime analysis [35]:

- Even the most precise conservative static analysis may suffer from false positives in the worst case. If the user does not want to deal with false positives generated from the static checker, runtime analysis is a good alternative.

- No need to change the development life cycle. While some organizations have an established software development process, which includes well-defined development and testing phases, others do not. As a result, requiring developers to adopt a static analysis tool is not a viable option at those organizations. However, a runtime tool does not require much involvement on developers’ part.
• No need for the source code. Unlike the static approach, runtime analysis does not require changes to the original program and does not need to access the source code. Runtime analysis can be especially advantageous when dealing with applications that rely heavily on libraries, whose source code is unavailable. In those cases, vulnerabilities that span library code can not be easily detected by static approaches.

There are also a number of commercial or open source tools (i.e. “Daikon invariant detector” [25]) available supporting dynamic runtime analysis. In summary, static analysis techniques analyze the source code without running the application while dynamic techniques perform the analysis by run the program code. The selection of the tools is depend on the type of application to evaluate, the programming language and the type of vulnerability to detect. The static techniques cover all possible execution paths but require the source code while dynamic techniques have the difficulty of requiring the preparation of test cases and the possibility that not all paths in the program are covered, but the advantage that the problems if any, are found in the running code [58].

2.2.3 SAVE Tool

The Software Architecture Visualization and Evaluation (SAVE) [39] [14] can be used as a reverse engineering tool to detect architecture violations. SAVE tool provides various features, such as architecture conformance checking, architecture metrics collecting, and commonality and variability comparison. Some new features like security analysis, run-time analysis is planned to be introduced in future version of the tool.

One of the main features is architecture compliance checking. It compares the architecture extracted from source code with the intended architecture which is pre-defined by architect or user. So it can automatically check whether an application conforms to a planned architecture and provide the visualization results, such as convergence, divergence, and absence metrics.

As the tool is based on Eclipse IDE and plug-in architecture, it is extensible and flexible to add new plug-ins. SAVE tool mainly includes three components: generators, analyzer and visualization component. Generators can extract fact information from source code of applications with the help of JDT (Java Development Tools), and build the system model which includes all the software artifacts needed in further steps. The Analyzer can analyze the system model and add additional information to the original model based on analyzing and computation. It’s like a model-to-model (M2M) transformation. Finally the visualization component can visualize the previous generated and analyzed models. The results can be graphs or reports, depending what the users or developers want.
3 Introduction of the Method

This section firstly introduces the method which was investigated in the thesis. The method tries to provide some support to the gap that there is no support for security engineers in the analysis and assessment of security requirement on architecture level of SOA based systems. After introducing the method, results of a preliminary reflection on the method based on initial field study at institute A are presented.

3.1 Introduction

The method is intended to be used for detecting security vulnerabilities based on security goals of SOA-based system. The method can be divided into three main processes and one knowledge base (see figure 3). In the first extraction process, SOA system’s source code and configuration/definition files are parsed out by parsers. The output of extraction process is a system model which contains all the information of those artifacts and their relations among them. Before the second identification process, security experts set up security goals and checking rules in the knowledge base model and performance matching check. The inputs of identification process are system model and knowledge base model, and the output is identified model. Finally, based on the output of second process, security analysis views/reports are displayed in analysis process. The detail approach is introduced in the following sections.
3.1.1 Extraction Process

The aim of the extraction process is to parse the SOA system source codes and configuration files and set up their connection relations. The input of extraction process is SOA system, including all the source code and configuration/definition files. Although currently it only supports Java based SOA systems, it intends to be extended to other programming language based SOA systems soon, i.e. .NET, Business Process Execution Language (BPEL), etc. The source code are Java classes, implementing some interfaces, services, etc. These information are parsed out by Java parser and the output are stored in the system model (see figure 4 right). The configuration/definition files consists of the composition information of these services and components, providing manifestation of how services are implemented and how system is composed. They are parsed out by configuration parser and the output are stored in the system model (see figure 4 left). In the middle of figure 4, there is a connector. It provides the explicit
map of services to implementation java classes. The map is also stored in the system model.

In summary, the whole system artifacts are parsed out and stored in the system model. The system model contains system architecture and all the information of those artifacts and their relations. The whole process of extraction can be seen in figure 4.

Figure 4: Extraction process

3.1.2 Knowledge Base Model

The knowledge base model is similar to a database which provides support for setting various security goals and rules. Security experts can continue to enrich the knowledge base by adding new elements, i.e. security artifacts, security tags, security indicators, etc. Setting of rules in knowledge base is a point of view of top-down approach. The highest level is security goals/indicators, and then under each goal/indicator there are several security tags which represent security behavior and constraints of system and are linked to security artifacts. The detail descriptions are as follows:

- **SOA Security Goals and Indicators**

  SOA Security Goals present SOA security aims, like Authentication, Authorization, and Confidentiality, etc described in section 2. How to achieve
these security goals? Indicators are imported as various metrics to answer how security goals are fulfilled. Each security goal can contains one or more indicators.

- **Security Artifacts**
  A software artifact is a piece of software (i.e. documents, codes, models/descriptions, etc). An artifact related to security aspect is called a security artifact. In knowledge base, a security artifact can be assigned to one or more security tags. For example, security artifact can be annotations, import, calls, or other relation type.

- **Security Tags**
  Security tags can be regarded as a combination of some security requirements in a specific aspect of the system. For example, in an online system, some business functionalities are required as “Stateful service”, whereas some other functionalities are required as “Stateless service”. Therefore, this kind of requirement can be designed as a security tag. Similar to the data structure, a security tag can be seen as a tag tree linking to security artifacts. And there are four basic logical operators defined in tag structure: OR, AND, XOR and NOT. XOR means only one condition can be chosen from the sub branches. For AND, OR and NOT, each operator has the same meaning in logical definition. Each artifact is assigned a weight number to show the importance of the artifact in the tag. The following figure 5 shows what a security tag “stateful service” looks like.

![Figure 5: Example of security tag - stateful service](image)

The circle in figure 5 is the tag name. The diamond are logical operators. The rectangles are security artifacts. The numbers are the weight of each artifacts. As shown in the figure, the “stateful service” tag contains two logical operators and five security artifacts. Two artifacts and an XOR operator are under the operator AND. The artifact “Annotation: @Service” means an annotation should be checked in the source code. The
annotation starts with “@” in Java and the type is “Service”. The artifact “Import: org.osoa.annotations.Service” means an import text should be checked in the source code. In this example, the imported class is “org.osoa.annotations.Service”. Under the operator XOR, there are three similar artifacts. Take the artifact “Annotation: @Scope(“Conversion”)” as an example. It means an annotation should be checked in the source code. The annotation starts with “@” and the type is “Scope”. The text “Conversion” in the parentheses is the parameter of this annotation.

3.1.3 Identification Process

After the extraction process, the identification process starts. There are two main inputs in this process (see figure 6), one is the system model produced from the extraction process, another is the knowledge base model which contains security checking rules and defined by experienced security experts. The mainly purpose of this process is to identify the security artifacts according to security rules and then tag and group them. As we known, in a SOA system, not all software artifacts are security artifacts, therefore the first step is to have a knowledge base which defined security checking rules, tags and security artifacts; then the next step is to identify security artifacts and mark them from system model according to the rules defined in knowledge base (see the sub-process “identify security artifacts” in figure 6). Third step is to map security artifacts to security tags (see the sub-process “map & tag artifacts” in figure 6). The output of identification process is identified model. It contains information like architecture, connection between tags and artifacts, and identified security tags, etc.
In summary, there are two inputs (system model and knowledge base model), two sub-process (“identify security artifacts” and “Map & tag artifacts”), and one output (identified model). The whole identification process can be seen in figure 6.

3.1.4 Analysis Process

The analysis process of the method is still under development. The idea is that based on the result of the identification process, security experts can perform various analysis operations and get the corresponding views or reports with the help of some internal visualization tool i.e. SAVE tool. The possible operations are: weighting the identified system model (e.g. calculating the probability of the mapped artifacts and tags), merging or filtering security tags, etc.

To support the further weighting analysis, a weighting metric is imported. Each artifact can be assigned with a weight number in ordinal scale. The scale of 1-5 is described in the following table 4 inspired by the work of thomas [60].
Table 4: Weighting scale

<table>
<thead>
<tr>
<th>Scale</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>very low probability</td>
</tr>
<tr>
<td>2</td>
<td>low probability</td>
</tr>
<tr>
<td>3</td>
<td>median probability</td>
</tr>
<tr>
<td>4</td>
<td>high probability</td>
</tr>
<tr>
<td>5</td>
<td>very high probability</td>
</tr>
</tbody>
</table>

The probability percentage of a tag is computed by the ratio between the sums of weighting of detected security artifacts (called ‘numerator’) and the sum of weighting of all security artifacts (called ‘denominator’) within the tag.

Currently, the calculation rules originated from [17] are done according to the following formula:

$$probability = \frac{\sum_{i=1}^{n} isP_i \times w_i}{\sum_{i=1}^{n} w_i}, w_i \in \{1, 2, 3, 4, 5\}, isP_i \in \{0, 1\}$$

Note: $isP_i = 1$ means this artifact is matched, $isP_i = 0$ means not. When the logical operator is ‘XOR’, the calculation is a bit different, the explanation is as follows [17]:

- **OR/NOT/AND**: All weighting of the elements in an OR, NOT or AND group affect the overall probability. They are summed up to the denominator. The weighting of all found elements are also summed up to the numerator. The probability is calculated according to the formula mentioned above.

- **XOR**: If only if one subordinated element of a XOR-group is found, the probability is 1. Otherwise the probability is 0.

### 3.2 Preliminary Reflection Results

The field study of the method took one month in institute A. During the investigation and observation of the method, a number of question were asked, from general questions to technology specific questions, i.e. “Who may benefit from the method?”, “Why do you plan to reuse certain internal tool?”, “Is it one system model enough for both extraction and identification process?”, etc. To answer the questions with respect to the method, several face-to-face meetings are scheduled. Most of the questioned are answered or clarified by the method initiators and security experts in institute A. However, as the method is still under development, some questions are still not answered, i.e. “What are the exact detail steps of analysis process of the method?”
According to the field study, the preliminary reflection results of the method are as follows:

- Software architect, quality assurer, tester and security engineer can benefit from the method. The method can help stakeholder detect security vulnerabilities hidden in the source code, i.e. violation of secure coding rules, violation of architecture pattern, etc. So it also can be integrated into testing phase of software development life cycle.

- The method is a static analysis approach based on the architecture extracted from source code. It is a semi-automatic software inspection process focusing on security aspects. The method takes the software architecture, which is reverse-engineered from source code, and security expertise from security experts as inputs. The output is security analysis reports and views.

- It can not analyze the dynamic run-time security behaviour of the system. As the method is a static analysis method, it provides an approximation of runtime behaviour. However, as mentioned in section 2, the static solution may suffer from imprecision: some of the statically derived answers may be false positives. There maybe some possible improvement potential about precision enhancement of knowledge base. To avoid the false positives, user need retrospect the code to decide whether the problems really exist or not.

- One distinct component of the method is knowledge base model. Security expertise from security experts can be reused and packaged in a proper way in the model. However, more specific security context rules are needed to be defined. For example, some semantic information is missing in the rule definition. The user can not easily apply the rule in certain scenario unless he/she is experienced or a security expert.

- It is coupled with Software Architecture Visualization and Evaluation (SAVE) tool [39] [14]. As an internal reverse engineering tool in Institute A, SAVE has already been applied in some projects. Researchers inside institute A are familiar with it and already have some experience of how to use it. In addition, SAVE can already generate part of information needed to be stored in the system model of the method. Although there will be some integration effort, reuse SAVE is promising.

- In order to demonstrate the usage of the method, a prototype/tool supporting the method is necessary. Although the method is still under development and not mature, it has shown some potential for security analysis in SOA. A prototype/tool supporting the method can also help elaboration of the method, and proof its correctness and goodness. To gain reliable confidence, an evaluation is also needed to be conducted for the method and prototype/tool. The evaluation results may also help researchers make proper correction or improvement for the method.
4 Prototyping

To better understand the method and demonstrate its practical usage, a prototype supporting the method is necessary. This section elicits the requirement of the prototype. After that, two possible solution for the prototype are proposed, and the pros and cons of each approach are discussed and compared. Based on the current situation and constraints, the appropriate solution is selected. Then the system design for main process of the method is provided for the chosen solution. Based on the design, the prototype is implemented and tested in an running example.

4.1 Requirement

Requirements engineering is one of the most important and challenging tasks in software engineering [10] [20] [50]. Requirement serves as the the basis for discussion of system features, and it identifies what is needed to be designed and implemented. Therefore, in the following sections, the purpose of the requirement, the method used to elicit the requirement, functional and non-functional requirements are presented and discussed.

4.1.1 Purpose

According to section 3, the prototype supporting the method shall help user to inspect all the source files in SOA system and identify all the security artifacts based on security rules defined in knowledge base. The prototype will be implemented in Java and restricted to analyze systems that based on the Service Component Architecture (SCA) [49] paradigm, which is initialized by Open Service Oriented Architecture Consortium and can provide a powerful and flexible way to build SOA-based applications.

4.1.2 Elicitation Method

Before starting the requirement elicitation activities, we have to identify the stakeholders. As the method is still under development and not published yet, only people who are involved on the method are our main stakeholders (i.e. method initiators, security experts in institute A). Potential sources of requirements include stakeholders and internal reports.

There are many requirement elicitation methods (i.e. interviewing, workshop, focus groups, brainstorming, etc) can be used to understand the real needs of customer [50]. As each method has advantages and disadvantages, what we choose are a combination of these techniques to best gather requirements according to current situation. The methods used are as follows:

- **Documentation Review**: we reviewed the internal reports, documents, and relevant materials of the method during the field study. Based on these documents and materials, we have a clear idea of what functionalities shall the prototype provide.
• Observation: we observe the daily tasks (i.e. security inspection) of the method initiators, and ask question about the task and how they plan to use this prototype to make their daily work easier.

• Brainstorming: we scheduled several meeting with researchers and security experts. A number of ideas are created during the brainstorming (i.e. adding new attribute scope to a security tag to better distinguish and understand tags, reusing and integrating existing SCA model to system model, etc). We present initial demo/scenarios, then stakeholders come up with ideas and approaches.

4.1.3 Functional Requirements

As the method is still under development, the requirements are subject to change. After requirement elicitation, the main initial functional requirements identified are as shown in table 5.

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Requirement Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR1</td>
<td>The system shall extract data from source code and configuration/definition file in SOA system and store results in a system model.</td>
</tr>
<tr>
<td>FR2</td>
<td>The system shall create a knowledge base model based on the security expertise from security experts. It shall include security goals, security tags, and security artifacts.</td>
</tr>
<tr>
<td>FR3</td>
<td>The system shall identify the security artifacts from the system model based on the security tags in knowledge base model and store results in identified model.</td>
</tr>
<tr>
<td>FR4</td>
<td>The system shall calculate the probability of a component belonging to certain security tag based on weighting of identified security artifacts.</td>
</tr>
<tr>
<td>FR5</td>
<td>The system shall generate a security view/report of the SOA system based on identified results from FR4.</td>
</tr>
</tbody>
</table>

4.1.4 Non-Functional Requirements

1. Operational Requirements

As mentioned previously in section 3, the method is not mature and still under development. As a result the suitable software development process we chosen is rapid prototyping [48] based on model-driven software development process (see figure 7). After initially functional requirements
are elicited and gathered, there are iterations during the process. During each iteration, the most important set of requirements are selected, designed, implemented, tested. After integration, release and use of updated version of the prototype, the prototype will be improved according to the feedback, new identified requirements and rest of unimplemented requirements:

![Diagram of Development Process: Rapid Prototyping](image)

**Figure 7: Development process: rapid prototyping**

2. Software/Platform Requirements
Model-Driven Development (MDD) [29] has been adapted and integrated into the popular open source platform Eclipse in a project called Eclipse Modeling Framework (EMF) [51]. EMF uses the facilities provided by Eclipse. The usual steps of using EMF are: 1) Developer or Architect defines the system in Ecore model, which is a subset of UML model; 2) Developer uses the code generation feature to generate the code automatically; 3) Developer tailors the code to fulfill the specific requirements of
the system. It can help developers to build Java applications based on a UML model quickly, especially appropriate in rapid prototyping. The following part will show the usage of EMF with an excerpt in a library example.

(a) Create model in EMF

![Diagram of EMF model](image1)

**Figure 8: Library example in EMF**

Developer can define the model in EMF with class, reference, etc. For example, the library class has an attribute ‘name’ and a containment reference ‘books’ shown in figure 8. Each book has a title with a number of pages, and belongs to a category. The BookCategory is an enumeration relation.

(b) Generate Java code for model

![Code generation for EMF model](image2)

**Figure 9: Code generation for library example in EMF**
EMF can generate model code by clicking on the “Generate Model Code” feature (see figure 9) based on the defined model. For example, the code generated for library UML class are Interface Library.java and class LibraryImpl.java.

**Interface Library.java**

```java
package example;
import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.ecore.EObject;
/**
* model extendedMetaData="name='books'"
* generated
*/
public interface Library extends EObject {
/**
* see #setName(String)
* see example.ExamplePackage#getName()
* model extendedMetaData="name='books'"
* generated
*/
String getName();
/**
* see #getName()
* generated
*/
void setName(String value);
/**
* see example.ExamplePackage#Books()
* model containment="true"
* generated
*/
EList<Book> getBooks();
} // Library
```

**LibraryImpl.java**

```java
/** * <copyright>
* </copyright>
* * Id
* /
package example.impl;
import example.Book;
import example.ExamplePackage;
```
import example.Library;
import java.util.Collection;
import org.eclipse.emf.common.notify.Notification;
import org.eclipse.emf.common.notify.NotificationChain;
import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;
import org.eclipse.emf.ecore.util.EObjectContainmentEList;
import org.eclipse.emf.ecore.util.InternalEList;

/**
* The following features are implemented:
* 
* </p>
* **generated**
*/

public class LibraryImpl extends EObjectImpl implements Library {

/**
* The default value of the 'link #getName() <em>Name</em>' attribute.
*/

protected static final String NAME_EDEFAULT = null;

/**
* The cached value of the 'link #getName() <em>Name</em>' attribute.
*/

protected String name = NAME_EDEFAULT;
/**
 * The cached value of the 'link #getBooks() <em>Books</em>' containment reference list.
 */
protected EList<Book> books;
/**
 * generated
 */
protected LibraryImpl()
{
    super();
}
/**
 * generated not
 */
protected EClass eStaticClass()
{
    return ExamplePackage.Literals.LIBRARY;
}
/**
 * generated not
 */
public String getName()
{
    return "Prefix." + name;
}
public void setName(String newName) {
    String oldName = name;
    name = newName;
    if (eNotificationRequired())
        eNotify(new ENotificationImpl(this, Notification.SET, ExamplePackage.LIBRARY_NAME, oldName, name));
}

public EList<Book> getBooks() {
    if (books == null) {
        books = new EObjectContainmentEList<Book>(Book.class, this, ExamplePackage.LIBRARY_BOOKS);
    }
    return books;
}

public NotificationChain eInverseRemove(InternalEObject otherEnd, int featureID, NotificationChain msgs) {
    switch (featureID) {
    case ExamplePackage.LIBRARY_BOOKS:
        return ((InternalEList<?>)getBooks()).basicRemove(otherEnd, msgs);
    }
    return super.eInverseRemove(otherEnd, featureID, msgs);
}

public NotificationChain eInverseAdd(InternalEObject otherEnd, int featureID, NotificationChain msgs) {
    switch (featureID) {
    case ExamplePackage.LIBRARY_BOOKS:
        return ((InternalEList<?>)getBooks()).basicAdd(otherEnd, msgs);
    }
    return super.eInverseAdd(otherEnd, featureID, msgs);
}

/* */
*/
*/
*/
*/
*/
*/
*/

*/
*/
*/
*/
public Object eGet(int featureID, boolean resolve, boolean coreType) {
    switch (featureID) {
    case ExamplePackage.LIBRARY_NAME:
        return getName();
    case ExamplePackage.LIBRARY_BOOKS:
        return getBooks();
    }
    return super.eGet(featureID, resolve, coreType);
}

/**
 * <!- begin-user-doc ->
 * <!- end-user-doc ->
 * generated
 */
SuppressWarnings("unchecked")
public void eSet(int featureID, Object newValue) {
    switch (featureID) {
    case ExamplePackage.LIBRARY_NAME:
        setName((String)newValue);
        return;
    case ExamplePackage.LIBRARY_BOOKS:
        getBooks().clear();
        getBooks().addAll((Collection<? extends Book>)newValue);
        return;
    }
    super.eSet(featureID, newValue);
}
} //LibraryImpl

The all generated code can be seen in the appendix A of all the source code of the library example.

(c) **Tailor the generated code**

When developer wants to modify or tailor the generated code, he/she can use the annotation `@generated not` to distinguish the tailored code with the generated code. If the model defined is changed or updated, developer can generate the code again based on revised model. But the customized code will not be replaced (see the method “getName” in LibraryImpl.java). As a result, using EMF tool can improve productivity and efficiency. The tailored code can be seen in the appendix A.
4.2 Used example: Restaurant Application

This section introduces a SCA based restaurant sample used during the development of prototype. Service Component Architecture (SCA) [49] provides a new programming model for building applications based on Service Oriented Architecture. SCA provides a model for the composition of services from inside or third party service and components. In SCA, a composite file defines the configuration of an SCA domain. It describes the components, services, references, and the connections between them. A component may consist of one or several services/references. It also contains the information with respect to how the component is implemented. The component may expose the services to other components, and it also may depend on some other services provided by other components.

![Figure 10: Restaurant Sample based on SCA](image)

This sample provides a simple SCA composite named Restaurant (see figure 10). This composite is a composition containing three components:

- **RestaurantServiceComponent** shows the list of proposed menu.
- **BillServiceComponent** calculates the bill of a customer for a chosen menu.
- **PayServiceComponent** withdraws the money from the account provided by the customer.

4.3 Two Possible Solutions

Various solutions are possible for the prototyping. For example it can be prototype either from scratch, or based on some standard platforms i.e. SOA Tools Platform (STP) [42], or by reusing existing tool i.e. SAVE. Our main considerations are limited time period for prototyping without losing quality, especially the maintainability, modifiability, and extensibility of the tool. The following
paragraphs show two discussed examples of possible solutions w.r.t. the implementation of the method.

- **Approach A**

  Approach A is based on SAVE and STP. As a brief introduction, this approach uses SCA tools and SAVE to parse the SOA configurations and java source code separately in extraction process (see figure 11). After that an additional connector is used to map the SCA services in configuration to java class types.

- **Approach B**

  Figure 11: Approach A – Extraction Process

  Figure 12: Approach B – Extraction Process
Approach B is based on STP. The main difference between Approach B and A lies only in the first extraction process of the method (see figure 12). This approach reuses tools from STP, i.e. SCA parser, SCA introspection tool etc, to parse the configuration and source code of SOA applications. The comparison of approach A and B are summarized in table 6. SAVE can already parse out the source code level artifacts, and it also has been applied in some Institute A’s projects. Hence according to comparison of advantages/disadvantages of the two approaches, approach A is chosen as the solution for the prototyping of the method more fits to current situation.

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Approach A</strong></td>
<td>- enable to reuse internal tool SAVE’s functionalities (i.e. source code extraction, component visualization);</td>
</tr>
<tr>
<td></td>
<td>- is easy to understand by the researchers as they were already familiar with this tool and had some experience of using it;</td>
</tr>
<tr>
<td></td>
<td>- need less time to develop;</td>
</tr>
<tr>
<td></td>
<td>- is coupled with SAVE, and need effort to integrate;</td>
</tr>
<tr>
<td></td>
<td>- still need effort to add new features (i.e. parsing SCA configuration file, adding new relation types like annotation, exception, etc);</td>
</tr>
<tr>
<td><strong>Approach B</strong></td>
<td>- is independent of internal tool SAVE;</td>
</tr>
<tr>
<td></td>
<td>- is more flexible and extensible;</td>
</tr>
<tr>
<td></td>
<td>- is totally new approach and technology used, no experience to learn/resue, high risk;</td>
</tr>
<tr>
<td></td>
<td>- need more time to explore and develop.</td>
</tr>
</tbody>
</table>

### 4.4 Design and Implementation

The following parts present the design of system model, identification process, and knowledge base model. After that, they are implemented based on model-driven development.

#### 4.4.1 System Model

As mentioned before, the system model includes all necessary information for the security analysis, i.e. architecture, relations between components and services, etc. The system model is an extension of existing model of SAVE. As shown in figure 13, a connector named as SCAFSConnector is added to the model. It
helps to connect the Services from SCA Model to FSType, which are java class types. The SCAFSConnector can contain one or more SCAToFSTypeMapEntry, which is a map. The key of the SCAToFSTypeMapEntry is Service, which is from SCA Model. The value of the SCAToFSTypeMapEntry is FSType, which is from SAVE’s FSModel.

Figure 13: System model

4.4.2 Knowledge Base Model

The figure 14 shows the knowledge base model. It supports to store and display data in a tree structure. The key classes are Artifact, LogicalOperator, Tag, etc. Artifact can be extended to several artefact types like Import, Call, Annotation, etc.
4.4.3 Identification Process Model

As two main inputs of identification process, the system model and knowledge base model can be found in figure 15. In the middle of figure, a connector named as FSKBConnector is added. It is a connector for establishing the connection between system model and knowledge base model. The FSKBConnector can contain one or more FSKBRelationConnection, which is a map. There are existing source artifacts from system model. According to the security artifacts defined in knowledge base model, method getFSRelationsByArtifact() in FSKBConnector can be called to establish FSKBRelationConnections. In the
FSKBRelationConnection, the key is the artifact defined in knowledge base, and the value is the FSRelation from system model. The Grouping is a mapping to group the security artifacts to security tags and tag the IdentifiedComponents, which are extended from SAVEComponent. The SAVEComponent is a main component defined in SAVE identified model. It also contains tagging information, which contains information like tag and probability, etc.

![Identification process model](image)

Figure 15: Identification process model

After system model, knowledge base model are defined and identification process is modeled, the implementation is based on EMF model-driven development. The source code of the prototype can be found in appendix C.

4.5 Testing
This section provides the prototype testing and outcome with an running restaurant example. The prototype was applied and used in the sample application. Finally, the result of the testing is presented.
Based on the security goals in section 2, the confidentiality is defined in the knowledge base of the method. According to the security goal/indicator, the communication channel between Bill Service Component and Payment Service Component should fulfill the confidentiality goal (i.e. client information is not exploited by third-party). One way to avoid third-party compromisation is to make the Bill Service Component stateful, Payment Service Component stateless and communication channel encrypted. Then there is no need to store state information (i.e. context information, sensitive client information, etc) inside the Payment Service component. In this test, the prototype is executed to check what can be identified from the source code.

According to section 3, the first process of the prototype is to extract system artifacts and relations from the source code and configuration files. The result of this process is a system model. An excerpt of the system model can be seen in figure 16. From the figure, human can see the concrete software artifacts in the model. For instance, each line shows the attributes of each artifact i.e. name, id, relation type (i.e. import, call, or annotation), etc.

Next step is the identification process. Before the identification process starts, the knowledge base is defined by security experts. It contains some security related tags (i.e. Tag1: stateful service tag, Tag2: stateless service tag in figure 17):

Figure 16: An excerpt of the system model

Next step is the identification process. Before the identification process starts, the knowledge base is defined by security experts. It contains some security related tags (i.e. Tag1: stateful service tag, Tag2: stateless service tag in figure 17):
As shown above, there are two tags existed in knowledge base model. In this testing, three security artifacts are identified by the identification process based on security checking rules and system model. From the running result in figure 18, two artifacts belonged to Restaurant Service Component are linked to Tag1 and one artifact belonged to Restaurant Service Component is linked to Tag2. The probabilities of the mapped artefacts in each tag are calculated respectively according to their weights defined in security tag and the weighting formula (see section 3.1). For instance, according to the data output from figure 18, the probability of Tag1 and Tag2 are calculated as follows based on checking rules defined in figure 17:

Probability of Tag1 = \((1+5) / (1+1+5) = 86\%\)

Probability of Tag2 = \((1) / (1+1+1) = 33\%\)

However, the value of probability is a theoretical reference value based on the security experts' experience, if the users want to make a precisely judgment, he/she needs perform backtracking of source code or configuration settings to inspect if any fault could be hidden or missed.
Based on the result of identification process, the components in the restaurant sample can be tagged. For example, Payment Service Component is tagged as "Stateful Service". Similarly, the Bill Service Component is tagged as "Stateless Service", and the communication channel between them is annotated as "confidentiality". Based on these outcomes shown in figure 19, security staff can make a judgment whether the security goal is fulfilled or not.

![Figure 19: Identified results of restaurant example](image)
5 Evaluation

This section presents the evaluation of the method and prototype respectively. As an empirical research, main steps of empirical study are followed according to [28] [47]. The method and prototype are evaluated through a questionnaire by seven people. The detail of the planning, design, execution and data collection of the evaluation are presented in the following sections.

5.1 Aims and Method

As the method is a new security analysis method in SOA, and it is not mature and the prototype supporting the method is under development, the aim of the evaluation is to evaluate the method and prototype tool from the usability perspective.

Technology Acceptance Model (TAM) [55] has been validated (in terms of validity and robustness) as a powerful evaluation framework, and it is suitable for helping users to decide whether they accept/use a new technology. According to the evaluation aim, TAM was chosen as the measurement instrument. A number of factors could influence users’ decision about how and when they will use it, notably they are [10]:

- **Perceived usefulness** (PU) - This was defined by Fred Davis as “the degree to which a person believes that using a particular system would enhance his or her job performance”.

- **Perceived ease-of-use** (PEOU) - This was defined as “the degree to which a person believes that using a particular system would be free from effort”.

There are many approaches to conduct evaluation and execute TAM, i.e. survey/questionnaire, interview, observation or experiment [59]. However, due to the current status of the method and prototype, we need to quickly and easily get information from people who are interested in the method. According to [8], software engineering survey/questionnaire is an appropriate and fast method for data collection.

5.2 Evaluation Instruments

Different versions of TAM have been proposed in past years [1]. In this case, we extend the initial author Davis’s Model, questions of Ease of use and Usefulness. According to the TAM’s principles, three important aspects are used in this evaluation instrument (i.e. a questionnaire) (see table 7): Ease of use, Usefulness and Potential improvement. The numbers under each question from A1 to B4 are ranked on a five-point Likert scale (1. Strongly disagree; 2. Disagree; 3. Neither agree nor disagree; 4. Agree; 5. Strongly agree) [33]. The score of the aggregate metrics perceived usefulness and ease of use is defined as the average score of each answer.
Table 7: Questionnaire used

<table>
<thead>
<tr>
<th>ID</th>
<th>Question</th>
<th>Ease of use</th>
<th>Usefulness</th>
<th>Improvement potential</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 2 3 4 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>Is the method easy to understand?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>Is it easy to learn (from cost/time perspective)?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>Is it easy to use?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B1</td>
<td>Does it work as the method described?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td>Can it save time and improve effectiveness comparing to checking by human?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td>Do the security rules enable to help detect and analyze security?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B4</td>
<td>Is it a useful tool for security vulnerabilities inspection of SOA systems?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>Any problems hidden in it?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>What is missing in current method?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>What can be improved in future?</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.3 Population Sampling

Sampling is a very complex issue in evaluation [37], and our sampling in this evaluation also has some constraints. Convenience sampling is selected during the evaluation as accessibility is the major factor limiting the possibility to get access to large samples. The populations in this evaluation are people who will use this method, i.e. samples who work or study in SOA or security field, or who develop software in organizations, or who are interested in method. As the methods is still under development and not mature, the population is not very big as not many people currently know it and can access this method. So the sample was very close to the entire population as the number of people who are familiar with the method is limited.
5.4 Execution and Data Collection

Firstly, an introduction of the method was given to the participants. During this step, participants can also review the introduction document of the method. Questionnaires were sent out to the participants. Then the participants were asked to fill the questionnaire based on the conceptual impression/reflection of the method. After that, the prototype supporting the method is demonstrated to the participants. Based on the practical usage of prototype demo, the participants were asked again to fill the questionnaire. The detail process is shown in the figure 20 bellow.

In total, 14 (7*2) questionnaires were received after execution of evaluation. An overview of participants’ information is described in table 8. In this survey, three types of people were involved. The names of participants and involved companies will not be disclosed, for the sake of confidentiality. Two of them are professional software engineers who have working experience in SOA domain. Three of them are master/PHD students who have work experience in
software engineering. The rest two are students studying in computer science and involved actively in SOA and security areas.

Table 8: Overview of participants

<table>
<thead>
<tr>
<th>Number</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Software Engineer</td>
</tr>
<tr>
<td>3</td>
<td>Student, with 2-3 year work experience in SE</td>
</tr>
<tr>
<td>2</td>
<td>Student majored in IT, without work experience</td>
</tr>
</tbody>
</table>

The table 9 and table 10 show the questionnaires data collected from seven responded participants. All of the questionnaires were filled and collected with both authors being present during the evaluation process. The original questionnaires can be found in appendix B.

Table 9: Questionnaire results for the method

<table>
<thead>
<tr>
<th>Participants</th>
<th>Questions</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td></td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>A2</td>
<td></td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>A3</td>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>B1</td>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B2</td>
<td></td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td></td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B4</td>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Table 10: Questionnaire results for the prototype

<table>
<thead>
<tr>
<th>Participants</th>
<th>Questions</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td></td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>A2</td>
<td></td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>A3</td>
<td></td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>B1</td>
<td></td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B2</td>
<td></td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>B3</td>
<td></td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B4</td>
<td></td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
6 Discussion of Evaluation Results

This section presents the main results of the evaluation for the method and prototype. The collected data are analyzed and interpreted. After that, the validity threats are stated and discussed according to [59]. In the end, the further research for the evaluation is proposed.

6.1 Main Results

The table 11 shows the summary of each average results of ‘ease of use’ and ‘usefulness’ for the method and prototype. The results show that the method is easy to understand from the participants’ perspective.

Table 11: Summary of usefulness and ease of use for the method and prototype

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>3.00</td>
<td>3.67</td>
<td>3.33</td>
<td>2.67</td>
<td>3.33</td>
<td>2.67</td>
<td>3.00</td>
<td></td>
</tr>
<tr>
<td>Prototype</td>
<td>3.33</td>
<td>4.33</td>
<td>3.67</td>
<td>3.00</td>
<td>4.00</td>
<td>3.33</td>
<td>3.67</td>
<td></td>
</tr>
</tbody>
</table>

A box plot analysis was conducted for PU and PEOU results of the method and prototype, and the statistical results are shown in table 12. It was found that the medians of PEOU and PU are improved by 0.67 and 0.25 respectively. The data in table 12 are also visualized in figure 21.

Table 12: Statistics of PEOU & PU for the method and prototype

<table>
<thead>
<tr>
<th></th>
<th>Median</th>
<th>Average</th>
<th>1st Quartile</th>
<th>Upper Limit</th>
<th>Lower Limit</th>
<th>3rd Quartile</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEOU_method</td>
<td>3.00</td>
<td>3.10</td>
<td>2.83</td>
<td>3.67</td>
<td>2.67</td>
<td>3.33</td>
</tr>
<tr>
<td>PU_method</td>
<td>3.25</td>
<td>3.07</td>
<td>2.88</td>
<td>3.25</td>
<td>2.75</td>
<td>3.25</td>
</tr>
<tr>
<td>PEOU_prototype</td>
<td>3.67</td>
<td>3.67</td>
<td>3.33</td>
<td>4.33</td>
<td>3.33</td>
<td>3.83</td>
</tr>
<tr>
<td>PU_prototype</td>
<td>3.50</td>
<td>3.57</td>
<td>3.38</td>
<td>4.00</td>
<td>3.25</td>
<td>3.75</td>
</tr>
</tbody>
</table>

From the evaluation process and results, it can be seen that the evaluation was executed properly and it achieved the aims. According to the above statistic analysis, the median of PEOU and PU of the method is around 3, it indicates that the participants are not strong agree or disagree with the method.
interesting result is that after practical usage and experience of the prototype, participants almost gave a higher number than they did in the method which is based on conceptual impression.

![Box plots of PEOU & PU for the method and prototype](image)

Figure 21: Box plots of PEOU & PU for the method and prototype

### 6.2 Validity Threats

This section discusses the validity threats of this research. The discussion is based on the classification scheme of validity threats in [59], which includes four categories: internal validity, external validity, construct validity, and conclusion validity.

#### 6.2.1 Internal Validity

The internal validity [59] in this case is concerned with maturation and instrumentation. As time past, participants could be more familiar with the method and give a positively comment during the evaluation process. Another threat is instrumentation design. After the initial questionnaire form was designed, a software engineer student was invited to make a pilot test about whether the questionnaire is clear and understandable. Based on the feedback, an improved and refined questionnaire was produced.
6.2.2 External Validity

The population selection could be a threat to external validity [59]. This threat is caused by not having representative sample of the population. In the evaluation the sample was very close to the entire population as the number of people who are familiar with the method is limited due to the fact that the method is still under development and not published. To minimize this threat, evaluation on larger sample is planned in future version of the method and prototype.

6.2.3 Construct Validity

Construct validity are concerned with whether a scale measures the construct appropriately [59]. In this research, three threats for this category were identified: hypothesis guessing, evaluation apprehension, and experimenter expectancy.

Hypothesis guessing occurs when participants in a study try to figure out what the purpose and intended result of the study is. To minimize this threat, the participants were selected in convenience sampling based on individual’s competencies and knowledge.

Evaluation apprehension occurs when people are afraid of being evaluated. To minimize this threat, the participants received warrant anonymity.

Experimenter expectancy is caused by the experimenter who wants to achieve some results based on what he/she expected. To minimize this threat, the questions in the survey are formulated in a neutral way.

6.2.4 Conclusion Validity

Conclusion validity is concerned with issues that may affect the ability to draw the correct conclusions according to the performed study [59]. In this research, four threats for this category were identified: fishing, reliability of treatment implementation, random irrelevancies in experimental setting, random heterogeneity of subjects.

Fishing is concerned with the issue that the researcher may search for a specific result from the research. To minimize this threat, all questionnaire results from the participants, no matter positive or negative, were collected and analyzed.

Reliability of treatment implementation is concerned with the application of treatments to subjects. To minimize this threat, all the participants performed the evaluation in a same occasion.

Random irrelevancies in experimental setting are caused by disturbing elements outside the study setting. To minimize the threat, the survey is conducted in an uninterrupted environment.

Random heterogeneity of subjects is caused by a heterogeneous sample such that individual differences could affect the study’s result. To minimize this threat, the participants were selected in convenience sampling based on individual’s competencies and knowledge.
6.3 Further Research

As the method is not mature and the prototype/tool supporting the method is still under development, the reported evaluation should be considered as a preliminary or pilot evaluation, which will be further enhanced and used for proper final assessment of both the method and the accompanying software.

At present, the evaluation results play a secondary role, as the target is the method itself. Due to the fact that few participants responded their comments to the open questions about potential improvements, no additional information can be collected from that aspect. In addition, evaluation with larger representative samples will be replicated in the future when more people can access the method and prototype. When the method and prototype are getting more mature, the evaluation is planned to be extended, i.e. refining/enriching the current instrument questions in the questionnaire. In all, in order to gain a high confidence, a further evaluation extending to future version of prototype/tool with larger and better samples is currently under the plan.
7 Conclusions

SOA-based systems offer high degree of flexibility and interoperability. However, the securing of SOA-based applications is still a challenge. The thesis project is about a method proposed by researchers from Institute A. It focuses on the analysis of architectural level security artifacts, rules and indicators and the relation among them.

Our main contributions in the thesis project are:

1. Studying the current state of SOA security comprehensively, and introducing a security taxonomy fit for SOA context;
2. Identifying and elaborating the detailed process of a method for analyzing security in SOA, and reflecting the method based on field study;
3. Introducing a prototype/tool supporting and demonstrating the method (i.e. eliciting the requirement, providing the system design, implementing the identification process of the prototype which can also turn to prove the correctness of the method, etc);
4. Evaluating the method and prototype based on survey (i.e. questionnaire), and discussing and interpreting the evaluation results.

The detail answers to each research question are as follows:

• RQ1: What is the current state of SOA security?
For answering the RQ1, a comprehensive literature study and review was performed. As described in section 2, the security taxonomy shows the big picture of the overall aspects of security: security assets, security attributes, security threats, security solutions, and security metrics. Different threat modeling techniques to model security threats are mentioned. Two types of security solutions are presented and discussed: SOA security industry standards (i.e. WS-Security, SAML, WS-Trust, etc) and SOA security patterns (i.e message protection pattern, access control pattern, etc). Besides, the existing technologies (static analysis and dynamic detection techniques) for security analysis are explored, compared and discussed. In addition, for security inspection, static analysis techniques are most widely used in current state.

• RQ2: What are the preliminary reflection results of the method?
According to the field study of the method at institute A, the preliminary reflection results are presented in section 3, they are mainly as follows:

– Software architect, quality assurer, tester and security engineer can benefit from the method. The method can help stakeholder detect security vulnerabilities hidden in the source code, i.e. violation of secure coding rules, violation of architecture pattern, etc. So it also can be integrated into testing phase of software development life cycle.
– The method is a static analysis approach based on the architecture extracted from source code. It is a semi-automatic software inspection process focusing on security aspects. The method takes the software architecture, which is reverse-engineered from source code, and security expertise from security experts as inputs. The output is security analysis reports and views.

– One distinct component of the method is knowledge base model. Security expertise from security experts can be reused and packaged in a proper way in the model. However, more specific security context rules are needed to be defined. For example, some semantic information is missing in the rule definition. The user can not easily apply the rule in certain scenario unless he/she is experienced or a security expert.

– It is coupled with Software Architecture Visualization and Evaluation (SAVE) tool. As an internal reverse engineering tool in Institute A, SAVE has already been applied in some projects. Researchers inside institute A are familiar with it and already have some experience of how to use it. In addition, SAVE can already generate part of information needed to be stored in the system model of the method. Although there will be some integration effort, reuse SAVE is promising.

• RQ3: How to implement a prototype that supports the main processes of the method?
Section 4 answered the RQ3. The requirement of the prototype was elicited and analyzed. After that, two possible solution were proposed, and the pros and cons of each approach were discussed and compared (see table 6). Based on the current situation and constraints, the appropriate solution was chosen. Then the system design for main process of the method is provided for the chosen solution. Based on the design, the prototype was implemented and tested in an running example.

• RQ4: How to evaluate the method and prototype?
An evaluation process for the method and prototype was designed in section 5. The evaluation method is a survey (questionnaire) based on a evaluation instrument inspired by technology acceptance model. The population and sampling for the evaluation are also discussed and presented. The evaluation is properly executed and data are correctly collected.

• RQ5: What are the evaluation results of the the method and prototype?
The main evaluation results are discussed in section 6. The evaluation result shows the prototype supporting the method is a promising inspection tool for helping human to detect the software vulnerability. The possible validity threats are explicitly stated and discussed. According to the results, the further research with respect to evaluation is proposed and discussed.
8 Discussion and Future Work

8.1 Discussion

As a bottom-up approach, the method just provides an approximation of runtime behavior. One possible way to improve the precision is to integrate PMD (see section 2.2) into the method. As security checking rules are already defined by security experts in the knowledge base, it can be reused to write a PMD rule in Java. After the rule is written, it is added into a rule set factory. When violation or problem is detected by the rule, PMD will create an alarm of rule violation and generate the rule violation report in a file. Security inspector can have better understanding of the existing problem based on the detail context. As a result the precision of inspection can be improved.

From top-down perspective, the security checking rules in the security tag should strengthen the links with security requirements or security patterns. Although there are security goal categories defined in the knowledge base, their content are not as rich as security patterns. A completed solution should contain at least the following information: application context, problem description, possible solutions and practical examples or guidelines, etc. In addition, current security tags are not powerful enough to performance security inspection in a complicated scenario. Therefore, more security rules should be added and improved for the security tags.

8.2 Future Work

As the method is not mature and under development, only limited functionalities supporting the method have been implemented in the prototype. There are still improvements works for the method and prototype needed to be done in the future. A further evaluation with larger and better samples is currently under the plan given that the method will be published or more people can get access to the method. In order to evaluate the practical usage of the method, a prototype/tool supporting the method is necessary. So to make the evaluation more reliable, a prototype is currently under professional development. Then the evaluation can be extended with the mature prototype with extended/refined questions in the questionnaire to get more feedback based on practical usage. Based on the evaluation, the method can be properly improved and implemented.

In summary, the directions to the future work are as follows:

1. Adding more functionalities to the prototype/tool as new version of the method appears or the method matures;

2. Conducting an experiment as a mature evaluation when larger population can access the method, extending the evaluation with the prototype based on extended/refined instruments/questionnaire to get more reliable evaluation results;

3. Improving the method and prototype/tool properly based on the mature evaluation.
References


Appendices

A Source Code of Library Sample

The source code of the library sample can be found in the following link: http://www.mediafire.com/?jjqql5mv3mq
# B Questionnaire

## Questionnaire used

### Role:

<table>
<thead>
<tr>
<th>Ease of use</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usefulness</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Improvement potential</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Questionnaire results: Please see or download the results from the following link:
http://www.mediafire.com/?kvgznwllmvd
C Source Code of Prototype

Please download the source code from the following link: (Regarding to the password, please contact the authors.)
http://www.mediafire.com/?wynzwjtkzzl