Complexity Measures for Secure Service-Oriented Software Architectures

Yanguo (Michael) Liu, Issa Traore
Department of Electrical and Computer Engineering
University of Victoria, BC, Canada
Email: yliu@ece.uvic.ca, itraore@ece.uvic.ca

Abstract

As software attacks become widespread, the ability for a software system to resist malicious attacks has become a key concern in software quality engineering. Software attackability is a concept proposed recently in the research literature to measure the extent to which a software system or service could be the target of successful attacks. Like most external attributes, attackability is to some extent disconnected from the internal of software products. To mitigate software attackability, we need to identify and manipulate related internal software attributes. Our goal in this paper is to study software complexity as one such internal attribute. We apply the User System Interaction Effect (USIE) model, a security measurement abstraction paradigm proposed in previous research, to define and validate a sample metric for service complexity. We thereby establish the usefulness of our sample metric through empirical investigation using open source software system as target application.

1. Introduction

Software systems that run in open environments are facing more and more attacks or intrusions. This situation has brought security concerns into the software development process. Generally, software services are expected not only to satisfy functional requirements but also to be resistant to malicious attacks. Software *attackability* is defined as the likelihood that an attack on a software system will succeed [1]; in other words, it represents the ability of software services to resist malicious attacks.

The main objective of this research is to develop a quantitative framework for predicting and mitigating software attackability in the early stages of the development process at the architectural level. We explore a quantitative approach for attackability analysis because the use of software metrics as cost-effective quality predictors is widely accepted in the software community, both in academia and industry.

As argued by Whittaker [2], there are no software systems that are immune to all kinds of attacks. Some software systems may be strongly resistant to one form of software attacks, but seriously vulnerable to another kind. Whittaker believes that some inner features of software products bear directly on the ability of software products to resist against specific forms of software attacks. We need to be able to affect these internal features in order to improve software security as an external quality. Our objective in this paper is to study empirically software complexity as one such inner feature. We study software complexity in the context of service-oriented architectures. We have shown in previous work how the User System Interaction Effect (USIE) paradigm can be used as measurement abstraction to derive various software security metrics [3, 4]. In this paper, we define a sample metric for service complexity based on the USIE paradigm, and use such metric to study the empirical relationship between service complexity and attackability. In our study, we use an open source web-based software system as target application and focus on one form of security attack, namely the URL Jumping attack.

The rest of the paper is structured as follows. Section 2 gives an overview of service-oriented architecture, and briefly discusses our measurement abstraction based on the USIE paradigm. Section 3 outlines our understanding of service complexity measurement and presents sample metric using the USIE paradigm as measurement abstraction. Section 4 investigates empirically the relationship between service attackability and complexity using the sample metric. Section 5 summarizes and discusses related works. Finally, section 6 makes some concluding remarks and presents future works.

2. Software Service Model

In this section we discuss the notion of service oriented architecture, and give an overview of a related security measurement abstraction that serves as basis to derive service complexity metrics.

2.1 Service Oriented Architecture

The Service-Oriented Architecture (SOA) as introduced in [5] is an emerging architectural style for building enterprise applications. Simply put SOA is an architecture style in which functionalities of a software system are constructed and delivered as services to either end-user applications or other services.

Software services correspond to business functions derived from business processes analysis. Depending on the business process, the service may be composed from other services or consist of a collection of software components that work collectively to achieve corresponding business function. Software components are finer-grained than services, and also map into business entities and corresponding behaviors or business rules in contrast with services, which map into business functions. To deliver the business function it represents, a service may manage operations across a set of business entities. As such it creates and manages its own set of components.

From an abstract perspective, a service can be defined as a set of alternative configurations, where each configuration consists of a collection of components connected in a specific way, each providing specific services referred to as the *supporting services* for the corresponding configuration [6].

In this context, a software system S based on SOA can be defined (abstractly) as a set of service configurations with a partition $\overline{S}$ on S representing the set of services delivered by the system. Each partition consists of a collection of configurations, and maps into a software service. Each configuration in its turn maps into a set of supporting services. The partitioning in configurations defines a hierarchical structure between services. In this hierarchy, terminal services, which are isolated from one another, are considered atomic services. An atomic service has only one configuration and has no supporting services.

2.2 Security Measurement Abstraction

The UML [7] has become a key notation used to capture SOA abstractions [8, 9]. However, security analyses of UML models involve some challenges. Security related events could not be described satisfactorily using the basic or regular semantics of UML diagrams. Moreover, the modeling features provided by the UML interaction diagrams make it rather difficult to conduct directly security analysis.

In previous work, we have introduced the USIE paradigm, which addresses these deficiencies by isolating and capturing security intents and events [3, 4]. The collected information is used for automated security analysis of software services designs. More specifically, a USIE model captures in one hand the security dependencies underlying the hierarchical structure of software services, and on the other hand the traces of the communications in user-system interactions as defined by UML interaction diagrams. Such information can be collected automatically from UML models by defining or reusing adequate security profiles. The collected information can also be used to compute and analyze adequate security measures with the purpose of improving architecture design.

A USIE model consists of a graphical structure $U = (N, E, <, L)$, where N represents a set of nodes, E represents a set of directed edges, $<$ is a partial order relation over E, and L represents the set of labels expressing various relationships between USIE entities.

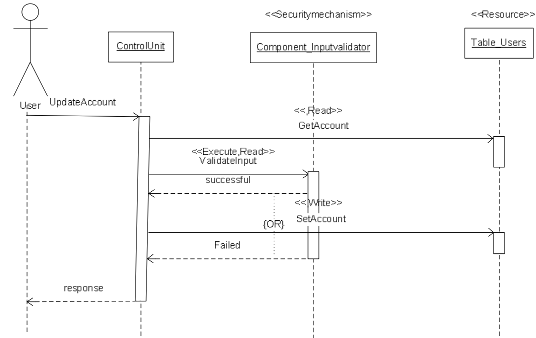

Figure 1. The Service Specification

USIE modeling involves three different types of graphs which cover the different levels of the hierarchical structure of service oriented architecture. These include the *USIE composite service graph*, the *USIE configuration graph*, and the *USIE atomic service graph*. A USIE composite service graph is a tree that captures the service hierarchy underlying a particular composite service by showing the relationships between the different configurations and corresponding supporting services. A USIE configuration graph describes for a given service configuration the dependencies between corresponding supporting services. A USIE atomic service graph is a tree that captures the user-system interactions underlying an atomic service.

For instance, to illustrate USIE model construction for an atomic service, we consider as an example an atomic service named *UpdateAccount* that is used to update customer account information in an online web store. The user-system interactions related to the

UpdateAccount service are illustrated in Figure 1 through a UML sequence diagram.

As shown in Figure 1, the "UpdateAccount" request from a user is sent directly to the "ControlUnit" component, which in its turn interacts internally with the "Component_Inputvalidator" and the "Table_Users" components. The "Table_Users" component will either be updated or remain unchanged based on the validation results returned by the "Component_Inputvalidator" component. In the sequence diagram of Figure 1, we highlight the security semantics of the underlying components and operations using appropriate stereotypes. These model elements are mapped into corresponding USIE modeling entities. For instance, the component named *Component_InputValidator* is stereotyped as *SecurityMechanism* component, while the *Table_Users* component is considered to be a *Resource* component. Accordingly, the incoming request to the *Component_InputValidator* and the *Table_Users* are also stereotyped based on their USIE effects. Briefly, a software component is considered to be a USIE *SecurityMechanism* component if it provides or implements some form of protection during a service execution. A component is considered to be a USIE *Resource* component if it represents a form of system resources needing some protection. The USIE effects stereotypes are defined in accordance to the collection of rights associated with the execution of operations on target components.

Figure 2 depicts the USIE model derived from the sequence diagram of Figure 1. The solid circle represents a USIE atomic service node that is named after the corresponding sequence diagram. The "RS" circles represent USIE resource components and the "SM" circles represent USIE security mechanism components. The USIE operations are represented by directed edges with operational rights specified. The diamond nodes carry labels expressing the execution order of outgoing USIE operations. For instance, the "Sequential" label in the diamond indicates that the outgoing edges on the left will be invoked before the outgoing edges on the right. The "Exclusive" label in the diamond indicates that only one of the outgoing edges will be taken during a service execution.

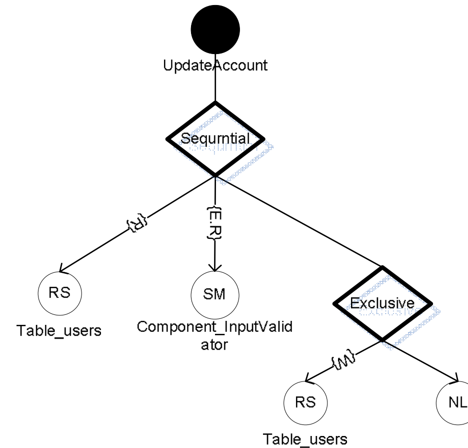

Figure 2. USIE Model derived for "UpdateAccount" Interaction Diagram

3. Software Service Complexity

In this section, we discuss notions of service complexity and the relevance of such notions to software security. We also present a sample service complexity metrics using the measurement abstraction introduced in the previous section.

3.1 Notions

In the last several years, a rich body of research has been produced on software complexity [10, 21, 22]. At the same time, with the growing interest in software security research there is a consensus that complexity has a negative impact on software security. To our knowledge, despite such consensus little effort has been made toward investigating empirically the link between complexity and security in software systems. Existing empirical works on software complexity have targeted for the most part traditional qualities such as correctness, reliability and maintainability, not security.

The link between complexity and security is a well-accepted fact in system security engineering. Such acceptance stems from popular security design principles outlined by Schroeder and Saltzer [23]. In particular two of these design principles, namely the principle of "psychological acceptability" and the principle of "economy of mechanisms", directly relate to the issue of complexity. The principle of "psychological acceptability" states that the introduction of a security mechanism should not make the system more complex than it is without it. The principle of "economy of mechanisms" recommends that system security mechanisms should be kept as simple as possible.

Simplicity is essential in secure systems engineering for obvious reasons: complex mechanisms are difficult to build, maintain, and use, and thereby tend to increase security risks.

Melton et al. stress the distinction between psychological complexity and structural complexity [10]. Psychological complexity is based both on system characteristics and human factors, while structural complexity refers to the complexity arising from the software system irrespective of any underlying cognitive considerations. Although, both forms of complexity affect software security, we limit the scope of this paper only to structural complexity.

3.2 Measures

Using USIE abstraction, a software system can be represented as a collection of services structured hierarchically where the top service is a composite service representing the application itself. Furthermore, each of the services involved in the hierarchy can be described using a corresponding USIE graph that can be used to derive various metrics. In this case, the measurement targets are the software service entities. Particularly, since we are interested, in this paper, in studying service complexity, we define a sample service complexity metric as follows:

Measure of Service Complexity: Given a composite service *cs* and its USIE model $U_{cs} = < N_{cs}, E_{cs}, <_{cs}, L_{cs} >$, the *Average Service Depth* (*ASD*) for U_{cs} is defined as

$$ASD(U_{cs}) = \frac{\sum_{n_i \in N_{cs}} NumOfServiceDependency(n_i) \times IsAtomicService(n_i)}{\sum_{n_i \in N_{cs}} IsAtomicService(n_i)}$$

Where:

$$IsAtomicService(n_i) = \begin{cases} 1, & if\ n_i\ is\ an\ atomic\ service\ node. \\ 0, & otherwise. \end{cases}$$

$NumOfServiceDependency(n_i)$ = *The number of service nodes* depending *directly or* indirectly *on* n_i.

The USIE model of a composite service explores the inner dependency relationships between atomic services involved. Intuitively, the more inner dependency relationships exist within a composite service, the more structurally complex the composite service ought to be. The ASD metric actually computes the average number of dependency relationships per atomic service node; therefore, high ASD values indicate high degree of dependencies in the service. In other words, ASD metric can (intuitively) be used as a complexity metric for composite services.

4. Empirical Study

We illustrate, in this section, the empirical investigation underlying this work by describing the main steps involved and by presenting and discussing the obtained results.

4.1 Guiding Principles

Many researchers have realized that formal experiments and surveys are extremely difficult to conduct in the area of software engineering [11]. The diversity and rapid changes of software development techniques make it very difficult to obtain enough resources for formal experiments. As a result, there has been an increasing interest in the methodology of case studies in the area of software engineering over the last several years. Case studies, as discussed in [12], can be used as evaluation methods allowing new software engineering techniques or methodologies to be assessed in a limited context. Case studies are beneficial for empirical studies in software engineering because they allow conducting the evaluation in a realistic and cost-effective way. In this paper, we study the empirical relations between software attackability and complexity as an internal security-related attribute using the methodology of case study.

Due to the diversity and complexity of software attacks, it is neither realistic nor practical to interpret software attackability uniformly. In our work, we explain attackability with respect to specific software attacks, for example, attackability with respect to password-cracking, attackability with respect to race condition, or attackability with respect to denial of service, etc. For the purpose of this paper, we use in our case study a single form of software attack to investigate the empirical relationship between attackability and service complexity. The approach could be adapted to study other forms of attacks and internal security-related attributes.

Our case study is based on an online Flower Shop system and focus on the URL jumping attack. Based on our intuitive understanding of how this attack operates, we make the assumption that with respect to URL Jumping attack, attackability increases as service complexity increases. This represents the hypothesis in this study.

4.2 URL Jumping Attack

4.2.1 Attack Pattern. A common web-based application relies on a specific flow of functionality that developers expect or assume will faithfully be

followed by users. For instance, in a typical e-commerce application, after placing an order a new customer is expected to register (by creating a secure account) before initiating the payment process. For a returning customer, the application typically will expect that after placing an order, the customer would login before paying. In either case there is a well-defined sequence of tasks that the developer would expect to be enforced. For instance, if a returning customer can jump directly from the Order page to the Payment page, without going through the Login page, this means that the system has no way to keep track of its customers. This is not very effective either from business perspective or from security standpoint. Unfortunately the inherently stateless nature of the HTTP protocol makes such scenario possible. It is possible for a user to jump to any pages just by entering the URL in a Web browser.

The purpose of the URL Jumping attack is to exploit such deficiency in poorly developed Web application, by identifying in a web application a set of tasks that should be sequenced, and attempt to skip certain (required) steps by browsing over them [2].

In principle, the service execution sequence should be validated in the program logics by software developers. Unfortunately, some developers, due to careless implementation or misunderstanding of complex services dependencies, may omit or miss validations of the preconditions associated with key services executions. As a result, malicious users may exploit such opportunity by breaking out of page sequences.

4.2.2 Attackability Measure. We adopt an *effort-reward* approach to assess the external attackability of software services in operational environments [14]. Specifically, we define specific measurements for attack effort and reward with respect to specific software attack. The relative attackability among different software services (or different software applications) facing the same type of attack in identical operational environments can be captured by comparing the attack efforts required under the same reward, or by observing the attack rewards involved under the same effort. Generally, we compute the relative attackability by the ratio $\frac{Attack\,Reward}{AttackEffort}$.

Specifically for URL jumping attack, we quantify attack effort and reward of the URL jumping attack by defining the following metrics:

Measure of URL Jumping Attack Effort:

$$AttackEffort_{URL-Jumping}(service_i) = \textit{The number of URLs exploited related to the service}_i.$$

Measure of URL Jumping Attack Reward:

$$Attack\,Reward_{URL-Jumping}(service_i) = \begin{cases} 1, & \textit{if a URL Jumping vulnerability is found in serviec}_i. \\ 0, & \textit{otherwise.} \end{cases}$$

Specifically, we define the attack effort for URL Jumping attack on a given composite service as the total number of the URLs explored in finding a URL jumping vulnerability. And we define the attack reward of URL jumping attack as a binary value which will be set to 1 if there is URL Jumping vulnerability in the given composite service and 0 otherwise.

Since there is no universal interpretation for the notions of attack effort and reward for URL Jumping attack pattern, we define in this work such measurements based on our own understanding. We believe that our measurements capture the URL Jumping attackability to some extent, but we don't claim that these measures are definitive. Defining universal attackability measurements is not in the scope of this work.

4.3 Target Application: The Online Flower Shop System

The Flower Shop (FS) system is a web-based open source software application proposed in [2], which allows customers to purchase flowers online. The FS application is based on three-layer architecture: user, services, and data. Customers visit online services using a browser.

The individual services implemented using PHP are deployed on a web server; confidential information is stored in a backend database server. The FS database consists of 8 tables used to organize information such as user accounts, flower supplies, online sessions, and so on. 38 PHP pages that can be remotely accessed by both online customers and system administrators under proper privileges implement the various FS services. Figure 3 depicts the service hierarchy of the FS application. In this figure, dual circles represent composite services and single circles represent atomic services; arrows denote the support relationships between services. An arrow pointing from service X to service Y indicates that Y is a supporting service for one of the configurations of X. For instance, the root service represented by the online store is supported by two services: customer service, and administrator service.

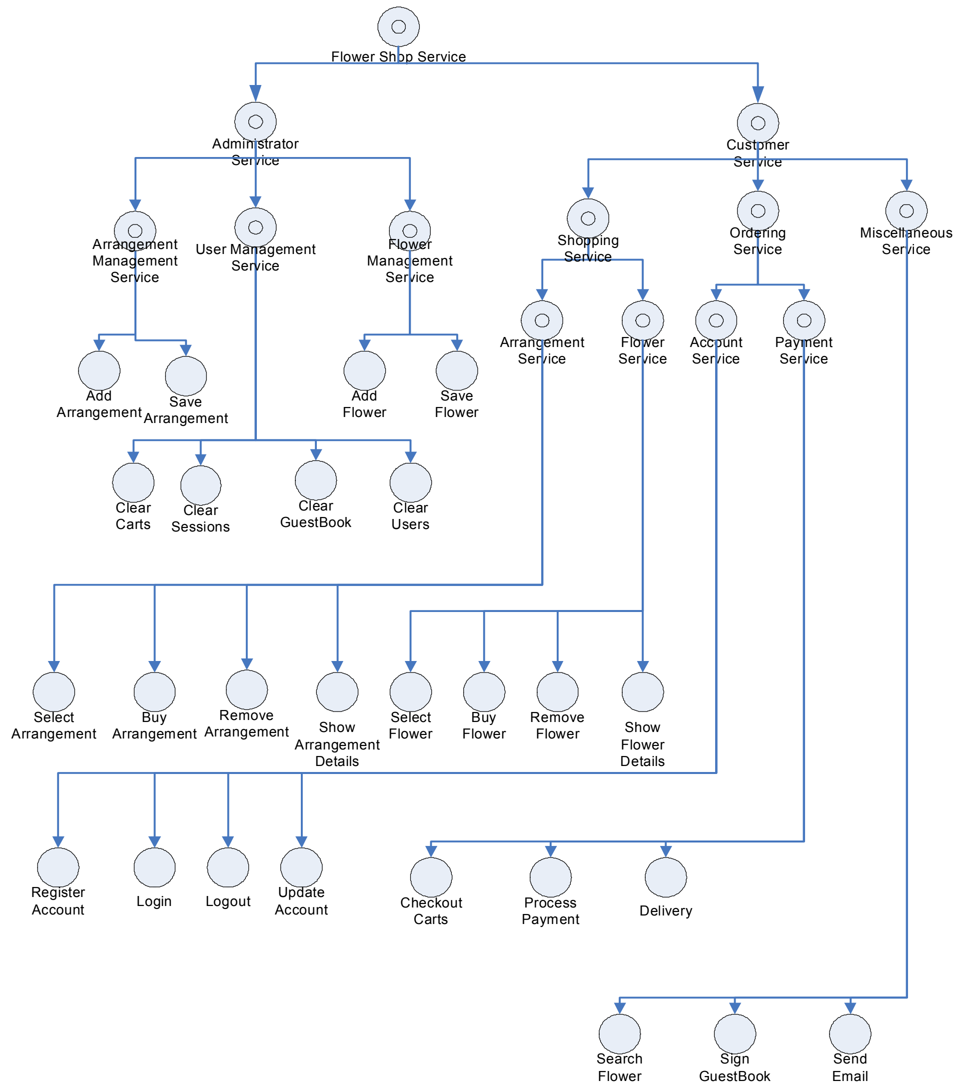

Figure 3. Service Hierarchy for FS Application

4.4 Study Environment

During the study, the main components of our target application (i.e., The Flower Shop system) were deployed on two different server machines: one running a DeveloperSide.Net web server version 1.16 [13], and the other deploying a database server MySQL 5.1 serving as central data storage.Attacks were conducted remotely from a different machine.

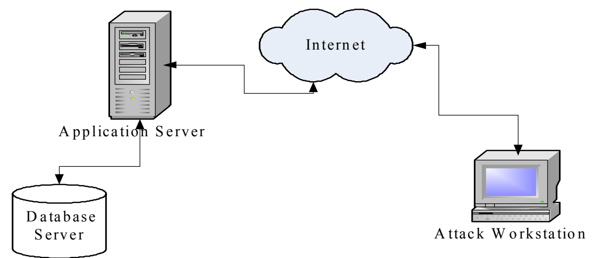

Figure 4. Experimental environment

Figure 4 illustrates the general architecture of the distributed system. The characteristics of the server machines involved are as follow:

- Application server computer: Intel ® Pentium 4 Mobile CPU, 2.0 GHz, 522MB RAM, Win 2000.
- Database Server Computer: Intel ® Pentium 3 CPU, 1.0 GHz, 224MB RAM, Windows XP.

4.5 Measurements

In our study, we selected *Administrator Service*, *Customer Shopping Servic*e and *Customer Ordering Service* from the flower shop application as the target services to validate our hypothesis empirically.

4.5.1 Service Complexity. Figures 5-7 show the USIE composite service graphs constructed for the three (composite) services targeted in the study. In these models, solid circles denote the atomic services and service dependencies are represented by directed edges.

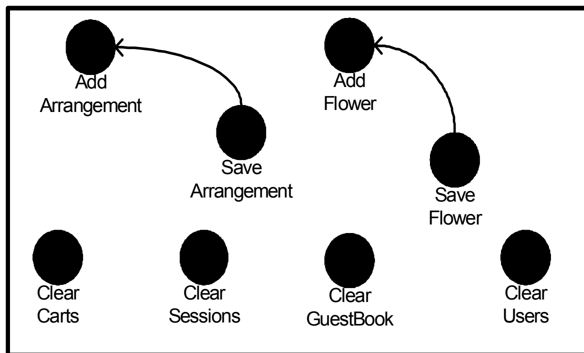

Figure 5. The USIE model of Administrator Service

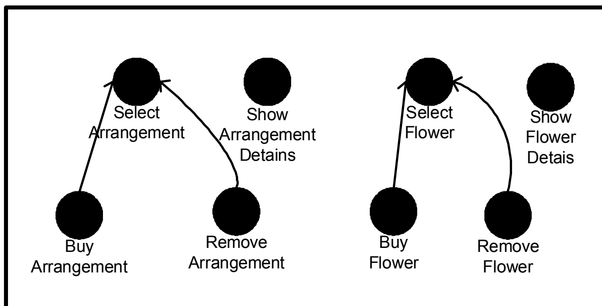

Figure 6. The USIE model of Customer Shopping Service

In addition, diamond nodes at the source of the edges specify the logical relationships between multiple service dependencies. For instance, in Figure 7, the "AND" diamond node specifies that the "Process Payment" service will be executed only if both the "Login" service and the "Checkout Carts" service have been executed.

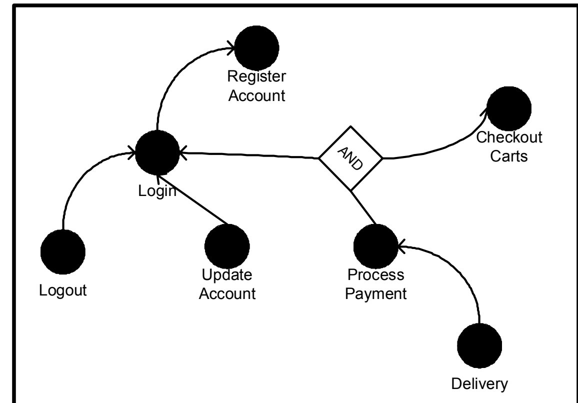

Figure 7. The USIE model of Customer Order Service

Based on Figures 5-7, we derive the values of the ASD metrics for each of the targeted services, as illustrated in Table 1.

Table 1. Metric Values of Service Complexity

No.	$Service_i$	$ASD(Service_i)$
1	Administrator Service	0.25
2	Customer Shopping Service	0.5
3	Customer Order Service	1.71

4.5.2 Attackability. Table 2 presents the results of the URL Jumping attacks over 20 independent runs. Table 3, which is derived from Table 2, presents the relative URL Jumping attackability for the three target services in each of the runs. In each run, we randomly select atomic services from the target composite services and perform URL jumping attacks on them.

Table 2. URL Jumping: *AttackReward* = 1

Experiment No.	*AttackEffort*		
	Administrator Service	Shopping Service	Order Service
1	∞	4	5
2	∞	6	4
3	∞	1	3
4	∞	7	2
5	∞	4	3
6	∞	5	4

7	∞	6	5
8	∞	4	5
9	∞	6	1
10	∞	4	3
11	∞	5	4
12	∞	4	4
13	∞	4	2
14	∞	4	6
15	∞	3	3
16	∞	7	3
17	∞	5	4
18	∞	4	5
19	∞	5	2
20	∞	7	3

Table 3. Relative URL Jumping Attackability

Experiment No.	Administrator Service	Shopping Service	Order Service
1	0	0.25	0.2
2	0	0.167	0.25
3	0	1	0.333
4	0	0.143	0.5
5	0	0.25	0.333
6	0	0.2	0.25
7	0	0.167	0.2
8	0	0.25	0.2
9	0	0.167	1
10	0	0.25	0.333
11	0	0.2	0.25
12	0	0.25	0.25
13	0	0.25	0.5
14	0	0.25	0.167
15	0	0.333	0.333
16	0	0.143	0.333
17	0	0.2	0.25
18	0	0.25	0.2
19	0	0.2	0.5
20	0	0.143	0.333

We decided, for the URL Jumping attack, to compare service relative attackability under the same attack reward that is *AttackReward = 1*. Therefore, the attacks on the composite services will stop once we identify URL Jumping vulnerability for the corresponding composite services.

Accordingly, the relative attackability between these services can be compared based on their attack efforts. If no URL Jumping vulnerability is found for a target service when all of its atomic services have been targeted, we consider the corresponding attack effort to be infinite (i.e., ∞).

4.6 Analysis of the Results

Table 4 presents the correlation coefficients between service ASD measures (presented in Table 1) and the service relative attackability measures (presented in Table 3).

Table 4. Correlation Coefficients for ASD Metric and Relative URL Jumping Attackability

Experiment No.	**Correlation Coefficients**
1	0.474372
2	0.850200
3	-0.029715
4	0.992740
5	0.799306
6	0.767224
7	0.743894
8	0.474372
9	0.999991
10	0.799306
11	0.767224
12	0.632190
13	0.934899
14	0.345139
15	0.632190
16	0.960636
17	0.767224
18	0.474372
19	0.969439
20	0.960636

Figure 8 presents the analysis results of the correlation coefficients based on data collected in Table 4. According to Figure 8, the mean of the correlation coefficients would be a good estimator for the relationship between ASD metric and URL Jumping attackability. The estimate is around 0.72, which indicates a strong positive correlation between ASD metric and URL Jumping attackability. In other words, as the value of service ASD metric increases, then the value of the service URL Jumping attackability will increase as well. The median of the correlation coefficients could also be a good measurement. The estimated value is around 0.77, which signals an even stronger correlation between the ASD metric and URL Jumping attackability. The histogram (the graph on the left side of Figure 8) shows the relative frequencies of all the correlation coefficients from our sample. Clearly, we observe that most of the correlation coefficients fall in the range 60-99%.

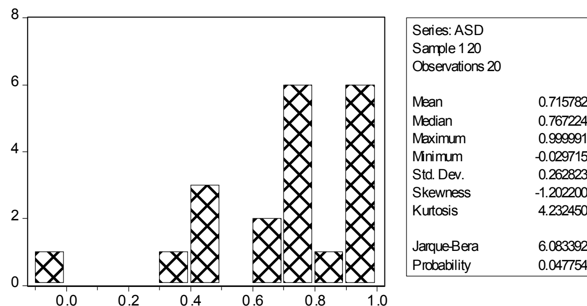

Figure 8. Analysis Results of Correlation Coefficients

In conclusion, the URL Jumping attackability of a software service and the service's ASD metric value tend strongly to increase in the same directions. As the service ASD metric increases, the service URL Jumping attackability increases. Therefore, our hypothesis is supported by the experimental results.

5. Related Work

A wide range of work has been undertaken in security risk analysis of computer systems, which includes the areas of adversary modeling, attack specification, vulnerability analysis, security-related taxonomies and databases, etc.

In [1], Howard et al. proposed to use attack surface to determine whether one version of software application has less attackability than another. They define attack surface in terms of system actions that are externally visible to system's users and the resources accessed or modified by each action. Their methodology is based on the intuitive understanding that more explored attack surfaces lead to higher likelihood of being successfully attacked. However, since defining the class of attack surfaces is application specific, their methodology requires human expertise to identify the class of attack surfaces for specific software systems.

Bug counts are commonly used as a measure of software security [15-18]. Software bugs are collected from either inspections or testing reports. However, it is difficult to determine in advance the seriousness of a defect, and in practice, a very small portion of defects in a system might cause almost all the observed security breaches.

Voas et al. [19] proposed a quantitative approach to assess relative security among different versions of the same software system. Their approach, which is named Adaptive Vulnerability Analysis (AVA), exercises software source codes by simulating incoming attacks. A metric is computed by determining whether the simulated attacks undermine the security of the system as defined by the user according to specific application program. AVA applies fault-injection techniques to the source codes of software applications, which makes AVA only applicable in the late stage of software development. In contrast, our framework targets primarily the software design phase.

Ortalo et al. proposed a theoretical model for attackability measurement and associated tools [20]. The theoretical model is based on a description of attack scenarios using the so-called privilege graph. A measurement of the difficulty for attackers to compromise the system is generated from the privilege graph using Markov chains. Global metrics are computed by combining weights assigned by security officers to the arcs of the graph. Their methodology focuses primarily on improving system and network security, and does not target individual software applications. Moreover, like many other methodologies, their approach suffers from the fact that they rely to a large extent on subjective primitives.

6. Conclusion

We have studied, in this paper, through a case study the empirical relationship between software complexity and attackability, confirming to some extent the widely held belief that complexity has a negative impact on security. We recognize that it is not sufficient to infer a general correlation between the measured structural complexity and the likelihood of successful attack based on only one case study. Although many empirical studies would be required to draw a general conclusion, the current study is a good step in this direction.

In future work, we plan to investigate using the methodology of case study the empirical relationship between attackability and many other software security related attributes such as sharing or coupling, excess privilege, and internal security mechanism strength. This will allow establishing better understanding of the underpinnings and intricacies of secure software and thereby leading to better products.

References

[1] Howard M., Pincus J., Wing J. M, "Measuring Relative Attack Surfaces", *Proceeding of Workshop on Advanced Developments in Software and System Security*, 2003.

[2] M. Andrews, James A. Whittaker, "How to Break Web Software", *Addison-Wesley*, 2005.

[3] M. Y. Liu, Issa Traore, “Empirical Relationships between attackability and coupling: case study for DOS”, *ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, Ottawa, Canada, June 10, 2006

[4] M. Y. Liu, Issa Traore, “Measurement Framework for Software Privilege Protection based on User Interaction Analysis”, *11th International Software Metrics Symposium*, Sponsored by the IEEE Computer Society, 19-22 September 2005, Como, Italy.

[5] Endrei M., et al., “Patterns: Service-oriented Architecture and Web Services”, *Redbook*, SG24-6303-00, April 2004.

[6] J. K. Millen, “Survivability Measure”, *Research Report*, *Computer Science Laboratory (CSL)*, SRI International, Menlo Park, CA, USA.

[7] G. Booch, J. Rumbaugh, I. Jacobson, “The Unified Modeling Language Reference Manual”, *Rational Software Corporation*, 1999, Addison Wesley Longman, Inc.

[8] O. Zimmermann, P. Krogdahl, C. Gee, “Elements of Service-Oriented Analysis and Design”, *IBM developer Works*, June 2nd 2004.

[9] A. Arsanjani, “Service Oriented Modeling and Architecture”, *IBM developer Works*, Nov 9th 2004.

[10] A. C. Melton, D. A. Gustafson, J. M. Bieman, and A. L. Baker, "A mathematical perspective for software measures research," *Software Eng. J.*, vol. 5, no. 5, pp. 246-254, Sept. 1990.

[11] B. A. Kitchenham, “Evaluating Software Engineering Methods and Tools”, *Software Engineering Notes*, Vol. 21 N o. 2, 1996.

[12] B. A. Kitchenham, “Evaluating Software Engineering Methods and Tools”, *Software Engineering Notes*, Vol. 23 No. 1, January 1998.

[13] http://www.devside.net/

[14] S. Brocklehurst, B. Littlewood, T. Olovsson, and E. Johsson, “On Measurement of Operational Security”, *Proceedings of the 9th Annual Conference on Computer Assurance*,1994.

[15] A. Chou, J. Yang, B. Chelf, S. Hallen, D. Engler, “An empirical study of operating systems errors”, *ACM Symposium on Operating Systems Principles*, Oct. 2001, pp. 73–88.

[16] J. Gray, “A census of tandem system availability between 1985 and 1990”, *IEEE Transactions on Software Engineering*, vol. 39, no. 4, Oct. 1990.

[17] I. Lee, R. Iyer, “Faults, Symptoms, And Software Fault Tolerance”. *In The Tandem GUARDIAN Operating System, Proceedings of the International Symposium on Fault-Tolerant Computing*, 1993.

[18] M. Sullivan, R. Chillarge, “Software Defects And Their Impact On System 118 Availability”, *Proceedings of the International Symposium on Fault-Tolerant Computing*, June 1991.

[19] J. Voas, A. Ghosh, G. McGraw, F. Charron, K. Miller, “Defining an Adaptive Software Security Metric from a Dynamic Software Failure Tolerance Measure”, *Proceedings of the 11th Annual Conference on Computer Assurance*, 1996.

[20] R. Ortalo, Y. Deswarte, M. Kaaniche, “Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security”, *IEEE Transactions on Software Engineering 25*,5 (1999) p.633-650.

[21] L. C. Briand, S. Morasca, V. R. Basili, “Property-Based Software Engineering Measurement”, *IEEE TSE*, Vol. 22, No. 1, Jan.1996.

[22] E. J. Weyuker, “Evaluating Software Complexity Measures”, *IEEE TSE*, vol. 14, no. 9, pp. 1357-1365, Sept. 1988.

[23] J. Saltzer, M. Schroeder, “The Protection of Information in Computer Systems”, *Proc. of the IEEE* 63 (9), pp. 1278-1308, Sep. 1975.