

On Estimating the Security Risks of Composite Software Services

Jian Yin, Chunqiang Tang, Xiaolan Zhang, and Michael McIntosh

IBM T.J. Watson Research Center

Abstract. With the rapid adoption of the Service Oriented Architecture (SOA), sophisticated software systems are increasingly built by composing coarse-grained service components offered by different organizations through standard web service interfaces. The ability to quantify end-to-end security risks of composite software services is extremely valuable to businesses that increasingly rely on web applications to interact with their customers and partners. In this position paper, we propose a framework that predicts the probability of end-to-end security breaches of a software service by using a combination of three models: (1) a software security model that describes the probability distribution of security bugs in individual components, (2) a service composition model that describes the interactions of components and the contribution of security bugs in individual components to the overall security of the service, and (3) a hacking exposure model that estimates hackers' knowledge of individual components and hence the probability that a security hole, if exists, may be exploited.

1 Introduction

The service sector has undergone a rapid expansion in the past several decades. In the United States, services account for approximately three quarters of GDP and eight out of ten jobs [16]. This trend has driven the Information Technology (IT) industry to shift its focus from the sales of computer hardware and software toward providing value-added IT services. Another trend in the industry is that many organizations increasingly rely on web applications to deliver critical services to their customers and partners. With the rapid adoption of the concepts of *Software as a Service* (SaaS) [13] and *Service Oriented Architecture* (SOA) [15], sophisticated software systems are increasingly built by composing [9] coarse-grained service components offered by different organizations through standard web service interfaces. In the past, the optimization process of software composition [9] was mainly driven by the cost of the composite solution and the quality of service provided by the composite solution. As the complexity and hence the number of software components in a service increase, the security risk can also increase rapidly if we do not pay enough attention to security in service architecture (see Figure 1). In this position paper, we focus on security issues in software service composition.

Software often has security bugs. In practice it is rather difficult to discover and eliminate all the security holes. When a piece of software is used as a component in a software service, the security holes can be exploited by hackers to compromise data

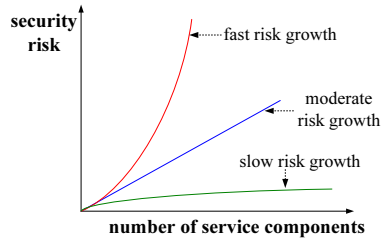


Fig. 1. Risk growth as a function of the number of service components.

confidentiality, integrity, or availability of the service. Compromising data confidentiality and integrity can result in legal liability. Compromising availability can hurt the reputation and thus the future revenue of an online service. Therefore, security breaches can be a significant risk factor to any business that operates a software system. The ability to quantify the likelihood of security breaches can be extremely useful in IT planning as well as in evaluating and choosing software components and service architecture for a software service.

Although there have been many previous attempts in predicting software quality[4, 7, 8, 10, 17], most of these studies have focused on software bugs in general. In this paper, we focus on security bugs exclusively. Moreover, we are more interested in *end-to-end* security risks of a software service comprising of multiple software components, rather than the absolute number of security bugs in individual components. On one hand, a bug in one component of the service may or may not increase the security risk of the service as a whole. For instance, the security vulnerabilities of an ftp server installed behind a firewall may not manifest themselves as the ftp server port is not exposed directly to the external world. Additionally, the familiarity of hackers on a given piece of software code also affects the likelihood that the security vulnerabilities of the code are exploited. In the proposed work, we plan to roughly quantify the risk growth rate of several typical composite services.

In this paper, we propose to predict the end-to-end security risk of composite software services using a combination of three models: a software security model, a service composition model, and a hacking exposure model. The software security model cap-

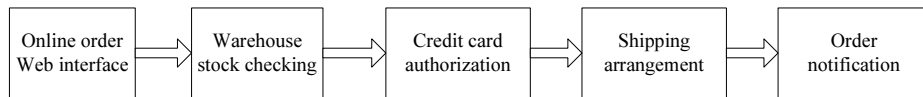


Fig. 2. Workflow of a composite software service. This example shows typical service components used in an online order process. The different components can be offered by different organizations from geographically distributed locations. For example, the “online order web interface” can be a three-tier J2EE application hosted at the online store, providing users with access to a product catalog stored in a local database. The “warehouse stock checking” component may be directly offered by a manufacturing company to allow online stores to check their product availability in realtime. “Credit card authorization” can be a service component remotely offered by a finance institution.

tures the distribution and lifetime of security bugs in each software component. The service composition model describes how the software components are put together to implement a service and the impacts of security bugs in one component on other components. The hacker exposure model is an abstract representation of the familiarity of the hacker exposure to a given piece of software and the likelihood that a security bug can be exploited given that this bug exists in the software.

2 Our Approach

In this section, we discuss our approach on predicting service security risks. A security breach prediction system uses its underlying model to assess the risk levels of a given service.

The complexity of the prediction model can vary significantly. The simplest model can treat the whole service as a black box and standard statistical regression methods are used to predict probability of future security breaches with historical data. The advantage of this approach is that it is simple to model and it places little burden on the user of the system. However, our experience indicates that this simplistic approach does not work well. The reason is that much of information that can help make better predictions is not leveraged. Such information can include the quality of individual software components, the impact of security vulnerability of each software component to the end-to-end service security, and how attractive that a particular type of service is to hackers.

On the other hand, a complex model seeks to incorporate as many relevant information as possible. There are two concerns for complex models. First, gathering relevant data can be costly and in some case requires human intervention. Second, processing overhead can grow large as it is expensive to process multi-dimensional data.

We aim to build a model that strikes the right balance between precision and performance. We incorporate as many relevant parameters as possible, yet we allow our system to be flexible in the sense that not all parameters need to be instantiated. Our system should still be able to make a prediction in cases where some parameters are not supplied, at the cost of prediction precision. We also seek to minimize human interactions by automating the process of extracting relevant information. For instance, we plan to use data mining techniques to automatically extract code quality information from a code repository.

To model security risks, one first needs to model security breaches. Instead of treating all security breaches uniformly, we divide security breaches into three categories: data confidentiality, integrity, and availability. Different security bugs can cause different security breaches depending on the severity of the bugs. For example, a denial-of-service vulnerability can cause availability problem but may not cause data confidentiality breaches. We also plan to investigate whether it is advantageous to further differentiate the security breaches in each category.

Our prediction system consists of three models: software security model, service composition model, and hacking exposure model. Next we describe each model in more detail.

2.1 Software Security Model

The goal of our software security model is to predict the number of security bugs in individual components of software. In our model, we leverage both the historical data of discovered security bugs and other relevant characteristics of the software for prediction. A security bug can be found by testers or hackers. We model software testing and hacking as sampling process and the intensity of testing and security exposure as parameters. We do not assume that the bugs are uniformly distributed. Instead, we assume that code that share similar characteristics have similar bug distributions. These characteristics capture the intrinsic complexity of the code and the degree to which good software practices are followed. Example characteristics include procedure size, number of parameters, number of local and global variables, and density of the call graph. The life cycle of a piece of code and its modification frequency and scope are other possible attributes. In many cases, the code developed by the same person or organization has similar bug distributions. Such information can be available in the comments of the source code and thus can be extracted automatically. Moreover, we plan to mine code repositories to discover other code characteristics that can potentially influence security bug distributions.

2.2 Service Composition Model

Our service composition model captures the interactions between components of the service and how the security bugs of individual software component affect the end-to-end security of the whole service. This model can be captured by a dependency graph that describes how bugs in each component affect the data of the service in terms of confidentiality, integrity and availability. Initially, this model can be constructed manually. Eventually, we want to build this model from configuration data automatically. Another fact affecting end-to-end security is the security IQ of the system administration staff. The historical security breach data of the organizations can be analyzed with similar software systems to capture this factor.

2.3 Hacking Exposure Model

The hacker community is another factor that determines the probability that a software service can be compromised. The presence of a security bug does not necessarily leads to security risks as long as the hacker community is not familiar with the piece of software to discover the bug. Our argument is similar to security through obscurity. There is still a debate on whether security can be achieved through obscurity. We hope that we can gather some real-world data to validate our hypothesis. Our model of the hacker community is used to predict the probability that security holes would be exploited given the number of exploitable security holes predicted by our software security model and service composition model. We model the knowledge of hacker community for a piece of software and its effects on the security risk. One challenge is that for some software, we may not have sufficient data to analyze. To address this issue we plan to explore similarity between pieces of software: the language that the software is implemented with, the overall architecture of this software, and the lower layer

libraries and system calls that a piece of software uses. Thus, if the hacker community is familiar with a piece of software A , we should assume that the hacker community is also somewhat familiar with other pieces of software that shares some common traits with A . The familiarity of hacker community to a type of software can be estimated by the number of hacking incidences for that software.

Service types can also influence the possibility that a service be hacked. For example, high profile commercial sites normally attract more hackers than some less well-known web sites. We also model how attractive a certain type of service is to the hackers.

3 Related work

There have been some attempts [4, 7, 8, 10, 17] to model and predict software quality. These approaches range from modeling the effect of software engineering factors that are known to affect software quality to using a black box with sophisticated analytic models. Our work differs from these previous studies in that we focus on security bugs and specialize on building models to predict security breaches.

There are some studies in measuring a particular security property of a software system. For example, Chow et. al. [3] propose to emulate the system to estimate the lifetime and thus exposure possibility of the sensitive data in a system.

There is also a large body of literature [2, 5, 11, 6, 1, 18, 12, 19] in building tools to detect software bugs. Some of them check the code dynamically during execution. Others statically analyze code with program analysis, model checking, information flow analysis, and mining code for certain pattern such as copy-n-paste. Unlike these previous studies, not only are we interested in identifying a piece of code with high possibility of containing security bugs, we are also interested in identifying a piece of code with low probability of containing security bugs as our task is to estimate the security bug distribution for the entire service. The study of software security risks is also related to the study of software reliability [14]. However, the characteristics and effect of security bugs are different from those of reliability bugs and security risks of a service are also influenced by how likely and difficult hackers will attack such a service.

4 Conclusion

In this paper, we identify the problem of modeling and predicting the probability of security breaches of composite software services. We propose a general framework that combines a software security model, a service composition model, and a hacking exposure model for predicting service security breach probability. We are in the process of evaluating our framework for real-life composite services.

5 Acknowledgement

The authors appreciate some helpful discussion with Isabelle Rouvroullou, Arun Iyengar, and Mudhakar Srivatsa. We also like to acknowledge the suggestions of the anonymous reviewers for helping improve the quality of this paper.

References

1. Ken Ashcraft and Dawson R. Engler. Using programmer-written compiler extensions to catch security holes. In *IEEE Symposium on Security and Privacy*, pages 143–159, 2002.
2. Jong-Deok Choi, Keunwoo Lee, Alexey Loginov, Robert O’Callahan, Vivek Sarkar, and Manu Sridharan. Efficient and precise datarace detection for multithreaded object-oriented programs. In *PLDI ’02: Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation*, pages 258–269, New York, NY, USA, 2002. ACM Press.
3. Jim Chow, Ben Pfaff, Tal Garfinkel, Kevin Christopher, and Mendel Rosenblum. Understanding data lifetime via whole system simulation. In *Proc. of the 13th Usenix Security Symposium*. Usenix, August 9–13 2004.
4. Mauricio Amaral de Almeida and Stan Matwin. Machine learning method for software quality model building. In *International Symposium on Methodologies for Intelligent Systems*, pages 565–573, 1999.
5. Dawson Engler and Ken Ashcraft. Racerx: effective, static detection of race conditions and deadlocks. In *SOSP ’03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 237–252, New York, NY, USA, 2003. ACM Press.
6. Dawson R. Engler, David Yu Chen, and Andy Chou. Bugs as inconsistent behavior: A general approach to inferring errors in systems code. In *SOSP*, pages 57–72, 2001.
7. Norman Fenton, Paul Krause, and Martin Neil. Probabilistic modelling for software quality control. *Lecture Notes in Computer Science*, 2143:444–454, 2001.
8. Todd L. Graves, Alan F. Karr, J. S. Marron, and Harvey P. Siy. Predicting fault incidence using software change history. *Software Engineering*, 26(7):653–661, 2000.
9. Xiaohui Gu, Klara Nahrstedt, Rong N. Chang, and Christopher Ward. QoS-Assured Service Composition in Managed Service Overlay Networks. In *ICDCS’03*, 2003.
10. Ahmed E. Hassan. Mining software repositories to guide software development.
11. Z. Li, Z. Chen, S. Srinivasan, and Y. Zhou. C-miner: Mining block correlations in storage systems, 2004.
12. Madanlal Musuvathi, David Y. W. Park, Andy Chou, Dawson R. Engler, and David L. Dill. Cmc: A pragmatic approach to model checking real code. In *OSDI*, 2002.
13. Service Oriented Architecture. http://en.wikipedia.org/wiki/Service-oriented_architecture.
14. Martin L. Shooman. *Software engineering: reliability, development, and management*. McGraw-Hill, Inc., New York, NY, USA, 1983.
15. Software as a Service. http://en.wikipedia.org/wiki/Software_as_a_Service.
16. Statistics of services. <http://www.us-mission.ch/Press2005/0531WTO.htm>.
17. J. Wu, A. Hassan, and R. Holt. Exploring software evolution using spectrographs, 2004.
18. Yichen Xie, Andy Chou, and Dawson Engler. Archer: using symbolic, path-sensitive analysis to detect memory access errors. In *ESEC/FSE-11: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 327–336, New York, NY, USA, 2003. ACM Press.
19. Junfeng Yang, Ted Kremenek, Yichen Xie, and Dawson Engler. Meca: an extensible, expressive system and language for statically checking security properties. In *CCS ’03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 321–334, New York, NY, USA, 2003. ACM Press.