ISSN: 2321-7782 (Online)
Volume 3, Issue 3, March 2015
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

Web Services Threats, Vulnerabilities and Countermeasures

Razia Sultan[1]
Student
Shri Venkateshwara University
Lucknow, India

Dr. S. Q Abbas[2]
Director General
AIMT Lucknow
Lucknow, India

Abstract: ***This document gives information about the threats, vulnerabilities and countermeasures of web services. Securing web services is a complex task and it is very difficult to completely secure the web services, an engineered process of security consists of detailed plans and designs for security features and controls that support the delivery of solutions satisfying not only functional requirements, but also preventing misuse and malicious behaviour.***

Keywords: web services; threats; vulnerabilities; countermeasurures; security.

I. INTRODUCTION

Securing a Web service requires us to protect, as far as possible, all of its basic components, and their interactions along with the Web service life cycle, from the design to the operational phase. This is a difficult and complex task, due to the vulnerabilities of software component, the large number of attacks that can eventually exploit the vulnerabilities of a specific component, and to the interactions between the components themselves. It requires us to combine and enhance methods, tools, and techniques for networks, distributed systems, computers, and application security and adopt an engineered security process. This type of engineered process consists of detailed plans and designs for security features and controls that support the delivery of solutions satisfying not only functional requirements, but also preventing misuse and malicious behaviour. Main steps of an engineered security process are security planning, security requirements analysis, the design of a security architecture, secure coding, security testing, and security operations and maintenance. Each core practice encompasses specific methods and tools, or even specific methodologies. Threat modelling is a methodology used to ensure that a software component's security is duly analyzed and taken into account at design time.

Threat modelling is not a new concept, nor it is peculiar to Web services. At its most basic, threat modelling assumes that a software component might be subject to malicious attacks or inadvertent misuse and that these attacks may lead to loss or compromise of some valuable asset. At the same time, malicious attacks or inadvertent misuse arise by use of software components' vulnerabilities. So, it is compulsory to identify in a principled way the software component vulnerabilities, which can derive from wrong design choices, from implementation errors or from the lack of appropriate control, the ways in which such vulnerabilities could be exploited by a malicious attacker and finally the suitable countermeasures. W3C also postulated the need of developing a threat model specifically targeting Web services and suggested that such a threat model should be an integral part of a Web services security framework.

II. THREATS AND VULNERABILITIES CONCEPT DEFINITION

» ***Threat***

According to the Glossary of Key Information Security Terms of NIST , a threat is defined as: "Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, or individuals through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service, also, the potential for a threat-source to successfully exploit particular information system vulnerability."

» ***Vulnerability***

A vulnerability can be defined as a "Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source.

A vulnerability is also commonly defined as a flaw in a software component that can be exploited by an adversary, giving raise to a threat. The former definition is more general, since it considers not only software components vulnerabilities but so equally relevant organizational aspects. Vulnerability is a software mistake that can be *directly* used by a hacker to gain access to a system or network, while the exposure is a configuration issue or a software mistake that does not allow an attacker to directly compromise a system or a resource, but that could be used as a primary point of entry of a successful attack, and that violates a reasonable security policy.

» ***Attack***

An attack, referred to also as *intrusion* or *exploit*, can be defined as an assault to a system that derives from an intentional use of a vulnerability.

» ***Attack cost***

The cost of an attack can be defined as a measure of the effort to be expended by the attacker, expressed in terms of her expertise, resources and motivation.

» ***Incident -*** An incident is the result of a successful attack.

» ***Countermeasure***

A countermeasure can be thought as any organizational action or tool able to mitigate a risk deriving from one or more attack classes intended to exploit one or more classes of vulnerabilities.

Software components vulnerabilities can be broadly classified as:

1. *Software defects*. They can be further refined in:

- *Design flaws*, that is, design decisions made at the design time that create an inherently insecure system. An example of architectural level error for session-based systems, leading to a vulnerability, is the generation of easily guessable session identifiers (i.e. using a counter and not a random number).
- *Coding errors*, that can be traced back to the lack of appropriate controls in the code that may lead to a variety of errors including for example buffer overflows or race conditions.
- *Configuration errors*, They are becoming the major source of vulnerabilities.

This is not surprising, since software components are more and more driven by configurations which are becoming quite complex to manage.

Configuration errors can be further classified in:

• *Unnecessary (or dangerous) services*. Since it is usually easier to install a software component with its default configuration, systems are often configured to bring up services and allow connections that are not strictly required.

• *Access control misconfiguration*. Complex systems might have elaborate access control policies based on groups and/or roles and permissions. Properly configuring reference monitors, which is the engines enforcing access control decisions and enforcement, is a complex and error-prone task.

III. THREAT MODELLING

Threat modelling is a methodology to identify, assess and document the threats, attacks, vulnerabilities, and countermeasures in the context of a software component life cycle. The overall goals of threat modelling are to reduce security risks during design, development and operations and to help making trade-offs in key engineering decisions.

In a Web service setting, threat modelling might be applied to any software component of the Web service stack, from the HTTP processor to the Web application. While the components that constitute the infrastructure of a Web service, such as the HTTP processor, the XML parser, the SOAP processor, etc., benefit from being developed by large organizations which can afford the costs of security, Web applications are mostly developed as a custom component. Web applications are one of the weakest components of the Web service stack, since development teams often operate under tight time and/or resource constraints that may also include the lack of security trained personnel. The situation is worsened when security functions, such as authorization, are implemented by the Web application itself.

However, applying threat modelling to Web application requires to provide a structured framework for the security analysis. A first step in this direction is to organize threats and vulnerabilities in a set of categories, corresponding to security or security related functions. Such functions could be directly implemented by the application, or could be implemented by the infrastructure and used by the application. Depending on the application security requirements, vulnerability categories can be used during the application design phase to check which specific security functions must be implemented. At the same time, vulnerability categories can provide indications about the threats that could result from the exploitation of one or more vulnerabilities deriving from having decided to not implement one or more security functions.

Input validation Controls that the application should perform on its input: lexical, syntactical and type checks; input integrity; input origin (does it come from a valid user?)

Authentication Entity authentication checks: does the application properly authenticate the sender of the input data; does the application authenticate the users?

Authorization Does the application verify that the requesting users can legitimately access/use the resource they are requesting?

Configuration Management Does the application run with the least needed privileges?

Does the application connect to the proper database? If the application uses its own configuration data, are they stored and accessed in a secure way?

Sensitive Data If the application manages sensitive data, does the application use suitable techniques to protect confidentiality and integrity of data in transit and of data at rest?

Session Management Does the application manage sessions? If so, are sessions properly protected?

Cryptography Does the application properly use cryptography to protect data and messages confidentiality and integrity?

Exception management Does the application fail securely? Does the application properly manage error information reported to the user to avoid information leakage?

Auditing and Logging

Does the application keep a record of who did what?

Threat modelling is an iterative process that starts during the early phase of the design of the application and continues throughout the application life cycle. The iterative nature of the threat modelling process stems from the difficulty of identifying all possible threats in a single pass and from the need to accommodate changes to the application over its lifetime.

Building a threat model involves the following steps ,

1. Identify the valuable assets.

2. Define the security objectives.

3. Create an architecture overview of the Web application (or of the software component at hand).

4. Create a security profile of the Web application (or of the software component at hand).

5. Identify threats and vulnerabilities.

6. Document threats and vulnerabilities.

7. Rate the threats.

The final output of threat modelling is a threat model document. A threat model document consist of a definition of the architecture of the application and of a list of threats for the application scenario. A threat model document supports the various members of the project team in assessing the security architecture of the system, and provides a basis for a penetration test plan and for the code review activity.

The threat model constitutes the input for the identification and the evaluation of the suitable countermeasures, where the evaluation can encompass not only the countermeasures' technical aspects but also their cost/benefit ratio.

Identifying the assets

The goal of this step is to identify the assets that must be protected from attacks. It is worth noting that such assets could be resources directly managed by the Web application, resources that could be someway indirectly accessed through the application, as well assets related to the business, such as the company reputation, which do not have a direct electronic embodiment.

Defining the security objectives

Security objectives are strictly related with the definition of the level of confidentiality, integrity and availability required for the valuable information assets and for the information systems managing them. Defining security objectives, in turn, requires categorizing the relevant information types and information systems. As for this aspect, the U.S. Federal Information Processing Standard, Standards for Security Categorization of Federal Information and Information Systems identifies three security categories, confidentiality, integrity and availability, respectively. Security objectives derive from the estimate of the impact of a security breach, that is, a confidentiality, integrity and availability breach, on the valuable information assets.

Creating an architecture overview

The main goal of this step is to identify and document the functions of the application, its architecture, its physical deployment configuration and the subsystems that it uses.

Create a security profile of the Web application

In this step a security profile for the application is created, according to the chosen vulnerability categorization.

Decomposition and security profile

Crucial activities in this phase are the identification of the Web application entry and exit points, which broadly represent the applications' perimeter, and for each of them, the level of trust associated with the external software entities which the input originates from or the applications' output is directed to. An entry points is a location where data or control is transferred between the application being modelled and another subsystem. Trust level is basically related to whether the external

subsystem is authenticated or not and to the privileges it has. Trust levels are needed to identify data flows that are potentially malicious, and thus need to be more thoroughly validated by the application.

Identify the threats

In this step, the security or security related functions implemented or used by the application are analyzed in order to discover possible threats. According to Microsoft guidelines, this activity should ideally be carried out by a team consisting of application architects, security professionals, developers, testers, and system administrators. The main output of this step is a threat profile, which identifies the likely paths of attack. Attacks can be also analyzed, and categorized, with respect to the objectives an attack may intend to achieve. For example, Microsoft describes these objectives by using the STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of privilege)

Spoofing: The illegal access and use of a user's identitification or authentication information, such as the user's username and password.

Tampering (also called integrity threats) an unauthorized modification of data. Examples include making unauthorized changes to persistent data, such as that held in a database, or altering data as it flows between two computers over an open network, such as the Internet. Repudiation the ability of users (legitimate or otherwise) to deny that they performed specific actions or transactions. An example is a user performing an illegal operation in a system that can't trace the prohibited operations.

Information disclosure the unwanted exposure of private data.

Denial of service the process of making a system or application unavailable.

Elevation of privilege, it occurs when a user with limited privileges assumes the identity of a privileged user to gain privileged access to an application and thereby he/she has the ability to compromise or destroy an entire system.

For each threat category STRIDE provides guidelines as to the appropriate countermeasures to adopt in order to reduce the risk. Most common known threats can also be grouped based on the component which may be subject to the threat, that is, the network, the host, and the application. Such categorized threat lists can provide a more focused framework to conduct the security analysis.

Categorized lists of known threats include common well known threats.

For each category, in order to identify other possible threats, other techniques based on *attack trees* and *attack patterns* can be used. *Attack trees* represent the possible paths followed by an attack as trees. The root node of such a tree is the global goal of an attacker. Children of a node are refinements of this goal, and leafs represent goals that can no longer be refined. Attack trees provide a formal methodology for analyzing the security of systems and subsystems, to capture and reuse expertise about security, and to respond to changes in security requirements. Attack trees can become complex, especially when dealing with specific attacks. A full attack tree may contain hundreds or thousands of different paths all leading to completion of the attack. Although the creation of an attack tree may require substantial security expertise and practice, attack trees, once organized in a library, can provide a reusable knowledge base and so contribute to an engineered approach to security.

Attack patterns are a mechanism to capture and communicate the attackers perspective. They describe common methods for exploiting software that can occur in a variety of different contexts, and apply the problem-solution paradigm of design patterns . An attack pattern is based on the analysis of observed attack exploitations, and usually contains the following information:

• Pattern name and classification;

• Attack prerequisites;

• Attack Description;

• Targeted vulnerabilities or weaknesses;

• Method of attack;

• Attacker goal;

• Attacker skill level required;

• Resources required;

• Blocking solutions;

• Context description.

To promote the standardization of attack pattern description and their use in the secure software development, attack patterns are being enumerated and classified by the Common Attack Pattern Enumeration and Classification (CAPEC) initiative sponsored by the Department of Homeland Security (DHS) and led by Cigital.

Altogether, the information provided by attack trees and attack patterns about the methods of attacks, the required attacker skill level, and the resources needed to conduct the attack represent one of the perspectives to be taken into account when rating threats and vulnerabilities.

Rating threats

The final step of threat modelling is rating discovered threats based on the risk they pose to the identified valuable assets, whose confidentiality, integrity, and availability should be assured. Rating threats is not a simple task, because it depends on multiple factors, such as the set of existing known vulnerabilities, the estimate of the likelihood of attacks, the estimate of the potential impact of an attack against valuable assets, to mention just a few. In principle, each perspective requires to define possibly multiple metrics. As to Microsoft guidelines, they suggest to evaluate the risk as the product of the probability of the threat occurrence and the damage potential, which is an estimate of the adverse effect of a threat occurrence on the valuable assets.

Countermeasures

Countermeasures are the controls and techniques to adopt in order to mitigate or prevent the exploitation of known and/or unknown vulnerabilities of a specific software component. Countermeasures could encompass both organizational processes and technical tools. The adoption of a secure software development methodology or security design guidelines is an example of an organizational countermeasure. As for the identification of countermeasures in the threat modelling process, the security profile of the Web application provides the input to identify the countermeasures to adopt against the identified vulnerabilities and threats.

Microsoft guidelines define an initial list of countermeasures for each threat category described by STRIDE, or for each vulnerability category of the categorized threat lists, respectively. Countermeasures mainly take the form of security guidelines the application or the security subsystem should comply to (for example, the authentication subsystem should not transmit the password over the network). Countermeasures are further specialized depending on the specific system component under analysis, namely the network, the host or the application. As far as Web applications are concerned, we already observed that a vulnerability category can be viewed as security or security-related functions directly implemented by the application or used by the application and provided by the (security) infrastructure. In an ideal world, authentication, access control, auditing and logging, cryptographic key generation and management should be implemented as functions or services of a security infrastructure provided as an integral part of the middleware. In such ideal scenario the application should take care of

implementing by itself only the security controls and countermeasures strictly dependent from the application semantics which cannot be fully implemented by the infrastructure, such as application input validation, application configuration management, exception management. It is worth noting, however, that even though the security functions provided by the middleware are becoming more and more reach and complete, providing a general purpose, customizable authorization mechanism as a security infrastructure service is still an open issue. As a consequence, Web applications continue to implement by themselves authorization functions, relying upon authorization mechanisms provided by the Operating System or by the DBMS, such as RBAC-based authorization mechanisms.

Countermeasures, both organizational and technical ones, can be also classified according to when they are used during the Web application life cycle, namely:

• **Development-time countermeasures**. Examples of technical tools that can be used in the development phase are code analyzers, penetration test tools and vulnerability scanners. The first obviously requires the availability of the source code of the application or of the software component at hand, while the last can also be used for applications and or software components acquired from third parties, and for which the source code is not available.

• **Deployment-time countermeasures**. These countermeasures encompass the activities and the related supporting tools that allow to ascertain the correct configuration of the Web application and of the diverse software components of the Web service stack.

• **Run-time countermeasures.** Vulnerability analysis tools, including intrusion detection and prevention systems (IDS/IPS) and file integrity checkers are examples of tools in this category

IV. THREAT AND VULNERABILITIES METRICS

Once vulnerabilities are identified and classified, there is the need to determine their severity and hence, indirectly, the security (or the insecurity!) of the specific system at hand. Rating vulnerabilities, that is, defining a vulnerability metrics, is a difficult task. The severity of a given vulnerability depends upon several factors. These factors are related to the amount of resources needed by a potential attacker, including the skill that the attacker should have, and hence to the attack cost, and to the estimated extent of the damage to the organization's valuable assets in case of an exploitation of the vulnerability.

Dealing with known vulnerabilities, different vendors may adopt different methods for rating software vulnerabilities in their bulletins or advisories; the same situation also arises for different vulnerability databases. The consequence is that IT system and application managers, who are in charge of managing large and heterogeneous systems, lack consistent indications to prioritize the vulnerability to be addressed.

Vulnerabilities are classified into four categories, namely:

• **Critical:** A vulnerability whose exploitation could allow the propagation of an Internet worm without user action.

• **Important:** A vulnerability whose exploitation could result in compromise of the confidentiality, integrity, or availability of user data, or of the integrity or availability of processing resources.

• **Moderate.** Exploitability is mitigated to a significant degree by factors such as default configuration, auditing, or difficulty of exploitation.

• **Low**: A vulnerability whose exploitation is extremely difficult, or whose impact is minimal.

US-CERT Vulnerability Metric uses a quantitative metric and scores the severity of a vulnerability by assigning to it a number between 0 and 180. This number results from several factors reported by the users, including:

• The degree of diffusion of information or knowledge about the vulnerability.

• Whether incidents reported to US-CERT were caused by the exploitation of the vulnerability.

• The risk to the Internet infrastructure deriving from the vulnerability.

• The number of systems on the Internet at risk because of the vulnerability.

• The impact of exploiting the vulnerability.

• Whether the vulnerability can be easily exploited.

• The preconditions required to exploit the vulnerability.

As described on the US-CERT Web page, the factors above are attributed with approximate values that may differ significantly from one site to another, and hence users are suggested not to rely solely upon this metric for prioritizing vulnerabilities. However, the factors above are used by US-CERT for separating the very serious vulnerabilities from the large number of less severe vulnerabilities described in its database. Typically, vulnerabilities with a metric greater than 40 are candidates for US-CERT Technical Alerts. The factors above described are not all equally weighted, and the resulting score is not linear (a vulnerability with a metric of 40 is not twice as severe as one with a metric of 20).

Common Vulnerability Scoring System

The Common Vulnerability Scoring System (CVSS) aims at providing an open framework for measuring the impact of IT vulnerabilities. CVSS is based on a quantitative model in order to ensure repeatable and consistent measurement of the vulnerabilities' impact. Two common uses of CVSS are prioritization of vulnerability remediation activities and the calculation of the severity of vulnerabilities discovered on a given system. CVSS is composed of three metrics groups: base, temporal, and environmental, each consisting of a set of metrics. The base metrics group represents characteristics of a vulnerability that are constant over time and user environments. This group includes six metrics. The first one, the access vector metric, reflects the type of access (local/adjacent; network/network) which the attacker has to use in order to exploit the vulnerability. The second one, the access complexity metric, measures the complexity of the attack required to exploit the vulnerability once an attacker has gained access to the target system. The third one, the authentication metric, measures the number of times an attacker must authenticate to a target before exploiting a vulnerability. The fourth one, the confidentiality impact metric, measures the impact on confidentiality of a successfully exploited vulnerability. The fifth one, the integrity impact metric, measures the impact to integrity of a successfully exploited vulnerability. The sixth one, the availability impact metric, measures the impact to availability of a successfully exploited vulnerability.

The temporal metrics group tries to capture the threat deriving from the time-changing characteristics of a vulnerability. This metrics group encompasses three metrics, the exploitability, the remediation level, and the report confidence metrics, respectively. The first one measures the current state of exploit techniques or code availability, which may range from the exploit being only theoretically possible, to the existence of publicly available details about how to build code that performs the exploitation. The second one allows one to take into account the existence of temporary fixes (workarounds or hotfixes) or of official patches or upgrades. The third one measures the degree of confidence in the existence of the vulnerability and the credibility of the known technical details. The metrics above are optional and do not concur to the calculation of the overall score.

The environmental metric group captures the characteristics of a vulnerability that are relevant and unique to a particular user environment. These metrics are optional and do not concur to the calculation of the overall score.

The Collateral Damage Potential metric measures the potential for loss of life or physical assets through damage or theft of property or equipment and possibly the economic loss of productivity or revenue. The Target Distribution metric measures the percentage of systems that could be affected by the vulnerability. The Security Requirements metric allows one to re-weight the

base security metrics (confidentiality, integrity, availability), by taking into account the relevance of the corresponding security requirement for the affected IT asset to a user's organization.

The metrics described above require the involvement of different stakeholders and organizations' roles in the different phases of the application or system life cycle. Generally, the base and temporal metrics should be specified by vulnerability bulletin analysts, security product vendors, or application vendors because they typically have better information about the characteristics of a vulnerability than users. The environmental metrics, however, should be specified by users because they can more properly assess the potential impact of a vulnerability within their own environment.

SANS Critical Vulnerabiliy Analysis Scale

SANS Critical Vulnerability Analysis Scale Ratings ranks vulnerabilities using several key factors and varying degrees of weight, as reported by the users, such as:

• The diffusion of the affected product.

• Whether the vulnerability affected a server or client system.

• Whether the vulnerability is related to default configurations/installations.

• The IT assets affected (e.g. databases, e-commerce servers).

• The network infrastructure affected (DNS, routers, firewalls).

• The public availability of exploit code.

• The difficulty in exploiting the vulnerability.

Depending on the above factors, vulnerabilities are ranked as critical, high,moderate, or low. Critical vulnerabilities typically affect default installationsof very widely deployed software, and result in root compromise of servers or infrastructure devices. Moreover, the information required for exploiting them (such as example exploit code) is widely available to attackers. Further, exploitation is usually straightforward, in the sense that the attacker does not need any special authentication credentials, or knowledge about individual victims, and does not need to social engineer a target user into performing any special functions. High vulnerabilities are typically those that have the potential to become critical, but have one or a few mitigating factors that make exploitation less attractive to attackers. For example, vulnerabilities that have many critical characteristics but are difficult to exploit, that do not result in elevated privileges, or that have a minimally sized victim pool are usually 44 3 Web Services Threats, Vulnerabilities, and Countermeasures rated high. Note that high vulnerabilities where the mitigating factor arises from a lack of technical exploit details will become critical if these details are later made available. Moderate vulnerabilities are those where the scales are slightly tipped in favor of the potential victim. Denial of service vulnerabilities are typically rated moderate, since they do not result in compromise of a target. Exploits that require an attacker to reside on the same local network as a victim, only affect nonstandard configurations or custom applications, require the attacker to social engineer individual victims, or where exploitation only provides very limited access are likely to be rated moderate. Low ranked vulnerabilities have little impact on an organization's infrastructure. These types of vulnerabilities usually require local or physical system access or may often result in client side privacy or denial of service issues and information leakage of organizational structure, system configuration and versions, or network topology. Alternatively, a low ranking may be applied when there is not enough information to fully assess the implications of a vulnerability. For example, vendors often imply that exploitation of a buffer overflow will only result in a denial of service.

V. CONCLUSION

vulnerabilities are ranked as critical, high,moderate, or low. Critical vulnerabilities typically affect default installations of very widely deployed software, and result in root compromise of servers or infrastructure devices. Moreover, the information

required for exploiting them (such as example exploit code) is widely available to attackers. Further, exploitation is usually straightforward, in the sense that the attacker does not need any special authentication credentials, or knowledge about individual victims, and does not need to social engineer a target user into performing any special functions. High vulnerabilities are typically those that have the potential to become critical, but have one or a few mitigating factors that make exploitation less attractive to attackers.

References

1. J. Malcolmson, "What is security culture? Does it differ in content from general organisational culture?" in Proc. 43rd Annual 2009 International Carnahan Conference on Security Technology, 2009, 2009, pp. 361–366.
2. N. R. Mathiasen, M. G. Pedersen, and S. Bødker, "While working around security," in Pro. 3rd International Conference on Human Computer Interaction, New York, NY, USA, 2011, pp. 1–9.
3. F. B. Shaikh and S. Haider, "Security threats in cloud computing," in Proc. 2011 International Conference for Internet Technology and Secured Transactions (ICITST), 2011, pp. 214–219
4. M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, "A view of cloud computing," Commun. Acm, vol. 53, no. 4, pp. 50, Apr. 2010.
5. H.-Z. Wang and L.-S. Huang, "An improved trusted cloud computing platform model based on DAA and privacy CA scheme," in Proc. 2010 International Conference on Computer Application and System Modeling (ICCASM), 2010, vol. 13, pp. 33–39.
6. G. Cheng and A. K. Ohoussou, "Sealed storage for trusted cloud computing," in Proc. 2010 International Conference on Computer Design and Application (ICCDA), 2010, vol. 5, pp. 335–339.
7. F. Lombardi and R. Di Pietro, "Transparent security for cloud," in Proc. 2010 ACM Symposium on Applied Computing, New York, NY, USA, 2010, pp. 414–415
8. D. Lin and A. Squicciarini, "Data protection models for service provisioning in the cloud," in Proc. 15th ACM symposium on Access control models and technologies, New York, NY, USA, 2010, pp. 183–192.
9. X. Zhang, S.-Q. Lai, and N.-W. Liu, "Research on cloud computing data security model based on multi-dimension," in Proc. 2012 International Symposium on Information Technology in Medicine andEducation (ITME), 2012, vol. 2, pp. 897–900.
10. Virtualize: definition of virtualize in Oxford dictionary (British &World English). [Online]. Available: http://oxforddictionaries.com/definition/english/virtualize?q=Virtualiz ation#virtualize__4..
11. C.-H. Huang and P.-A. Hsiung, "Hardware resource virtualization for dynamically partially reconfigurable systems," IEEE Embedded Systems Letters, vol. 1, no. 1, pp. 19–23, 2009.

