Logic Vulnerabilities in Web Applications

Varun Patil (vcpatil@indiana.edu)

12th December 2014

Abstract

Web applications have become increasingly popular these days. Almost all the services are made available to the user through their respective web applications. Thus web applications deal with a lot of critical data. Hence, it is mandatory for the developers to ensure the complete security of the web applications.

The vulnerabilities in the web application are broadly classified into two broad categories viz. vulnerabilities that have common characteristics across the various applications and the vulnerabilities that are application specific. The focus in this paper is towards the later category of vulnerabilities. It is often observed that the vulnerabilities that are specific to the application are very difficult to identify and analyze. Also, there has been very limited research on this subject.

This paper presents an analysis of the different logic flaws that commonly exist in the web applications, various types of attacks that can be launched to exploit the vulnerability and the techniques and tools that are used to mitigate and avoid these flaws. The paper also presents some of the startling statistics that were uncovered during the process of this research.

Introduction

It is a well known fact that almost all the services are provided on the internet these days. The internet hosts a variety of applications including the application for the various banking and e-commerce websites as well as the different social media websites like Twitter and Facebook. All of these applications require the users to log on to the systems by entering their personal information.

Owing to the variety of the applications that the internet hosts, the number of users using these websites has also increased. It has been observed that the number of users has increased from 757 million in May 2004 to 3,035 million in June 2014 [1]. This rise in the user population means that the web applications have to deal with a deluge of data. Hence, there is an ever increasing need to ensure the security of the data because if such critical and private data falls in the hands of a malicious attacker it can lead to grave consequences. In case of the banking applications that are hosted on the internet, a breach in the security can lead to a large financial loss. Consequently, the bank will again have to invest a lot to introduce newer security features to safeguard against any future attacks.

With the increase in the standards of technology in the past decade, the complexity of the web application has increased exponentially. Unfortunately, this has also increased the number of attacks that have been launched on the web applications. The attackers have reinvented the wheel to explore newer vulnerabilities and compromise the security of the web application. On the other hand, there is a constant pressure on the developers to get their product into deployment and enable swifter transition to the markets. Owing to these periodic constraints, the developers often end up ignoring the security aspects of the application. Additionally, the managers try to save the financial resource and reduce the cost of the product by asking the software developer to perform his own security analysis of the product instead of hiring a security expert to do the same. This ignorance can lead to a huge monetary loss if some vulnerabilities in the applications are duly found and attacked by some malicious party.

Background

The vulnerabilities in the web application have been classified into two broad categories

1) Vulnerabilities that have common characteristics across different applications

2) Vulnerabilities that are specific to the application [2].

The first category of vulnerabilities are caused due to faulty input validation. These class of vulnerabilities is caused when the application depends on the user input for its critical functionality and the user inputs are handled without proper sanitization of the data. Cross-site scripting and SQL injection belong to the first category of vulnerabilities.

Cross-site scripting (XSS) is a type of of computer security vulnerability which enables an attacker to inject a client-side script on the web pages that are meant to be viewed by other users. Such an attack enables the malicious users to bypass the access controls like the same-origin policy. According to the vulnerabilities documented by Symantec in the year 2007 [3], 84 percent of all the attacks on the internet were carried out by using cross-site scripting.

SQL injection is a code injection technique that is usually used to attack the data driven applications. In this attack SQL statements are inserted into an entry field for execution. The cause for such an attack can be traced to user inputs which are not correctly filtered for string literal escape sequences and when the user input is not strongly typed and accidentally executed. In a 2012 study, security company Imperva observed that the average web application received 4 attack campaigns per month, and retailers received twice as many attacks as other industries [4].

The second category of the vulnerabilities is referred to as the logic vulnerabilities. It results from the faulty application logic. Consequently, business logic vulnerability allows an attackers to misuse the application by circumventing the business rules of the application. In a business logic flow, the attacker uses the legitimate processing flow of the organization that has a negative consequence to the organization [5].

So far the research related to the detection of the logic vulnerabilities in the web application has been very sparse. This is because this vulnerabilities are specific to the functionality of the application. As a result there cannot be any generic solution that can be implemented which would result in the detection of vulnerabilities in all the application [6]. It is very difficult to come with a solution for all the applications in the same domain like e-commerce. Since the business rules that an e-commerce website like Amazon applies will be different from those applied by a website like E-bay.

One possible approach can be to use the requirement specification and the design documents of the application to our advantage. Then one can find the parts of the implementation that do not respect the intended functionality of the application. It is very rare to find such documents and hence this technique fails to be implemented. Also, it is very difficult to expect any organization to make its code available for third party analysis. Consequently, any research pertaining to the automated detection of logic vulnerabilities in a web application is usually limited to informal discussions.

Common Logic Flaws

With the increase in the complexity of the website application, the developers have made a transition to the multi-modular websites that often are a result of the integration of various modules. Following are some of the common logic flaws that crop up during the development of such websites [9]

1. Authentication flags and privilege escalation

Every web application has a set of access control lists and certain admin rights. An admin is an authorized entity which has particular rights that enables the person to alter the data in a specific way. If an attacker gets these special rights he will use the rights to his advantage and can inflict some financial loss on the organization.

2. Critical parameter manipulation and access to unauthorized information/content
HTTP GET and POST requests are typically accompanied with several parameters which are submitted to the application. These parameters usually consist of some key-value pairs. An attacker can be able to successfully predict the values in this request and can launch an attack, like an attack on currency, in an e-commerce application.

3. Developer's cookie tampering and business process/logic bypass
Most applications have a tendency to retain the username and password of the user in a cookie. If such critical data falls into the hands of a malicious party they can gain unauthorized access to the user's account and end up misusing the account.

4. LDAP parameter identification and critical infrastructure access
LDAP parameters can carry business-logic decision flags which can be abused or leveraged by the attackers. Attackers can use them to bypass the business layer logic to inject some malicious script in the LDAP requests if there is insufficient validation of the parameters.

5. Business constraint exploitation
If the business logic rules in an application are not properly defined, the attacker can exploit these constraints. Consider, the use of a coupon in an e-commerce application. The coupon with a particular code is supposed to be used only once to avail the said discount on an item. However, an attacker can bypass the business constraint and use the same coupon again and again to avail its benefits.

6. Business flow bypass
A web application is usually guided by its flow. In an e-commerce application, you are only guided to an order confirmation page once you have paid a due amount. But an attack on this flow can enable an attacker to land on the order confirmation page without having made the necessary payments.

7.Exploiting client-side business routines embedded in JavaScript, Flash or Silverlight
Theses days almost all the web applications run on the rich internet frameworks. Many of these website have the business logic of the application embedded in the client-side of the application. In the absence of the proper security mechanism these findings can be exploited by an attacker.

8. Identity or Profile extraction

Web application require an user to register himself on a website in order to receive a particular service. At the time of registration, the website asks for certain personal details of the user. In case of a banking application. These details would include the bank account number. If a person with malicious intent gets his hands on these details, he can use them to carry unauthorized transactions.

9. Denial of Service

The denial of service vulnerability poses a huge risk. The attacker can make the application unavailable in a period of downtime. This would complete cut down the working of the application. This has the potential of causing a lot of financial loss to the owners of the application.

10. File or unauthorized URL access and business information extraction

Such an attack can allow the user to illegitimately access a file that contains some of the critical application data. In order to avoid such an attack, the access controls an the file should be properly defined. It is necessary to test out these access controls before the website is fully functional.

Attacks

The attackers usually develop a two step attack on any of the logic vulnerabilities in the web application. The first step of the attack is called the inject step and the second step is the exploitation step where the code injected in the first step is used inflict a negative consequence on the operation of the web application via the second step.

In the inject phase, the attacker focuses his attentions on finding some taint sources within the application. The injection is achieved by using the techniques like parameter manipulation, hidden-field manipulation, header manipulation and cookie poisoning. Parameter manipulation is the manipulation of the requests sent between the web application and the browser. Parameter manipulation can be achieved by tampering the form fields, cookies, HTTP Headers and the URL Query strings [13]. Some of the web applications have hidden fields embedded in a web application. These fields are used for communicating information within the web application and the server. Some of this hidden fields may contain some confidential information within them. Due to the poor coding practices these fields may be accessible to the attacker. The attacker can then stage an injection by viewing the HTML source code, changing the contents in the field and reposting the code back to the server to launch an attack. Cookie poisoning is when the attacker can access the information stored with the cookies to gain unauthorized access to the web application [14]. A header manipulation occurs when a data enters a web application via untrusted sources, usually through HTTP headers. This data is then included in a HTTP response header and sent to the user without any proper validation [15].

The techniques used in the exploitation step are HTTP request splitting, path traversal and command injection. HTTP request splitting is an attack where the attacker can force the browser to send arbitrary HTTP requests. As a result of this attack the browser is forced to load the malicious contents of the attackers HTML page [16]. A path traversal attack aims to access the files and directories that are outside the root folder. With the help of this attack the attacker can access all the files and directories in the file system including the application source code and the critical files whose access is limited [17]. In command injection the goal is the execution of arbitrary commands on the host operating system by a malicious application. The attacker is keen on extending the default functionality of the application without the necessity of executing system commands [18].

The statistics provided by the iViz Security [8] have identified e-commerce applications as the ones who suffer the most due to the logic vulnerabilities in them. In this section we will focus on three such attacks that have negative consequence on E-commerce web applications [19].

Three critical steps in a typical checkout process that involves a merchant, a cashier and a user are:
1) order initiation on the merchants server
2) payment transaction on the cashiers server
3) order confirmation on the merchants server.
In the first example we look at the attack on the currency. Initially an ordered was place for an Ubuntu notebook from the Ubuntu online shop by Canonical Ltd. by using the payment module RBS WorldPay. The currency associated with the order was British Pounds. However, during the checkout process, the attackers manipulated the parameter for the GET Request within the HTTP request header and changed the currency to U.S. Dollars. As a result, the researchers were able exploit the logical flaw within the organization by paying less for their orders.

In the later part of the paper the researchers demonstrate an attack on the orderId. This time an order was placed for a diaper game using the payment module Authorize.net Credit Card SIM. However, this time the users were successful in executing an attack which caused them to pay nothing as they replayed the token from a previous order on the same website. Consider a case where there are orders with OrderId '1001' and another with OrderId '1002'. Hence while making the checkout for OrderId '1002' the users ended up replaying the token issued while successfully checking out OrderId '1001'. Since they had already paid for the first order, they were issued the second order (with orderId '1002') without any payment.

For the third demonstration the researchers placed an order for chocolate pieces from a California chocolate online shop by using the PayPal payment module. In this case the researchers were able to engage in a header manipulation attack such they changed the merchantId of the chocolate shop owner in the requesting URL to their own merchantId. As a result, they ended up paying the amount due for the chocolates to themselves. In other words, having the chocolates ordered for free. It is very easy for any user to set up an account on PayPal.

After having placed orders for the three commodities and having received those commodities, the researchers promptly alerted the involved parties about these attacks and paid them whatever amount was due.

Detection Techniques

An automated scanner works fine for detecting the vulnerabilities that have common characteristics across different applications. It is successful in analyzing the syntax errors in the statements and hence can efficiently detect the attacks such as SQL Injection and Cross-site scripting (XSS). However, it falters when it comes to the detection of logic vulnerabilities. It is because it is not programmed to understand the logic of the programmer and identify the discrepancies.

There are two approaches to the detection of logic vulnerabilities in the web application. The first approach is by using the manual technique of detection and the other is by using an automated tool that would report a vulnerability after scanning through the code.

Manual testing is a widely used approach for the detection of logic vulnerabilities in the web application. Manual testing is effective in uncovering the logical flaws. However, it involves a person who has to make sure that all the web pages follow the flow and the application behaves in the intended way. This approach is labor intensive and time consuming. Since, the person is not able to use the pruning techniques that an automated approach would include.

In this section we will analyze the various prototypes that have been published for the purpose of automated detection of web applications:

1) Waler [2] works on a two step process. In the first step, the authors use dynamic analysis and observe the normal operation of the web application. As a result of this dynamic analysis, we can obtain a simple set of behavioral specifications. By using the knowledge of the typical execution paradigm of web applications, the derived results are filtered to refine the false positives. The second step involves the use of model checking over symbolic input to identify the program paths that are likely to violate these specifications, thus indicting the presence of a logic vulnerability.

2) MiMoSA [22] is tool that categorizes both the extended state and the intended workflow of a web application. Using this approach, the prototype can account for inter-module relationships as well as the interaction of the applications' modules with back-end databases. This prototype can effectively defend the multi-step attacks against the applications' workflow.

3) Swaddler [23] detects attacks when the software is at the deployment phase of its life-cycle. It first learns the normal behavior of the application and then monitors state variables at runtime looking for deviations from the normal behavior.

4) BLOCK [24] is a tool that learns model and invariants by observing HTTP conversations and then detects authentication bypass attacks. It does not attempt to violate the logic of the application but it proposes a stateful crawler with an input fuzzer.

5) Inte-Guard [25] aims at protecting multi-party web applications from exploitation of vulnerabilities in the API integration. InteGuard focuses mainly on the browser-relayed messages in which data values are exchanged between the parties through the web browser. In particular, InteGuard uses a passive model inference technique based on data-flow analysis and differential analysis to extract inter-services dataflow-related invariants.

Statistics

The vulnerabilities in a web application continue to be one of the leading causes of enterprise data loss. Inspite of the very high profile security breaches in the recent years, organization have failed to secure their applications with the necessary security measures. The average annual cost of achieving regulatory security compliance in multinational companies is 3.5 million dollars [7].

It has also been observed that Logic vulnerability continues to be the vulnerability that is exploited the most by the attackers. The vulnerabilities pertaining to the business logic category include weak password recovery, abusing the discount logic or coupons, denial of service attacks, price manipulation and the One Time Password(OTP) bypass.

Some of the other startling facts from the survey are:

- 99 percent of the Apps tested had at least 1 vulnerability.
- 82 percent of the web application had at least 1 High/Critical Vulnerability.
- 90 percent of hacking incidents were not publicly reported.
- 30 percent of the hacked organizations knew of the vulnerability that led to the breach beforehand.

- A very low correlation between Security and Compliance (Correlation Coefficient: 0.2) [8].

The last point of these statistics happens to be the most surprising. There should be a conscious effort from the application developers to increase the correlation between the security and compliance to an higher value. The study also found that the average vulnerability per website is 35. All these statistic can seem very inviting for an attacker.
Conclusion:
It is clearly evident that logic vulnerabilities in the web applications are one of the most attacked vulnerabilities. There has been an effort to come up with an automated solution to detect these vulnerabilities. Also apart from Waler most of the prototypes require a human security expert to operate these prototypes. The effectivity of Waler on large scale applications is still questionable as it was only tested on a few mini-projects. Hence, we are far from having a completely automated and platform independent solution to these problems. However, the researches that have been already conducted within this domain have set up a strong platform for any future solution.
Resources

[1] http://www.internetworldstats.com/emarketing.htm

[2] V. Felmetsger, L. Cavedon, C. Kruegel, and G. Vigna, Toward automated detection of logic vulnerabilities in web applications, in Proceedings of the 19th USENIX conference on Security, ser. USENIX Security 10. Berkeley, CA, USA: USENIX Association, 2010.

[3]During the second half of 2007, 11,253 site-specific cross-site vulnerabilities were documented by XSSed, compared to 2,134 ”traditional” vulnerabilities documented by Symantec, in ”Symantec Internet Security Threat Report: Trends for JulyDecember 2007 (Executive Summary)” (PDF) XIII. Symantec Corp. April 2008. pp. 13. Retrieved May 11, 2008.

[4] Imperva (July 2012). ”Imperva Web Application Attack Report” (PDF). Retrieved 2013-08-04.

[5] https://www.owasp.org/index.php/Business$_Logic_Security_Cheat_Sheet$

[6] GROSSMAN, J. Seven Business Logic Flaws That Put Your Website at Risk. http://www.whitehatsec.com/home/assets/WP bizlogic092407.pdf, September 2007.

[7] http://www.mykonossoftware.com/statistics.php

[8] http://www.securitybistro.com/?p=4966

[9] http://www.networkworld.com/article/2188238/security/the-10-worst-web-application-logic-flaws-that-hackers-love-to-abuse.html

[10] C. Bodei, L. Brodo, and R. Bruni, Static detection of logic flaws in service-oriented applications, in ARSPAWITS, ser. LNCS, P. Degano and L. Vigan‘o, Eds., vol. 5511. Springer, 2009.

[11] N. Jovanovic, E. Kirda, and C. Kruegel. Preventing Cross Site Request Forgery Attacks. In Proceedings of the IEEE International Conference on Security and Privacy for Emerging Areas in Communication Networks (Securecomm), pages 110, September 2006.

[12] B. Livshits and M. Lam. Finding Security Vulnerabilities in Java Applications with Static Analysis. In Proceedings of the USENIX Security Symposium (USENIX05), pages 271286, August 2005.

[13] http://www.cgisecurity.com/owasp/html/ch11s04.html

[14] http://www.dummies.com/how-to/content/hidden-field-manipulation-hacks-in-web-application.html

[15] http://www.hpenterprisesecurity.com/vulncat/en/vulncat/php/header$_{m}anipulation.html$

[16] http://projects.webappsec.org/w/page/13246929/HTTP%20Request%20Splitting

[17] https://www.owasp.org/index.php/Path$_{T}raversal$

[18] https://www.owasp.org/index.php/Command$_{I}njection$

[19] Fangqi Sun, Liang Xu, Zhendong Su: Detecting Logic Vulnerabilities in E-Commerce Applications.

[20] http://www.acunetix.com/blog/web-security-zone/logical-and-technical-vulnerabilities/

[21] http://www.academia.edu/381405/Security$_{T}esting_{A}utomated_{o}r_{M}anual$

[22] D. Balzarotti, M. Cova, V. V. Felmetsger, and G. Vigna, Multi-module vulnerability analysis of web-based applications, in Proceedings of the 14th ACM conference on Computer and communications security, ser. CCS 07. New York, NY, USA: ACM, 2007.

[23] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna, Swaddler: An approach for the anomaly-based detection of state violations in web applications, in RAID, ser. LNCS, C. Krugel, R. Lippmann, and A. Clark, Eds., vol. 4637. Springer, 2007.

[24] X. Li and Y. Xue, Block: a black-box approach for detection of state violation attacks towards web applications, in Proceedings of the 27th Annual Computer Security Applications Conference, ser. ACSAC 11. New York, NY, USA: ACM, 2011.

[25] L. Xing, Y. Chen, X. Wang, and S. Chen, Integuard: Toward automatic protection of third-party web service integrations, in 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013.