Breaking the WEB with logics

by Shahmeer Amir

Authorized Publishers

getwhitehats

DEFCON LUCKNOW

HACKERS DAY

CSA DEHRADUN cloud security alliance

Acknowledgements

This research paper is initially dedicated to my mother who often falls sick thinking that I may fall sick by investing time in the research I do and then solely dedicated all of the white-hat hacker community who are dedicated to make the internet safer and act as an immune system for the internet. I would like to acknowledge the following individuals for taking time to proof read this paper.

Proof reader credits:

Ajin Ibrahim

Zeeshan Sultan

Sajjad Ahmed

Shritam Bhowmick

Mukarram Khalid

Abstract:

This research paper focuses on the Modern Business Logic flaws present in web applications that pose threat to the application. This paper covers several steps to explain the hierarchy of business logic testing methodology emphasizing on each phase separately by means of an example. It also contains several business logic flaws in that are present in today's web applications and also an approach on developers should remediate those flaws from the web applications. The main goal of this paper is to help web application security researcher and developers differentiate between web application conventional vulnerabilities and the ones that are related to logical aspect. The purpose of writing this paper is to convince the reader that at a certain point of time in the future automated scanners will be advanced at the level that conventional vulnerabilities will not require human intervention and only Business logic vulnerabilities will be present to leverage conventional ones.

Contents

Intentionally left blank

Background overview:

Looking through the past lanes – an amid amount of application security researchers have done adequate & extra-ordinary analysis with different web vulnerabilities including but not limited to: Injections, Cross site scripting, Cross site request forgery some of which are most of OWASP top 10 vulnerabilities.

However, to describe Business Logic Vulnerabilities – there's an evident misconception (which will be delivered & cleared across the table through this paper); learning to which I dedicate this document to enumerate some of my personal experiences which enables security guru's, work research of other dedicated personnel's & my folks @Information Security to summarize & uplift security culture for more ground-breaking innovations with regards to information security. Some vulnerabilities which are often called to be business logic results of an unprecedented logic is termed as business logic, which have no impact at business at-all. This paper will reveal some aspects of such propaganda which are without a doubt - left unaddressed.

Moving ahead to detail in this paper are the methodologies which are used in business logic security testing, breaks work-flow of the application which is being tested in phases, notes several distinct business logic flaws in modern web applications, elaborates distinctive features of conventional security flaws v/s business logic security flaws & advises mitigation techniques which could be followed by application developers to maintain the security of an application.

I hope folks @Information Security would love the work & writes back as I'd be great to hear from you regarding any feedbacks & related intellectual work criticism – I invite them all.

Modern definition of Business Logic flaws:

For a long time, there has been a debate between what a business logic vulnerability really is, that debate has led many researchers in confusion, However the exact definition that stands out in all circumstances is the modern definition. The modern definition of Business Logic flaws states that

"Any function of an application that the application is or is not designed to do and that can be used in a malicious manner is called a Business Logic Flaw."

According to the definition, a business logic flaw can not only be subjected to attacks from attackers on itself, it can also be used to attack other applications using its own resources.

Using the generic aspect showed in the definition, we can classify a business Logic flaw in to two types.

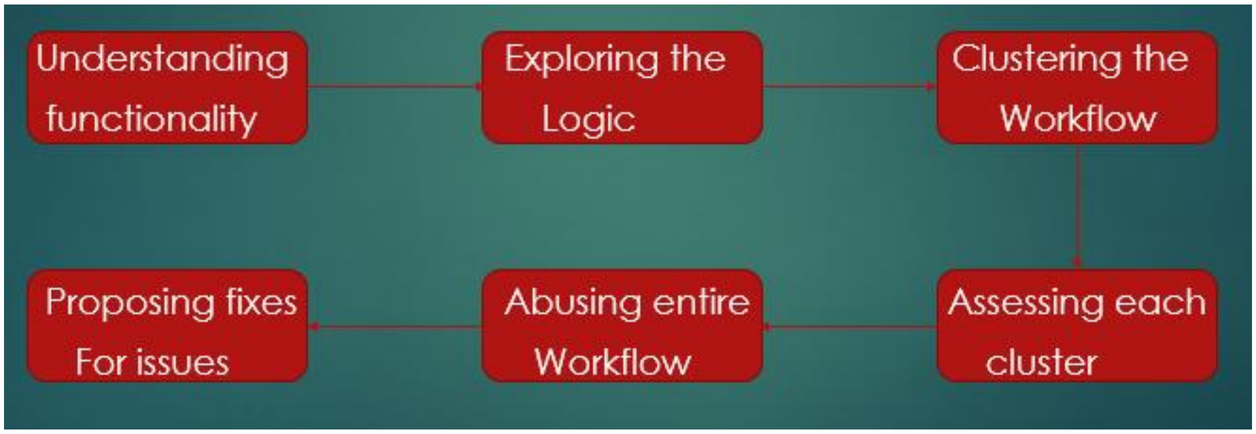
1. Flaws that aid an attacker to compromise the application itself
2. Flaws that aid an attacker to use the application maliciously

This definition is composed with the aim of describing & re-defining modern business logic flaws in a generic manner so a researcher can take it into reference while performing his own research.

Business Logic Testing Methodology:

The foremost refined way to test a business logic vulnerability can be clustered into six functional blocks. In order to test an application for a business logic vulnerability, it is advised to follow the methodology as defined.

1. Understanding the functionality
2. Exploring the logic
3. Clustering the workflow
4. Assessing each cluster
5. Abusing entire workflow
6. Proposing fixes for issues



Testing Methodology diagram

Each steps as defined is distinct and essential to test an application for a business logic flaw. To understand better how each steps contributes to the testing methodology, let us look at the flow in separate clusters on the basis of a renowned example of an Online auction site as explained in OWASP testing guide v4.

Exemplary Application of Methodology:

An online antique auction website lets its user's signup using unique usernames which are displayed on the side tab while a particular artifact is being auctioned. The site implements a rate limitation protection to protect user accounts against brute force attack by locking the user out and killing current sessions to prevent account compromise. A user is required to enter the credit card information before entering for a specific artifact auction so that buyers cannot back out if the item is sold. The site calculates the timeline of the bidding by computing using the number of bidders. To prevent browser caching of user data that is used once upon purchase the site also deletes all auction data once a user if logs out of the application.

Phase 1: Understanding the functionality

Since the basic functionality of the site is bidding, Bidding is basically an offer of setting a cost one is interested to pay for some artifact or a kind of demand that something be done. A cost offer is referred to as a bid. This term may be used in context of auctions, stock exchange, card games, or real estate. Internet is one of the most favorite platforms for providing bidding facilities, it is the most natural way of determining the price of a commodity in a free market economy.

Phase 2: Exploring the Logic

The logic that we are able to extract from the application's defined structure can be further classified into three segments basing it on how the entire application works

1. The application allows more than one user who may be able to bid on a site on a single item at a single time
2. The highest bidder in the lowest amount of time wins the bid and gets the artifact
3. The bid timeline increases as the number of users increase on the bidding site so does the amount of the initial bid

Phase 3: Clustering the Work Flow

The third phase is to cluster the workflow of the application based on the actions performed by the user and the responding actions taken by the application to entertain those actions

User	**Application**
User creates an account using a unique username	Application verifies the username's uniqueness and displays it as the user's identity
User logs in the account using the account credentials	Application monitors the number of trials made while logging in and stores them for session management
User enters their credit card details and enters into an auction	Applications calculates the number of users and puts in bid time
User wins or loses the bid	Application deducts amount from user's bank or credit card
User logs out of the application	Application deletes the credit card and session data including all the previous bidding information

Phase 4: Assessing each Cluster

In this phase, the previous information gathered from the clustering phase is used to asses each cluster separately and identify potential flaws in each of the cluster. So considering the clusters based on our example, we can narrow down 4 vulnerabilities from this information.

1. Displayed login Usernames depicts sensitive information disclosure
2. Rate limitation lockout DOSs even a legitimate user from logging in
3. Account lockout also terminates current session
4. Session termination deletes current user data

Phase 5: Abusing entire Workflow

According to the analysis done so far from the provided information, the fifth step is to use that information and collectively abstract a way to abuse the entire workflow of the application. The abuse would be based on a malicious user's actions causing harm to productivity of the application and the integrity of other users. Consider that the attacker misuses the application, in terms of its logic, here are the interpreted steps he would follow

Step 1: Attacker logs in
Step 2: Attacker enters into an auction
Step 3: Attacker collects usernames of bidders
Step 4: Attacker bids lowest
Step 5: Attacker launches brute force attack on all usernames
Step 6: Attacker buys artifact at lowest price

Phase 6: Proposing fixes for the issues

Since now it is known at this stage that how the application can be entirely misused, it is strongly advised to propose fixes, this step cannot be ignored because failing to propose a fix for a certain flaw can result in a greater risk if the flaw is not patched in an appropriate manner, because a business logic flaw is not a mere code based issue but a flaw in the behavior of the very application. So the patches listed below can remove this logical flaw from the web application:

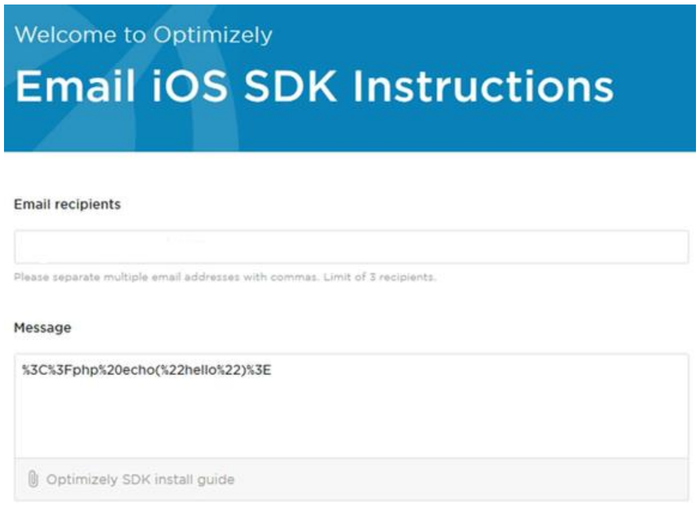
- Apply Captcha instead of account lockout
- Disable session termination
- Use email addresses as login credentials
- Apply Cache control header to prevent caching

Business Logic Flaws in Modern Web applications:

Application's business logic flaws that are present in modern web applications can vary from the application being exploited and the application exploiting other applications, In this section we will consider some examples of business logic flaws that have emerged over the years and developers are still not aware of these common mistakes. These findings were discovered in some of most renowned web application vendors that run bug bounty programs and are documented in this paper

1. Optimizely Note Scam attack

This is a relatively new kind of attack which takes of advantage of the web application's unfiltered message sending forms to deliver the attacker's custom code to another email address, at first glance this may not seem as dangerous, but it is a vulnerability that targets the vulnerable source web application vendor's reputation. i.e. 10000 emails having a scam page scripts are delivered to customer emails from microsoft's support email address would not be good for the company's reputation. A similar flaw was discovered in the optimizely's SDK code sending form where the user was allowed to include URI encoded script in the message field that was not striped when the email was sent and thus was included in the HTML of the email as stand-alone PHP script


Welcome to Optimizely

Email iOS SDK Instructions

Email recipients

Please separate multiple email addresses with commas. Limit of 3 recipients.

Message

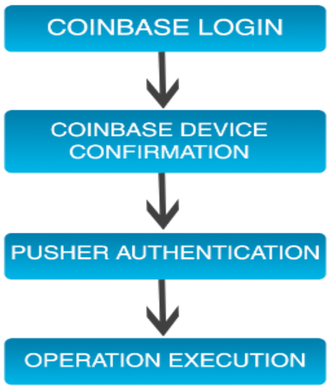
```
%3C%3Fphp%20echo(%22hello%22)%3E
```

Optimizely message panel picture

However, the limitation here is that the victim's email provider must not have filters in place at their end, which some common private email providers such as horde, Zoho mail do not. This is also a clear case where one endpoint service relies on the other for their own security and doing so both fail to implement countermeasures resulting in service compromise.

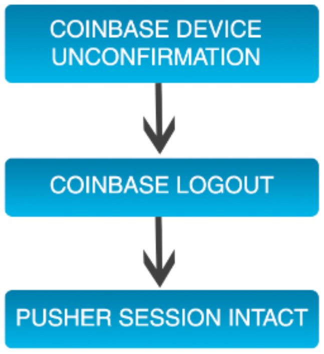
2. Coinbase Transaction Visibility Flaw:

When a single service uses a third party service to carry out some of its functions, the web application then incorporates more than one sessions in its cookies in order maintain and run the operation smoothly. Pusher is a renowned extension for applications which can be used to push notifications having to save the time of writing a separate piece of code. However, invalidating the pusher session which is separate from the session of the web application is a practice not followed by web application developers. Such a flaw existed in Coinbase a famous Bitcoin wallet which used Pusher to send out transaction notifications. A flowchart below shows how the Coinbase authentication mechanism worked



Coinbase authentication flowchart

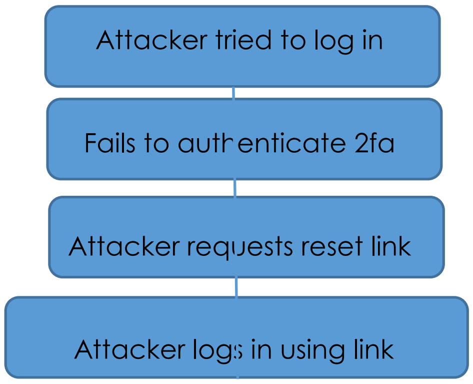
The normal process flow was not followed when the session was destroyed or the device was unconfirmed, the pusher session remains intact allowing the malicious user to switch tabs and view the transaction notifications. So if a legitimate user had two tabs opened and he was to logout from one tab in the browser, and the pusher session remained intact so the notifications were visible in the other tab. The de-auth flow is as follows



Coinbase de-authentication flowchart

3. Session based Two factor authentication bypass:

Two factor authentication is a secure practice to add an extra layer of security but due to some common development mistakes, it is often ignored at certain endpoints. The name of the web application vendor has remained hidden on discretion. Like in certain web applications, Session login is performed at the instance of external password change and mostly 2fa is ignored at that point. This flaw is the password reset link implementation of certain web applications that they ignore 2fa when engaging in session after the external password. So an attacker who does not have access to the victim's mobile device can still bypass the 2fa using he password reset link implementation. A workflow would be:


Attacker tried to log in
Fails to authenticate 2fa
Attacker requests reset link
Attacker logs in using link

Difference b/w Conventional Flaws and Business Logic Flaws:

As explained throughout this paper, the business logic vulnerabilities are specifically the ones that can be used to abuse the workflow of an application. Consider the fact, below are some of the common misconceptions about business logic flaws

- Any flaw that exists universally is not a Business Logic Flaw
- Any flaw that relies on user direct user intervention is not a Business Logic flaw
- Any flaw that relies on any other conventional flaw to be executed is not a business logic flaw

To better explain the case, below is a comparative difference table:

What a Business Logic Flaw is not	**What a Business Logic Flaw is**
It is not a code based flaw	It is a logical based flaw
It is not about what code is written	It is about how a code is written
It is not what an application does with the input	It is about how the input in handled and parsed
It is not malicious behavior of the application	It is the legitimate workflow used in a malicious way

In the of the difference statements it is easier to observe the relative difference between the conventional web application vulnerabilities and the non-conventional ones.

Mitigation techniques of Business logic flaws:

As we know that for vulnerabilities like SQL injection and Cross site scripting a number of prevention cheat sheets and filtration techniques are available so that the possibility of occurrence can be minimized. But when it comes to Business logic vulnerabilities there is no certain checklist that can be followed to prevent these flaws, since they are the abusive methods related the entire workflow of the application. However certain practices can be suggested to developers to harden their applications.

1. Good code:

Developers should be encouraged to write the code as neatly as possible, because writing a neat and refined code can up to some extent prevent the aspect of misuse in an application. This however does not guarantee the security of the application from workflow abuse but a neatly written code with appropriate comments can omit the possibility of misuse because it shows that the developer knows what he is doing.

2. Workflow review:

The best way to avoid any kind of business logic vulnerability from occurring in an application is to find it yourself. The workflow of an application should be reviewed multiple times from an aspect of an attacker, malicious behaviors can only be patched by reviewing the workflow more than one times.

3. Multiple overviews:

Another technique to avoid business logic flaws in an application is to have extra pair of eyes review the code multiple times. This is an optional but a very crucial practice that can aid in prevention of business logic vulnerabilities. This is however mostly disregarded in code reviews.

Conclusion:

There is a distinct possibility that in the near future automated tools will be highly advanced and intelligent to detect and fix conventional vulnerabilities and the attackers would be left with only business logic flaws to compromise the application. So it is about time that we should be training ourselves to learn how to detect these kinds of flaws and share our research on public platforms. Business logic flaws are indeed the future of web application vulnerabilities

In order to bring back the originality of work related to web vulnerabilities which grow from the soils of Business Logic Security – the temptation to water the topic gave me a thrill to invest my time & subjectively analyze the sheer abilities of discovering unattended business logic security flaws around modern web applications – both to defend & for offense at the cost of hunting hilarious bugs.