DESIGN AND DEVELOPMENT AUTOMATED DETECTION OF SESSION FIXATION VULNERABILITY IN WEB APPLICATION.

SUPERVISED BY:

MR. MD. MARUF HASSAN
ASSOCIATE PROFESSOR
DEPARTMENT of SOFTWARE ENGINEERING
DAFFODIL INTERNATIONAL UNIVERSITY

SUBMITTED BY:

SHANTANU DEY ANIK
181-35-2368
DEPARTMENT of SOFTWARE ENGINEERING
DAFFODIL INTERNATIONAL UNIVERSITY

This Thesis report has been submitted in fulfilment of the requirements for the Degree of Bachelor of Science in Software Engineering

Fall 2022

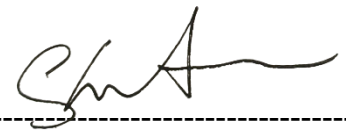
APPROVAL

This thesis titled on "**Design and Develop automated detection of Session fixation vulnerability in Web Application.**", submitted by **Shantanu Dey Anik (ID: 181-35-2368)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfilment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

BOARD OF EXAMINERS

---------- **Chairman**

Dr. Imran Mahmud
Head and Associate Professor
Department of Software Engineering
Faculty of Science and Information
Technology Daffodil International University



---------- **Internal Examiner 1**

Md. Shohel Arman
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

---------- **Internal Examiner 2**

Khalid Been Badruzzaman Biplob
Lecturer (Senior)
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

---------- **External Examiner**

Md. Tanvir Quader
Senior Software Engineer
Technology Team
a2i Programme

DECLARATION

It's been declared that this thesis including all the research-based experimental works has been completed by me under the supervision of Md. Maruf Hassan (Associate Professor). Department of Software Engineering of Daffodil International University.

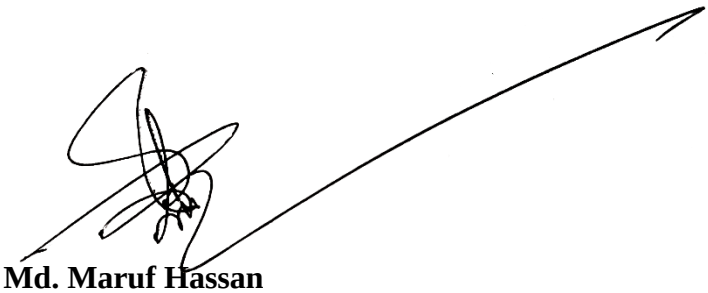
I also declare that neither this thesis nor any part of this whole research-based experiment has been submitted elsewhere for the award of any degree.

Shantanu Dey Anik

181-35-2368

Department of Software Engineering

Daffodil International University



Md. Maruf Hassan

Associate Professor

Department of Software Engineering

Daffodil International University

ACKNOWLEDGEMENT

Whose blessings begin with the name of the almighty god who created me and the entire universe directed completion of this dissertation

I am extremely grateful to Daffodil International University for assisting me in completing my Bachelor of Science degree through this research. I am very grateful to our supervisor, Mr. Md. Maruf Hassan, Associate Professor, Department of Software Engineering, Daffodil International University. In addition to thanking Dr. Imran Mahmud, Associate Professor & Head in Charge of the Department of Software Engineering, Daffodil International University, for motivating me to conduct quality research, I would like to thank my teachers for their ongoing support throughout this undergraduate program. At the same time, I had to cope with implementation challenges, and my supervising teacher's encouragement and guidance during these times was genuinely unique. His superb scientific guidance and encouragement, as well as his enthusiastic supervision, enabled this thesis to be completed. This would not have been feasible without his unwavering assistance. Many thanks to my classmates for their encouragement and assistance throughout this project. Certainly, and above all, I am grateful to my parents for their unwavering support.

ABSTRACT

Session Fixation is one of the most serious broken authentication vulnerabilities for a web application, and it is now listed as Identification and Authentication Failures in OWASP's Top 10 Web Application Vulnerabilities for 2021. A Session fixation attack is actually performed on a web application that has a lack of proper session management. Secure session management implementation calls for a thorough, all-encompassing strategy and should be incorporated as an integral module during the web applications' design and development lifecycle. A successful session fixation attack can be harmful to the user and also to the application. Some of the solutions and detection models are already proposed but require extensions from existing tools. This paper explores session fixation, which is a unique instance of a session management issue. The major goal of this paper is to propose a design for the automation tool for the detection of session fixation vulnerability based on useragent.

Keywords: *Session Fixation, Session Management, OWASP, Automation tool, Useragent*

Table of Contents

List of Figures

List of Tables

CHAPTER 1
INTRODUCTION

Every second, more people utilize the internet and web applications have emerged as one of the most widely used platforms for user communication, information sharing and financial transactions. Due to the vast quantity of information, they now deliver, Web applications have evolved from simple read-only to interactive and complex. Due to its widespread use, it was challenging to prevent unwanted access to confidential data that would result in irreparable loss. A web application that encourages daily use by users without the trouble of logging in every time. But this is what can put them at risk.

1.1 Background & Motivation

A session is a persistent method to interact with a web application without any hassle. Session fixation is an attack that can be performed on web applications due to an access control vulnerability. The attack that tricks a user into using a Session identifier (SID) has already been fixed by the attackers. After users have accessed the application using that SID, attackers use the same SID to bypass authentication. Then the attacker can perform malicious activity as the authorized user permitted to access the web application. This vulnerability arises when session management is not properly controlled in web applications.

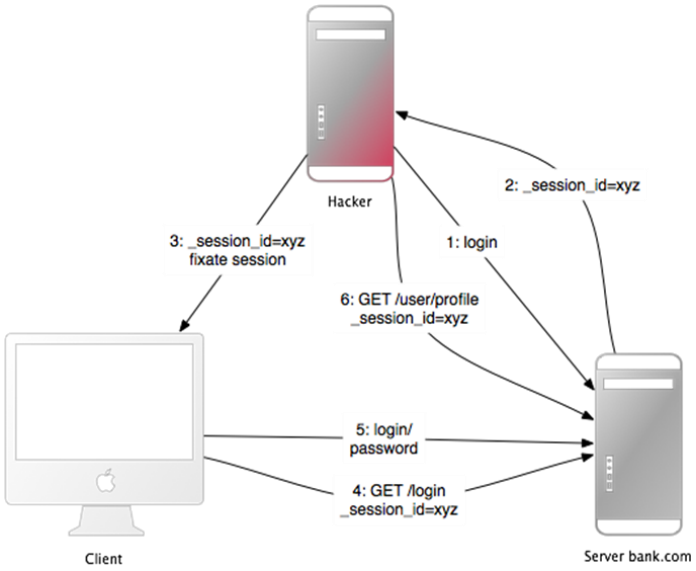


Figure 1.1 Session Fixation Attack

Session fixation attack can be performed by crafting Session ID in URL, hidden value in form, cookie attribute, other vulnerabilities like CSRF and XSS to forge the Session ID. Session fixation attack affect the victim as well the web application.

Every day, more and more things are becoming modern. As a result, digitalized banking transactions, online stores, patient portals for hospitals, and other internet services are growing rapidly. A user must have authorized to the web application in order to use them. The web application initiates a Session ID to monitor and record the user's actions. If appropriate session management is broken on the web application, an attacker could utilize the Session ID to pose as the victim and carry out nefarious operations which is session fixation attack.

Session ID is the main element to find out the vulnerability. Till now security specialists proposed some mitigation techniques to prevent session fixation attacks. One of them are using User-Agent. Though many automated detection solutions are proposed, but very few worked with the User-Agent.

A successful session fixation attack may be extremely destructive to both the business and the single user. It can result in ruined business relationships, illicit money transfers, suspicious interconnections, and data theft even when the legitimate credential is unknown. In consideration of this, this paper proposed an automated method for detecting web applications' Session fixation vulnerabilities using User-Agent.

1.2 Problem Statement

Session Fixation vulnerability detection work is currently available. We learn that they are not paying attention to the user agent verification, which is one method to prevent the session fixation issue that security researchers have already identified. There are various detection systems already in existence, but they are ineffective against the stated mitigation strategies. Testing our suggested model against the mitigation strategy will make it more useful.

1.3 Research questions

- What is Session and How does it work?
- How does a Session Fixation attack works?
- Is there any automated detection solution against proposed mitigation techniques?
- How to detect session fixation vulnerability automatically using User-Agent in web applications?

1.4 Research Objectives

Below is a list of the goals and objectives for this paper:

- Reduce the chance of Broken Authentication by detecting Session Fixation Vulnerability.
- Increase the effectiveness of session fixation vulnerability detection.
- To determine if the web application has incorporated the mitigating strategies.
- Reduce time to detect session fixation vulnerability.

1.5 Research Scope

Even though session fixation is an old problem, some web applications still rely on it. A session fixation these days can be very impactful for a web application and its users. Session fixation was categorized as Broken Authentication in the OWASP Top 10 2017 but is now categorized as Identification and Authentication Failures in the OWASP Top 10 2021. CVEs built on CWE-384 are still operational. The suggested mitigation technique's detection can therefore be highly helpful in identifying web application vulnerabilities. There are a number mitigating strategies, including user-agent verification, IP binding, proxy servers, framework level protection, and code level countermeasures. However, only a small number of automated detection solutions are evaluated against these mitigation strategies.

1.6 Thesis Organization

This thesis is organized as follows: Chapter 1 outlines the thesis background, motivation of the research, problem statement, research questions, research objectives, research scope, and thesis organization. Complete reviews are provided from previous studies in chapter 2 and are divided into the background and summary. Chapter 3 is divided into the following sections: detection model overview; validating Session ID acceptance in URL; storing unauthorized Session ID; logging in to the application; comparing sessions; exchanging sessions; flow chart; and vulnerability detection algorithm. In chapter 4, we divided it into the implementation overview, information gathering, attack generating module, response analysis, report generating module. In chapter 5, there are results from tests. In chapter 6, conclusions and limitations of our work are elaborated.

CHAPTER 2
LITERATURE REVIEW

In this chapter, we will give an overview of previous work on session fixation vulnerability. When functions linked to a user's identity, authentication, or session management are not implemented appropriately or are not adequately safeguarded by an application, identification and authentication problems might occur. Attackers may be able to exploit identification and authentication failures by compromising passwords, keys, session tokens, or other implementation vulnerabilities to temporarily or permanently assume the identities of other users. We reviewed certain papers and concluded that earlier work on session fixation was inadequate. Yes, there are numerous model methodologies and tools available. However, when we check, just a few of them work with the most recent protection against session fixation. Where the security researcher recommends it. In this chapter, we will discuss our evaluated work and present a summary of its findings.

2.1 Background

Broken access control is a major vulnerability for any website, and it is considered among the top ten most dangerous vulnerabilities. OWASP's top ten (OWASP top 10). When access control was breached, these unauthorized individuals Users have the ability to read stuff that they are not permitted to see and execute unauthorized tasks. An attacker can also destroy or take over site administration. In the case of the faulty access control, there are several vulnerabilities, including improper authorization and improper access. Cross-Site Request Forgery (CSRF), Improper Control of Dynamically Managed Code Resources, Session Fixation, Sensitive Information Inclusion in Source Code, Direct Request ('Forced Browsing') (OWASP top 10) (Lukanta, 2014). Authenticating a user, or otherwise establishing a new user session, without invalidating any existing session identifier gives an attacker the opportunity to steal authenticated sessions. (CWE 384)

Session fixation and Cross-site Request Forgery (CSRF) are major attacks inflicted on the session management mechanisms. An attacker in a session fixation attack prepares a session ID with which a victim is forced to log into a target web application. After a victim has logged in, an attacker can access the web application with the victim’s privilege. (Takamatsu 2012) (Lukanta 2014). According to a report from WhiteHat Security, 14% of web applications were vulnerable to session fixation and 24% were exposed to CSRF as per Yusuke Takamatsu.

There is some detection mechanism for Session fixation mechanism has been proposed. To detect session management vulnerabilities, some developed a vulnerability scanning tool extending an existing open-source tool (Lukanta 2014). Virginia Mary Nadar

proposed an enhanced model that can detect two attacks within the same simulation environment with updated rule libraries. (Nadar 2016). They proposed an enhanced detecting model that can detect two attacks, which are cross-site request forgery attack, broken authentication and session management attack within the same simulation environment with updated rule libraries and also have proposed an effective test environment has been proposed (Nadar 2018). Yusuke Takamatsu, Yuji Kosuga, Kenji Kono proposed a technique that automatically detects session management vulnerabilities in web applications by simulating real attacks. Their technique requires the test operator to only enter a few pieces of basic information about the web application, without requiring a test environment to be set up or detailed knowledge on the web application. (Takamatsu 2012) (Rahul Kumar 2014). Researcher proposed a new static analysis of a web app which verifies the immunity against session fixation attack in PHP based web application. According to their approach they developed a static analyzer called SAWFIX which detection accuracy & processing time is promising. (Amira 2017) They discussed about how to manual fuzz & detect vulnerabilities which failed to detect by automatic scanning tool in a web application. This paper will also help to know the pattern & characteristics of a vulnerable web by performing manual testing. False positives are the main disadvantage of automation. Main goal of the paper is to detect false negative vulnerability by manual testing (Amira 2021)

However, as well as many detections mechanism so of the work has proposed protection mechanism. Michael Schrank, Bastian Braun, Martin Johns, Joachim Posegga discussed about to method how the session fixation attack can be done and the proposed a model to implement proxy server between client and application, which will provide protection against session fixation attack. (Schrank 2010). In revision they have proposed protection of framework level (Schrank 2011)

Given all of the above work, we may conclude that it is past time to take some steps to detect Session Fixation Vulnerability automatically. Some security researchers recommend that we check the useragent to be secure from this vulnerability. As a result, we created an automated detection system for session fixation vulnerabilities based on user agent analysis. Only automated session fixation vulnerabilities in web applications will be detected by our proposed method. Based on the user agent, we will detect this issue.

2.2 Summary

Session fixation attacks pose a significant risk to web applications. They can enable an attacker to perform unauthorized actions on behalf of a user without their knowledge. Successful session fixation attacks can be disastrous for both enterprises and users. They can harm client relationships, cause unwanted financial transactions, and steal data. This approach does not take the user's identity, but it does use the user to perform an action without their permission. There are numerous strategies to prevent this attack. However, the majority of security researchers advised using useagent checking to defend against this attack. In practice, we observe relatively few techniques that operate with this useragent. However, upon more investigation, we realized that their proposed model is not totally accurate. Taking everything into account, we present a novel method for quickly and accurately finding Session Fixation Vulnerable Web Applications based on User Agent.

CHAPTER 3
RESEARCH METHODOLOGY

In this chapter, the model will go over each and every stage of the methodology for finding session fixation vulnerabilities, which will help to follow along with the steps. Here demonstrate about detection model overview, request for the URL, extracting form data Session ID crafted in URL check, logging into user account using different browser, compare unauthorized and authorized Session ID, checking Session ID for same user from different browser, exchanging Session ID from one browser to another. So, this chapter will help to understand the model.

3.1 Detection model Overview

In this paper the implementation solution based on two different user-agent to detect the Session fixation vulnerability automatically. In figure 2.1 represent of general approaches

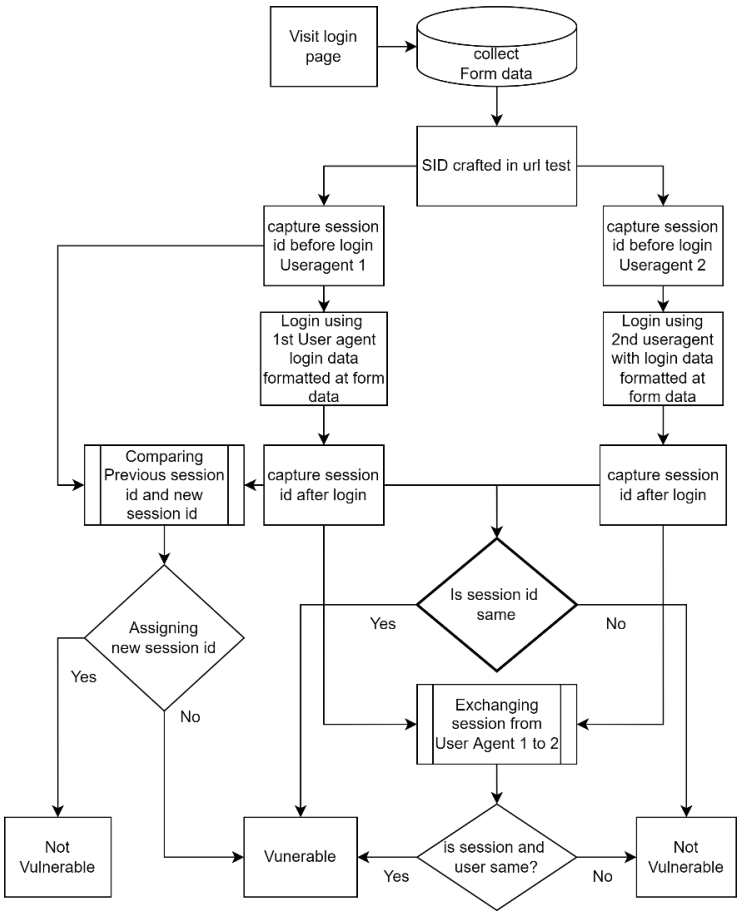


Figure 3.1: Automated Detection model for session fixation vulnerability

Each step is carried out iteratively in the figure. First, a GET request will be sent to the login page URL and collect the form data, because sometimes SID can be found hidden in the form value. After modifying the form data merged with the correct login information, it will be stored for further action. A GET request will be sent to the URL, which will capture the new Session ID and then initiate a new GET request to the same URL crafted with the captured Session ID and analysis if it accepts crafted Session ID. Then a new GET request will be performed using two different user agents separately and the session data will be stored correspondingly. Then two login requests using that user agents with already stored modified form data and store unauthorized Session ID will be sent respectively. If the web application doesn't reply with a new Session ID, that means the web application is vulnerable to a session fixation attack. The authorized Session ID for separate two user agents will be checked for the same user. If they are same, then the web application will be considered vulnerable. After that, a new GET request will be sent to the authorized page of the web application via 2nd user agent with the authorized Session ID from 1st user agent. If the web application returns the same Session ID as 1st user agent, then the web application will be considered vulnerable at Session Fixation.

This paper proposes those methods that have

- **Gather and modify login form data:** Send request to Grab login from data, modify data and store.
- **Validating Session ID acceptance in URL:** Session ID will be crafted with URL, check it accepts or not.
- **Storing Unauthorized Session ID:** Unauthorized Session ID will store for authentication.
- **Login to application:** Login using Different user agent, modified login form data and previous Session ID
- **Comparing session:** Comparison between unauthorized and authorized Session ID will be done.
- **Exchanging session:** Visiting authorized page from 2nd user agent using 1st user agents authorized Session ID.

3.2 Gather and modify login form data

As this is an automated design, it should be logged in automatically using valid user credentials. As we know, login form values are sent as data through an http request to be authenticated. So, in Figure 3.2, it is shown how the form data will be grabbed and modified. Firstly, a request to the login page will parse the HTML source. From there, the login form field and value will be gathered. Then modifications will be made with login credentials and stored. so that it could be later used to log in.

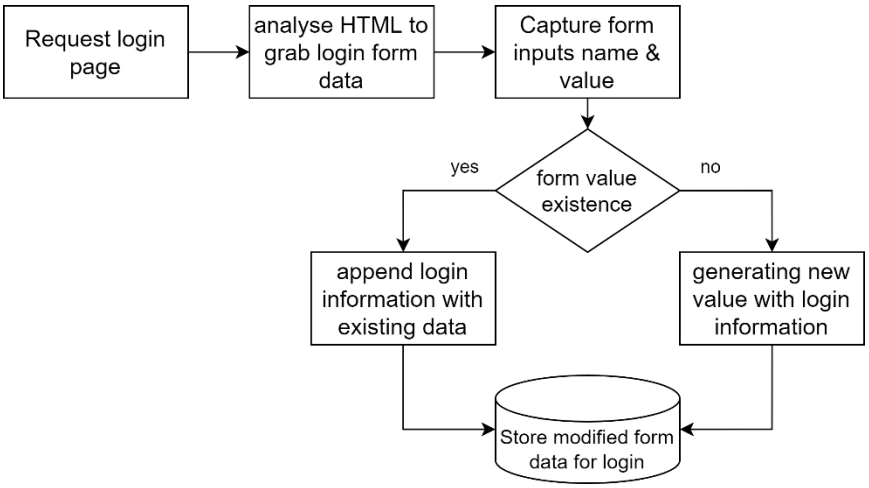


Figure 3.2 Gather and modify form data

3.3 Validating Session ID acceptance in URL

This module checks for the Session ID value in the URL. Because most attacks happen by sending a URL with a Session ID to the victim [Yusuke 2012]. So, a new get request will grab the first Session ID and store it. Again, a new request will be sent with that Session ID in the URL, and then check if the server replies with a new Session ID or not. If the new Session ID and previously stored Session ID are the same, then it's an issue. Illustrated in Figure 3.3

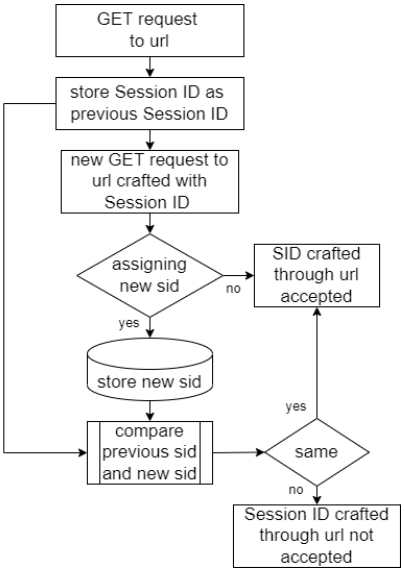


Figure 3.3 Checking Session ID acceptance in URL

3.4 Storing Unauthorized Session ID

This module will send a GET request to the URL and store the Session ID provided by the web application. Stored Session ID will later use to be authenticated and compared with authorized Session ID. This module will work on two different user agents respectively.

3.5 Login to application:

In this module, a request will be sent with the login form data (figure 3.2) and Session ID (figure 3.4), which are already stored. Then this will capture the redirected URL to be verified as authorized or not. because an unauthorized user information will return to the same login URL again. If the redirected URL is not the same as the login page URL, then it will grab the Session ID from the response from the web application. If the server doesn't return any new Session ID, then it will request to redirect the authorized page using the previously stored Session ID to ensure the Session ID is valid. In Figure 3.5 it demonstrates that after submitting login info, if it returns a new Session ID, then it will request to the redirected URL using the new Session ID. If it doesn't redirect to a new page, then the new Session ID will be stored. This module will work on two different user agents respectively.

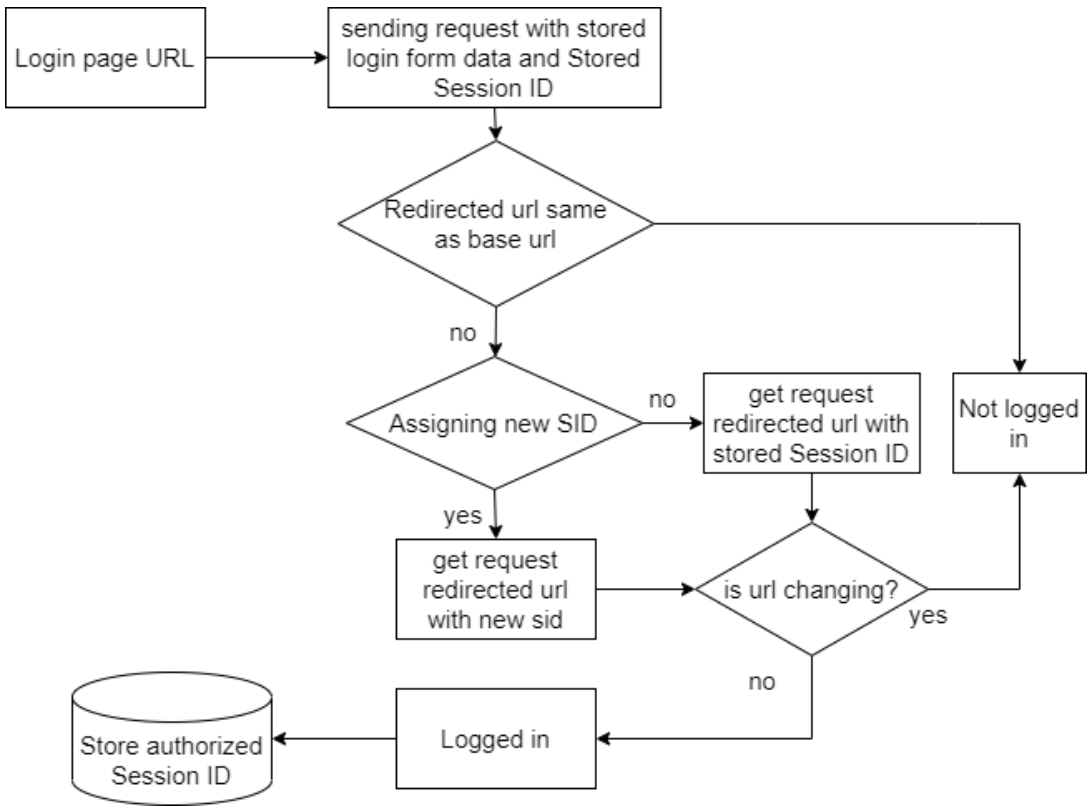


Figure 3.5 Login to application

3.6 Comparing Session

In this module, it compares between an unauthorized Session ID (Figure 3.4) and an authorized Session ID (Figure 3.5). If they are similar, then Similar Session ID is true and if not, then it is false. This

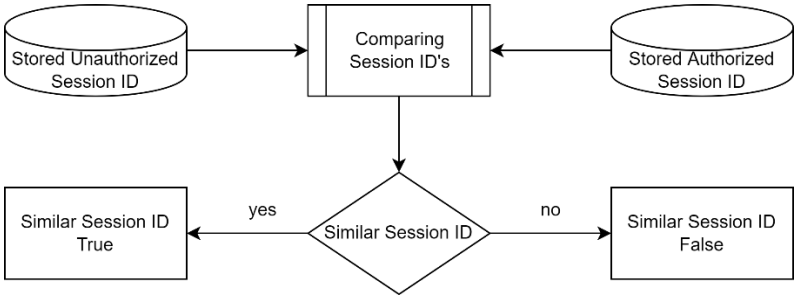


Figure 3.6 Comparing session

3.7 Exchanging Session:

Here, the stored authorized Session ID and URL of 1st user agent will be used to verify if the web application accepts that same Session ID for the authorized user. So, in Figure 3.6 it demonstrates that a new request from the 2nd user agent will be sent to the authorized page URL with the stored authorized Session ID of the 1st user agent. If the response redirects to a new URL, then that Session ID is not accepted by the 2nd user agent. If it relies on the same page, then it will check if the new Session ID is initiated or not. If not, then the stored authorized Session ID is accepted.

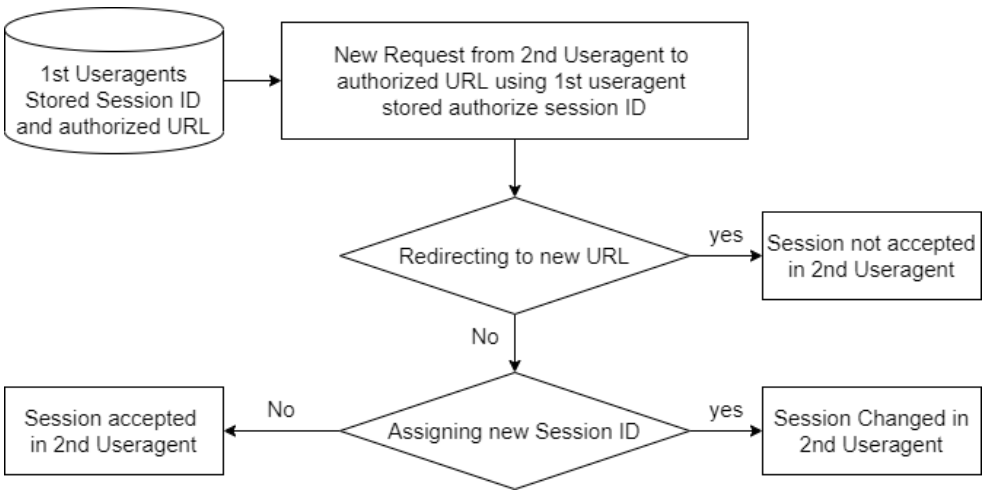


Figure 3.7 Exchanging Session

3.8 Vulnerability detection Algorithm:

So, vulnerability detection algorithm will work with gathered information from the previous modules

```
1.    #check for Session ID acceptance via URL
2.    if sid in url accepts then
3.       url_sid_accpetance = True
4.    else
5.       url_sid_accpetance = True
6.
7.
8.    #This will check for both useragent
9.    if authorized sid same as loginpage_sid:
10.    same_session_id = True
11. elif no new sid:
12.    similar_session_id = True
13. else:
14.    similar_session_id = False
15.
16.
17. #Comaprison between authorized session of both useragent for          same user
18. if first useragent authorized sid same as  second useragnet authorized sid:
19.    both_browser_same_session = True
20. else:
21.    both_browser_same_session = False
22.
23. #after sending first useragent's session to authorized page                from second
    useragent
24. if redirected url same as second browser authorized page url:
25.    if no new sid initiated:
26.       exchanging_session_accepted = True
27.    else:
28.       exchanging_session_accepted = False
29. else:
30.    exchanging_session_accepted = False
31.
32.
33. #Vulnerablity result generation
34. if same_session_id or both_browser_same_session or
    exchanging_session_accepted is True
35.    Result = "Vulnerable"
36. else:
37.    Result = "Not Vulnerable"
```

CHAPTER 4
IMPLEMENTATION AND EVALUATION

Implementation overview

An automated method for detecting session fixation vulnerabilities is implemented in this paper. All of the Session Fixation modules' implementation have been briefly discussed in this chapter. The proposed model is broken down into four sections in this paper, including information gathering, attack generation, response analysis, and reporting. For these methods, we have some key portion such as gather and modify login form data, Capture Session ID, Comparing Session ID, result generation.

4.1 Information Gathering (Gather and modify login form data)

To be logged in, a web application login form and its data are needed. In this module, the login form has been parsed from the URL given. To parse the login form from the html source, a python package called beautifulsoup(bs4) has been used. If found, it will categorize all data in "key:value" format in a dictionary, then it will find only the username/email and password fields, replace the value with the login credential received by the user-provided argument, append with the existing data, and continue. If the password field is not found, then it will show all the name:value for manual input. and then store modified data in either dictionary or json format.

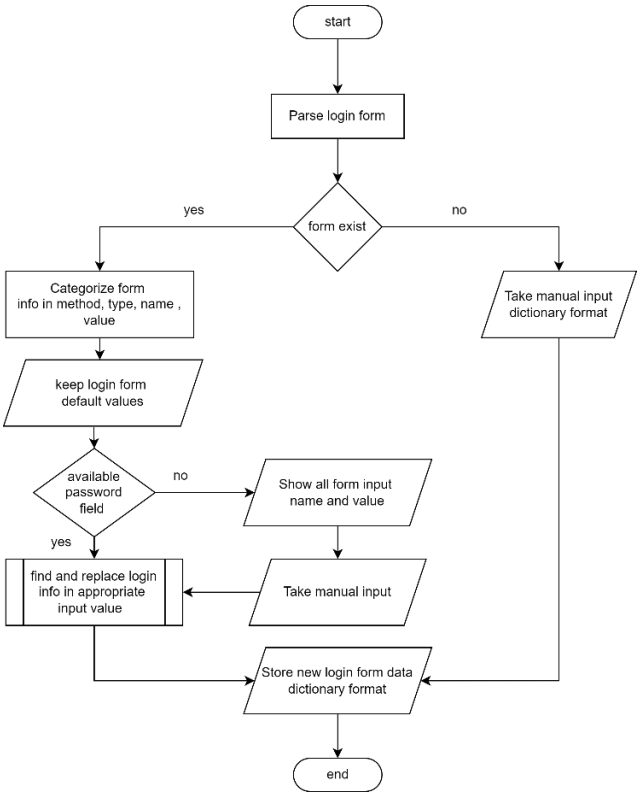


Figure 4.1 Gather and modify login form data

4.2 Attack Generation (Capture Session ID)

This module works to request the URL along with the credentials and user-agent. Capture the Session ID and store it. The Python requests module has been used for this. By using that, the request header can be modified to inject a custom user-agent, which will act like a different browser. For every request with a custom user agent, the Session ID, status code, and redirected location are captured for further action. During an exchange of the session from one user agent to another user agent, The captured Session ID and location will be exchanged, and the capture will begin again from the response.

```
GET /sf2/sflogin.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost/sf2/sfhome.php
Connection: close
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Cache-Control: max-age=0
```

Request Header

```
HTTP/1.1 200 OK
Date: Thu, 18 Aug 2022 18:09:58 GMT
Server: Apache/2.4.54 (Debian)
Set-Cookie: PHPSESSID=frclpgrj96hc3h9dr0griq83gn; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 690
Connection: close
Content-Type: text/html; charset=UTF-8
```

Response header with Session ID

```
HTTP/1.1 302 Found
Date: Thu, 18 Aug 2022 18:10:18 GMT
Server: Apache/2.4.54 (Debian)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: sfhome.php
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8

```

Response header with status code

4.3 Response analyser module (Comparing Session ID)

This module will compare the stored Session ID with the response header. In this step, the captured Session IDs from the attack generation module will be compared with the sent Session IDs. So, this can prove that Session IDs are similar or not. Some web applications validate the requested Session ID and don't issue new Session IDs. That means the requested Session ID has been accepted. Again, some web applications validate the requested Session ID and issue the same Session ID with a set-cookie parameter. Some web applications take the requested Session ID and invalidate it, then issue a new Session ID.

So, by comparing the requested Session ID and the responded Session ID, it can be helped to understand whether the web application is authenticating or not. The cookie parameter in the request header is used to send the request session, and the Set-cookie parameter in the response header is used to send the response Session ID. Here the Session ID is taken as an example, PHPSESSID.

Responded with same Session ID as Requested	
Cookie	PHPSESSID=17sp4be1klku1p9b35qaeuihfq
Set-Cookie	PHPSESSID=17sp4be1klku1p9b35qaeuihfq

Responded with No Session ID	
Cookie	PHPSESSID=17sp4be1klku1p9b35qaeuihfq
	No new Session ID

Responded with New Session ID	
Cookie	PHPSESSID=17sp4be1klku1p9b35qaeuihfq
Set-Cookie	PHPSESSID=crphsse1s1cogvbra308j417ut

So, this module determines whether unauthenticated session & authenticated session same or not, whether two browsers have the same session or not, and whether exchanging sessions between two different user agents is acceptable or not.

4.4 Report generating

From the analysis of the response analyser module, this module generated the result. This module will show if the web application is vulnerable to session fixation or not. From which it can be identified what the session management problem is. It will show how much time it takes to do the whole process.

CHAPTER 5
RESULTS

In this chapter, we will compare and contrast our suggested model with previous models that detect Session Fixation Vulnerability. In particular, in this chapter, we will demonstrate why our suggested model outperforms others in terms of performance. This section compares our methodology to two previously proposed methods (Lukanta, 2014) and another detection model (Nadar, 2016). We chose two earlier techniques to compare with our model because they both proposed methodologies based on Session Fixation Vulnerability.

5.1 Results

In this part, we analyse the strategies we presented to detect session fixation vulnerabilities. We have some well-known websites here. There are several approaches available, however they are not based on user agent testing. Previous detection models only work on HTTP websites. However, ours works on both http and https. We discovered that our approach is fast and accurate in outcomes by researching existing models and comparing them to our model. Some test results from our proposed model are shown in the table below.

	Test	Results	False Positive	False Negative
Facebook	Done	Secure		
Netflix	Done	Secure		
Twitter	Done	Secure		
Dailymotion	Done	Secure		
Vulndb	Done	Vulnerable		
zedge	Done	Secure		
wapka	Done	Vulnerable		
Rokomari	Done	Secure		
Awardspace	Done	Vulnerable		
Sundorban Courier	Done	Secure		
Artisian	Done	Secure		
Freepik	Done	Secure		
WackoPicle	Done	Vulnerable		
Standard Chattered	Done	Secure		

Table 5.1: Session Fixation Vulnerability Test Results

As you can see in Table 5.1 Our method is fully accurate. According to those web application, we don’t have any false-positive or false-negative results.

Using our methodology, we run some tests in various web applications. That web application is open to the public. We tested them at random and discovered that certain online applications utilize user agents to protect their users from attacks, while others do not. When we ran our test on them, we discovered the results shown in the table below.

Web Application	Test	Result	True Positive	False Positive	True Negative	False Negative
google.com	OK	Secure			*	
facebook.com	OK	Secure			*	
twitter.com	OK	Secure			*	
wapka.org	OK	Vulnerable	*			
wish.com	OK	Secure			*	
aliexpress.com	OK	Secure			*	
tomshardware.com	OK	Secure			*	
awardspace.com	OK	Vulnerable	*			
indeed.com	OK	Secure			*	
github.com	OK	Secure			*	
toscrape.com	OK	Secure			*	
askubuntu.com	OK	Secure			*	
stackoverflow.com	OK	Secure			*	
vuldb.com	OK	Vulnerable	*			
gsmarena.com	OK	Secure			*	
flixbus.com	OK	Secure			*	
eprice.it	OK	Secure				*
brilio.net	OK	Secure			*	
starnow.com.au	OK	Secure			*	
9gag.com	OK	Secure			*	

Table 5.2: Session Fixation Vulnerability Test Results

From The Previous Table 5.2 Now we will show you a chart of Session Vulnerability in figure 5.2. This chart represents the result of our proposed model. Its proven how effective our method is.

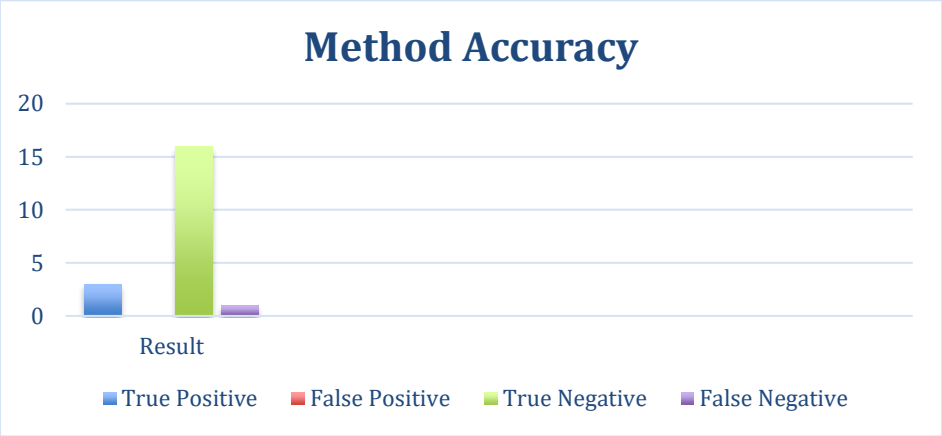


Figure 5.2: Methods Accuracy

Our proposed methodology is clearly successful, as evidenced by the chart in figure 5.2. Where we have a very low false positive and false negative ratio. Our chart shows that our methods' false positive is null and false negative is one of 20 well-known online applications. According to our findings, our technology outperforms the prior methods.

Now we will show our Methods performance by Pie Chart in the below. The pie chart will show our methods success rate from the previous chart in figure 5.2

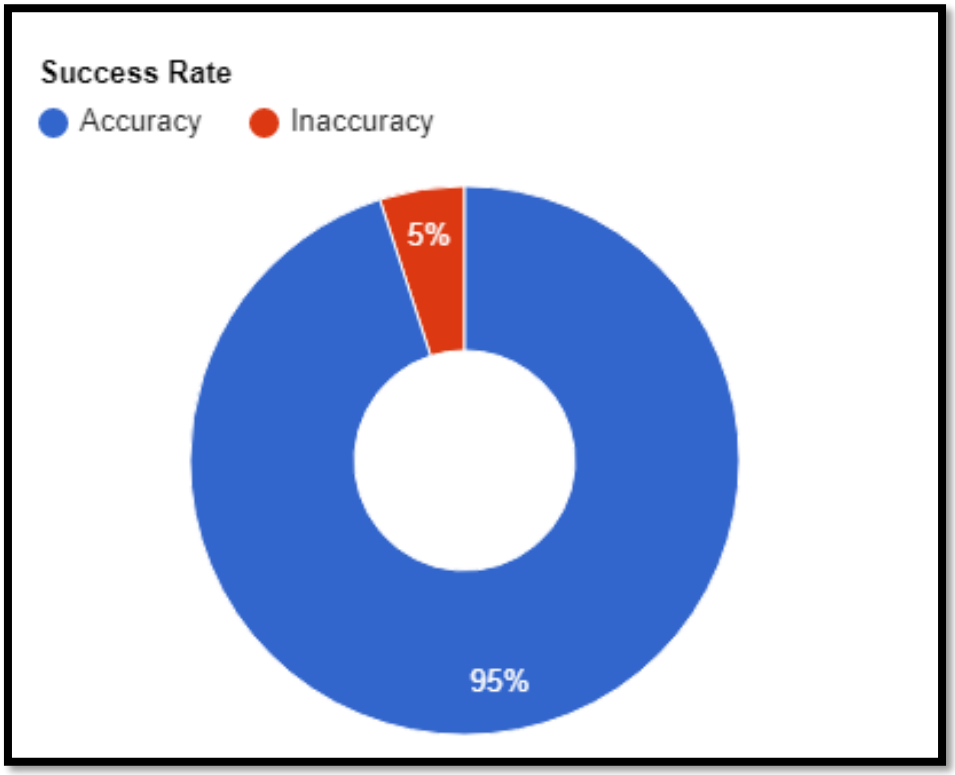


Figure 5.3: Methods Success Rate

According to our pie chart, our proposed methodology has a 95% success rate and a 5% error rate. We test only 20 web applications for those results because we are unable to test only one web application. We are successful with our automatic detector because we can test 19 web applications out of 20.

The websites examined are live and vulnerable. Finding using our method may assist the developer to fix the vulnerability by finding them.

CHAPTER 6
CONCLUSIONS AND LIMITATIONS

The primary goal of this research was to develop an automated session fixation vulnerability detector. By allowing session fixation attacks to perform an operation in a web application on behalf of a user without their knowledge, this type of vulnerability could result in the loss of millions of sensitive data. This attack can be damaging for both the business and the user. This form of vulnerability has received insufficient attention and has resulted in some research. The researchers advocate using a user agent approach to prevent session management difficulties.

The contribution of the paper is a fully automated method. This methodology requires only one user identification before producing a report. Human agents are not required in the proposed method. Automatically detecting session fixation vulnerabilities is really fast.

Only our recommended approaches can detect this issue automatically. We are unable to mitigate this critical session management flaw. We are concerned about various identity and authentication issues in addition to session fixation.

REFERENCES

Amira, A., Ouadjaout, A., Derhab, A., & Badache, N. (2017, March). Sound and static analysis of session fixation vulnerabilities in php web applications. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy* (pp. 139-141)

Bortz, A., Barth, A., & Czeskis, A. (2011). Origin cookies: Session integrity for web applications. *Web 2.0 Security and Privacy (W2SP)*

Bugliesi, M., Calzavara, S., Focardi, R., Khan, W., & Tempesta, M. (2014, July). Provably sound browser-based enforcement of web session integrity. In *2014 IEEE 27th Computer Security Foundations Symposium* (pp. 366-380). IEEE

Dande Alekya, D., & Mishra, P. (2021). Study on Manual Auditing for Web Application Vulnerability Detection. *Annals of the Romanian Society for Cell Biology*, 19612-19618.

Johns, M., Braun, B., Schrank, M., & Posegga, J. (2011, March). Reliable protection against session fixation attacks. In *Proceedings of the 2011 ACM Symposium on Applied Computing* (pp. 1531-1537).

https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

https://cwe.mitre.org/data/definitions/384.html

Kumar, R., & Goel, A. K. (2014, September). Automated session fixation vulnerability detection in web applications using the set-cookie HTTP response header in cookies. In *Proceedings of the 7th International Conference on Security of Information and Networks* (pp. 351-354).

Lukanta, R., Asnar, Y., & Kistijantoro, A. I. (2014, November). A vulnerability scanning tool for session management vulnerabilities. In *2014 International conference on data and software engineering (ICODSE)* (pp. 1-6). IEEE.

Nadar, V. M., Chatterjee, M., & Jacob, L. (2018). A defensive approach for CSRF and broken authentication and session management attack. In *Ambient Communications and Computer Systems* (pp. 577-588). Springer, Singapore.

Nadar, V., Chatterjee, M., & Jacob, L., (2016). Detection Model for CSRF and Broken Authentication and Session Management Attack. In *International Journal of Computer Science and Information Technologies*

Nikiforakis, N., Meert, W., Younan, Y., Johns, M., & Joosen, W. (2011, February). SessionShield: Lightweight protection against session hijacking. In *International Symposium on Engineering Secure Software and Systems* (pp. 87-100). Springer, Berlin, Heidelberg.

Ryck, P. D., Nikiforakis, N., Desmet, L., Piessens, F., & Joosen, W. (2012, June). Serene: Self-reliant client-side protection against session fixation. In *IFIP International Conference on Distributed Applications and Interoperable Systems* (pp. 59-72). Springer, Berlin, Heidelberg.

Schrank, M., Braun, B., Johns, M., & Posegga, J. (2010). Session fixation–the forgotten vulnerability? *Sicherheit 2010. Sicherheit, Schutz und Zuverlässigkeit*.

Takamatsu, Y., Kosuga, Y., & Kono, K. (2012, July). Automated detection of session management vulnerabilities in web applications. In *2012 Tenth Annual International Conference on Privacy, Security and Trust* (pp. 112-119). IEEE.

Tang, S., Dautenhahn, N., & King, S. T. (2011, October). Fortifying web-based applications automatically. In *Proceedings of the 18th ACM conference on Computer and communications security* (pp. 615-626).

