Security Vulnerabilities Originated from Dynamic JavaScripts in Web Applications of Bangladesh: An Empirical Study

SYEDA TASNIM FABIHA
BSSE 0622

A Thesis
Submitted to the Bachelor of Science in Software Engineering Program Office
of the Institute of Information Technology, University of Dhaka
in Partial Fulfillment of the
Requirements for the Degree

BACHELOR OF SCIENCE IN SOFTWARE ENGINEERING

Institute of Information Technology
University of Dhaka
DHAKA, BANGLADESH

Security Vulnerabilities Originated from Dynamic JavaScripts in Web Applications of Bangladesh: An Empirical Study

SYEDA TASNIM FABIHA

Approved:

Signature *Date*

Supervisor: Nadia Nahar

Lecturer

Institute of Information Technology

University of Dhaka

To *Mahmuda Begum*, my mother

whose endless inspiration has always kept me motivated

Abstract

Ensuring the security of web application has become very important in recent days. Modern web applications deal with millions of sensitive information and users' specific data. If these information are compromised, it can result in significant damages. Therefore, developers should always be aware of leaving no vulnerability on the developed web application. Moreover, the use of scripting language is seen on a large scale in the modern web applications. To enhance the performance and provide the users a great ease, scripting languages are used on regular basis. Using dynamic scripts also provides better performance to the web application. However, this may lead to an obscure vulnerability, which is named as XSSI (Cross-Site Script Inclusion). To detect this kind of vulnerability in the context of Bangladesh along with the identification of dynamic script usage on Bangladeshi websites is a research challenge.

An empirical study approach mixed with both automated and manual analysis is proposed here which will help to identify the true condition of Bangladesh in this security aspect. The study is consisted of six different steps. The first step is to select the target dataset which satisfies the study criteria. After that manual registration is performed on each of those sites. For the next step, a chrome browser extension is implemented along with a backend server. These automated system helps to identify and collect dynamic scripts on target dataset. The collected scripts are analyzed thoroughly to find excitability. Finally countermeasures to avoid such vulnerabilities are discussed.

The final results of the study shows that there is no exploitable scripts in the target dataset. The number of found dynamic scripts are also very low on target dataset. After attempting to find out the reason behind this, it is found that, the extended use of JavaScript language is hardly seen in the target dataset. Most of the websites tend to use direct HTML embedded scripts rather than including those from external links. JavaScript provides various upgraded features and and facilities which are hardly be used in the target website. Thus, this study successfully accomplishes the main goal of finding the prevalence of dynamic scripts along with the true condition of Bangladesh in this security aspect.

Acknowledgments

I would like to appreciate my supervisor Nadia Nahar, and Dr. Kazi Muheymin-Us-Sakib, for their supportive guidance, continuous inspiration and candid advice. Their support and help have inspired me to relentlessly work for producing the best quality work I can throughout the project. I would also like to thank my classmates of BSSE 6th Batch for their continuous motivation throughout the project.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> It is privacy, which is not for the passive
>
> *Jeffrey Rosen*

Web security is a very important term in web programming analysis. Ensuring security of web applications has become the main concern to all, as websites now-a-days deal with millions of users' sensitive data. The focus of this research is to investigate a new security aspect of web application, which has not been given importance so far. This security aspect is related to the vulnerability occurring from scripts, used in the web applications. This chapter demonstrates the issues of this vulnerability detection task and introduces the research challenges out of these. It also briefly describes the contribution and achievement of this research. Finally, the organization of this thesis is indicated for giving a reading guideline to the readers.

1.1 Introduction

Vulnerability means a security exposure which results from weakness of any system. Security vulnerabilities mainly allow an attacker to minimize the system's information assurance policy, which are system's integrity, availability or confidentiality [1]. Security of web based applications, containing sensitive data such as financial, medical or personal data, has become more and more important in recent days. If any of those sensitive data is compromised, it can result in significant damages [2]. Traditional network-level fortifications do not offer necessary defense from attacks, as most of the vulnerabilities are deeply embedded on the application logic. Moreover, developers mostly look behind the security flaws of the web application inadvertently. They use dynamically generated scripts to enhance the user experience without realizing that, it helps an attacker controlled web page to get the scripts [3]. Although, there are some policies such as browser's Same-Origin-Policy to protect websites from attack [4], use of dynamically generated scripts can easily violate those. The included scripts can contain user specific data as global objects, making those accessible by attacker controlled web pages. This type of vulnerability, introduced not many years ago, is known as Cross-Site Script Inclusion (XSSI). Despite the presence of this type of attack in literature [5], the popularity of this kind of flaw has not been studied on real-world websites since 2015 [3]. More analysis and study on the real-life web sites can highlight the severity of this kind of vulnerability, and developers may act in accordance.

XSSI is not the only a vulnerability generated from dynamically generated JavaScript, it can also lead Cross-Site Scripting (XSS) vulnerability or Cross-Site Request Forgery (CSRF) vulnerability [6] [7]. Moreover, there is no such analysis that was conducted on Bangladeshi websites. Existing vulnerabilities of Bangladeshi govt. websites are still undetected and unsolved. Therefore, possible vulnerabilities of these websites holding sensitive data should be analyzed and detected, helping developers avoiding these in future.

1.2 Issues in Existing Vulnerability Detection Approaches

Studies that have been conducted on different dataset about analyzing the vulnerabilities of web based applications, can be divided into three groups. The first group studies based on static analysis. The second one focused on dynamically generated scripts. Finally, the third group mostly focuses on other types of vulnerabilities such as DOM based analysis, etc. In the static analysis group, WebSSARI was introduced as a tool that conducted a type based analysis for PHP [8]. It used a simple tainting analysis to find cases if the user given input flows into sensitive functions. The algorithm used in WebSSARI, was based on static types, and it did not specifically cover the dynamic features in scripts. A further static analysis was conducted that detects vulnerabilities such as SQL injections, cross-site scripting and HTTP splitting attacks only on Java applications [9]. A tool was proposed by this research work which was implemented on top of Eclipse. That tool allows developer to perform security check quickly during the period of development. Another study that was also based on static analysis, was effective for PHP just like WebSSARI [10]. However, none of these studies cover the threat that may be occurred from the dynamically generated scripts. It also maybe added that, no Bangladeshi websites were ever undertaken such types of analysis.

The second group of studies focused on scripts that are generated dynamically on web pages. A newly evolved vulnerability, that is not given enough priorities till today, is known as XSSI [11]. Several other authors also used the term on later period. Nevertheless, it was only for referring to minimal variations of the attack [5] [12]. However, the prevalence of the flaw has never been analyzed on real world websites until 2015, which was an experiment regarding the threat XSSI [3]. This research work showed different ways of leaking sensitive data via dynamically generated scripts. It also stated that many dynamic scripts are not properly protected against XSSI attacks. Although, it selected Alexa top 150 websites as the dataset, it can be assumed that, analyzed result would not be the same for different dataset. Moreover, that study covers very few details about other possible vulnerabilities from dynamically generated JavaScript. Nevertheless, Bangladeshi govt. websites can be taken as a different dataset for highlighting the existing vulnerability conditions of these websites.

The last group of work presented an experimental study on using four commercial vulnerability scanners. It helped to detect security vulnerabilities in 300 publicly available real-life web services [13]. The four tested widely used commercial scanners including two different versions of a specific brand were HP WebInspect [14], IBM Rational AppScan [15], and Acunetix Web Vulnerability Scanner [16]. This study showed that confirmation on most web services are deployed without undergoing any proper security testing. Although, the study was focused on the behavior of different vulnerability scanner, types of vulnerabilities were not the main concern. On later period researchers focused on the DOM based XSS vulnerabilities [17]. On that study, a fully automated system was presented to detect and validate DOM-based XSS vulnerabilities. However, focusing only the DOM based vulnerabilities is not enough as dynamically generated script related flaws are becoming more popular. Only two studies were conducted considering the web applications vulnerabilities in Bangladesh, whether focusing on only XSS and CSRF [18], or on only SQLi [19]. Both research works was done by black box testing, and whole process was done manually. If automation of whole investigation is not possible, at least data collection process for analysis phase should be automated to deduct result processing time.

1.3 Research Questions

The existing researches on web application vulnerabilities focus on either static types or other vulnerabilities that are not generated from dynamically generated scripts.

The shortcomings of these individual approaches have been discussed above. Only one research work focused on XSSI, a vulnerability produced from dynamically generated scripts. That vulnerability was dependent on user's session state. Still, it was not enough to provide sufficient data for other similar kind of flaws. Moreover, all Bangladeshi websites are remained unchecked for any of these flaws. More analysis should be done on different dataset to cover all possible ways to exploit a website. Thus, this leads to following primary research question.

- *How common dynamic script generation is in websites of Bangladesh and how often these scripts are dependent on a user's session state?*

A framework will be presented here which will automatically check state-dependent dynamic script, and collect data about the impact of user's session state. A manual analysis will be conducted thereafter, to determine whether these scripts include any user sensitive data. This will help to accurately identify all the security aspects. Potential countermeasures will also be analyzed. These can be achieved through answering following sub-questions-

— How to develop a framework to detect the dynamically generated scripts from any websites?

What will be the main tasks of the framework? How to conduct the tasks? A browser extension can be developed to store all the data required for the analysis. The initial step is to collect all the external script resources included by the web pages. This can be done by using the so-called Mutation Observer [18]. Afterward, the extension will request the script files twice: once with authentication credentials attached to the request, and once without authentication credentials. After the response will have been received, both responses will be compared. If these differ from one another, the data will be stored for later analysis. Therefore, detecting the state-dependent dynamic scripts, will be the main task of the framework.

— How to detect the impact of user's session state on all the gathered scripts? What will be the prerequisites?

As mentioned above, the extension will send request twice. Since, the request should be sent along with the valid credentials, valid session cookies are a necessary precondition. the user will need to manually log in to the website under investigation beforehand for obtaining valid session cookies. 3. How the impacts will be analyzed? A manual analysis will be conducted with all the gathered data. In such way, it will be precisely determined which scripts have a dynamic nature, depending on user's session state in place of randomness. Finally, all the security aspects will be investigated depending on the exploitability, and potential countermeasures will also be analyzed. Thus, the overall impacts on the websites will be highlighted.

1.4 Contribution and Achievement

This research incorporates both automated and manual analysis for detecting information leakage vulnerability due to the presence of dynamic scripts. An empirical study is performed here, where Bangladeshi dataset is selected for investigation. The whole approach is divided by six separate steps. Each steps are considered as the foundation of next step. These steps include, (1) target website selection, (2) manual registration, (3) dynamic script identification, (4) investigation of security aspects and exploitability, (5) discussion of potential countermeasures, and finally, (6) discussion of the prevalence of dynamic scripts on target dataset.

All the steps mentioned above, are performed successfully to derive the final results. At the third step, the automated framework is used, which is consisted of a chrome browser extension and a backend server. The extension works on all the target dataset and collectively sends gathered data to the backend server for later analysis. The backend server performs another operation on the dataset based on data received by the extension. Finally, the number of dynamic scripts are calculated and identified. This numbers indicates the prevalence of the usage of dynamic scripts on Bangladeshi websites. Each of the scripts are then thoroughly examined whether there is a possibility of information leakage through those. At the final steps, the overall results are discussed on the context of selected dataset.

The proposed automated framework including the extension and the server is implemented in JavaScript and C# programming language respectively. .Net framework is used to implement both the extension and the server. The chrome browser extension is developed by following Chrome Developer's Guide [20]. The analysis results prove the effectiveness of the proposed framework. This approach is evaluated based on the condition that, if this framework can detect all existing dynamic scripts of the corresponding investigated website or not.

A case study has been conducted to assess the proposed approach. A demo php based website is developed to evaluate the performance of the approach. All the steps are performed on the demo project and outcome of each step is described in depth. The identified dynamic scripts on the demo project are analyzed in depth to find a way of exploitability. Finally, the website is exploited and information leakage is performed based on the dynamic scripts. This procedure helps to increase a clear idea about the whole approach.

1.5 Organization of the Thesis

This section gives an overview of the remaining chapters of this thesis. The chapters are organized as follows –

- **Chapter 2:** Some preliminaries of web application vulnerabilities are discussed in this chapter. The description of the main reasons behind all the security issues are also added along with the basic concept of some popular vulnerabilities.

- **Chapter 3:** To the best of author's knowledge, no existing literature incorporates any study, which is focused on XSSI and conducted on Bangladeshi dataset. This chapter shows the existing research works in the vulnerability detection and exploitation on vulnerable web applications.

- **Chapter 4:** The architecture of the proposed approach is briefly demonstrated in this chapter.

- **Chapter 5:** The implementation details of the automated framework, which is used as the base of the study is described in this chapter. Additionally, results found on each steps are shown in details for the better understandability about the conditions of Bangladeshi websites.

- **Chapter 6:** A case study on a sample php based web project is shown here for the assessment of the proposed approach.

- **Chapter 7:** It is the concluding chapter which contains a discussion about the framework and some future directions.

Chapter 2

Background Study

> Torture the data, and it will confess to anything
>
> *Ronald Coase*

Web applications security is very important as these deal with millions of user-sensitive data. Ensuring the security of these applications is very difficult in recent days. Researchers have done extensive study in this sector just to make web applications more secure. However, the root causes behind all these vulnerabilities hardly get any focus. Most of these vulnerabilities are mainly generated from complex client end web applications. With the growing complexity of client-side programming, development of web applications is also becoming complicated. Specially, use of dynamic JavaScript has become a very common scenario in the advanced web applications.

Dynamic scripts in web applications help to enhance the user experience. However, such practise also creates the opportunity for the attackers to gain user sensitive information, since sensitive information might be presented in those scripts. Although, there are polices like browser's *Same-Origin Policy* (SOP), to prevent information leakage via cross-origin [4], these can be violated easily. Such violations can occur by including contents from other origin in script tag. Attackers can leak private user-sensitive data by scrutinizing the execution behaviour of the included dynamic scripts. This incident may lead severe consequences [3].

This chapter briefly describes the above mentioned facts and the consequences of such information leakage in form of popular web security attacks.

2.1 Root Cause behind Major Vulnerabilities

In this section the author tried to identify the root cause behind all the major vulnerabilities. Modern web applications have observed drastic transformation on various aspects. Now-a-days the use of the second generation web (Web 2.0) applications is more frequently observed rather than the traditional ones. These applications focus on customized user information, which is assimilated with the help of dynamic JavaScript. These changes are considered as welcomed features since these are introduced to enhance the quality, user-interactivity and better performance-ability of the web applications. These features are hardly considered as the main aspects of any kind of security vulnerabilities. However, these are the facts that are identified as the root cause behind major vulnerabilities. Out of those facts following two are considered as major amongst all other.

— **Transformation of the traditional websites into second generation (Web 2.0)**

— **Exemption of browser's *Same-Origin Policy***.

Therefore, these facts are needed to be observed closely before attempting to prevent any kind of exploitation.

2.1.1 Web 2.0

> *"Although the term "Web 2.0" does not have a rigorous definition, it is commonly used in at least two ways. First, it refers to Web applications that encourage social interaction or collective contribution for a common good. Second, it refers to Web programming techniques that lead to a rich and user-friendly interface."* [21] —Brian Chess in 2007

In simple words, the second generation of web applications is the improved version of the initial world wide web. This improved version is categorized by the transformation web applications from static to dynamic. The transformation also includes user-generated contents along with the growth of social networking. The first version was considered as mostly read-only web, whereas the second version of it, is considered as widely read-write web [22].

Although, this transformation has enhanced the interface by making it more user-friendly, it has also opened the opportunity for some dangerous vulnerabilities. These vulnerabilities can lead into severe security damages. Modern web applications are now developed by gathering properties from numerous of independent web applications. This is also a popular feature of Web 2.0, known as mash-ups [23]. This facility makes many web applications vulnerable in a sense that, the ill-intended attackers try to gain sensitive information from other web applications and use it for their own evil interest.

2.1.2 Same-Origin Policy

The *Same-Origin Policy* (SOP) is the primary security policy of Web browsers. The SOP strongly prevents browsers communicating directly with a different origin [4]. More specifically, SOP permits a given content access only to resources that have the same origin. The origin is specified as the port number, protocol and the host name of the involved resources. This means, if a content under one origin, requests for accessing the contents from another origin, the request will be failed. Thus, web pages are unable to access user's sensitive information of a different origin. However, the concept does not hold true for cross-domain inclusion of web content using HTML tag. Using this process, the HTML script tag can request for an external script resources by its `src` attribute.

In the recent era, client-side programming has been made enriched by using various scripting languages (i.e. JavaScript) for better performance. Modern web applications frequently use inclusion of third-party scripts for information fetch purposes. While browser's *Same-Origin Policy* prevents requesting information from other origin, including remote script via HTML script tag, make it possible to exempt from the policy. This is in fact, a very common and well-established mechanism. This mechanism is popularly used in the web for various valid purposes such as advertising purpose or traffic analysis purpose. However, this facility creates opportunity for malicious attackers to gain sensitive information from other origins. Thus, user sensitive data loses its confidentiality and integrity.

2.2 Security Attacks

In this section, major security vulnerabilities, which are generated from the previously mentioned facts, are discussed.

Since, server side responsibility has been reduced to a great extent, developers just try to get data from server through many API. That is where, intruders try to take the advantage of leaking information. The emerged vulnerabilities are related to user information security, rather than other malicious attacks such as *SQL injection* , which may hamper the web application server or total application architecture. Therefore, the discussed web application vulnerabilities are mainly JavaScript related. The key concern here is the security violations of critical user sensitive data, occurring from JavaScript inclusion. Thus, this lead the focus to three severe web application vulnerabilities, which are: *Cross-Site Scripting* (XSS), *Cross-Site Request Forgery* (CSRF), and *Cross-Site Script Inclusion* (XSSI).

2.2.1 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is a type of attacks where, malicious scripts are injected into trusted web applications [6]. To enable successfully this attack, firstly, attacker needs to discover that the target website contains a vulnerability, which enables script injection. Subsequently, attacker injects a malicious script, that becomes active at each visit of any valid users towards that vulnerable website. Thus, confidentiality of information can be compromised by the action of that malicious script. This attack can take place on any web applications, that allows user input without any validation or encoding methodology.

The whole mechanism is shown in Figure 2.1.

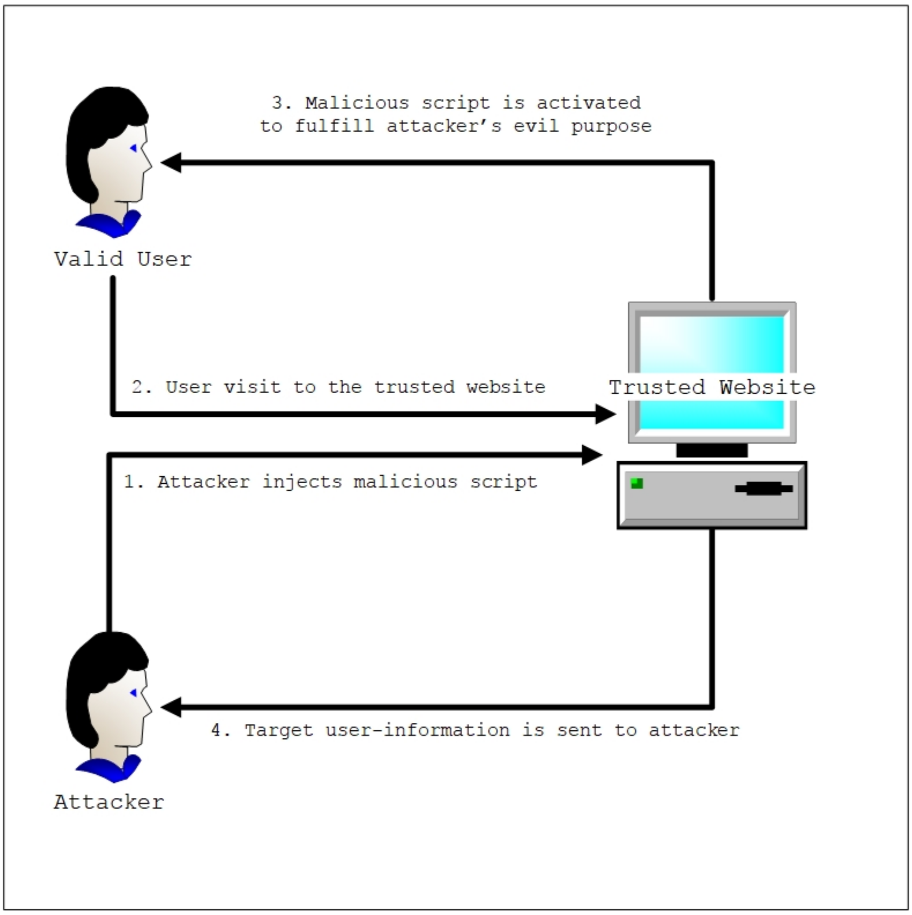

3. Malicious script is activated
to fulfill attacker's evil purpose
Valid User
2. User visit to the trusted website
Trusted Website
1. Attacker injects malicious script
4. Target user-information is sent to attacker
Attacker

Figure 2.1: Cross-Site Scripting Attack Technique

XSS is used to send a malicious script to a legitimate user. It is impossible for the browser of end user to detect that the script holds malicious contents. Hence, it will execute the malicious script, as it is considered to come from any trusted source. Thus, the malicious script can get the access to any sensitive user specific data like session tokens, browser cookies or even any other information given to that trusted site. The contents of HTML page can also be rewritten by these malicious scripts.

2.2.2 Cross-Site Request Forgery (CSRF)

The attack technique that forges the end users to conduct an unwanted action on a web application, where the users are currently authenticated, is named as Cross-Site Request Forgery Attack (CSRF) [7].

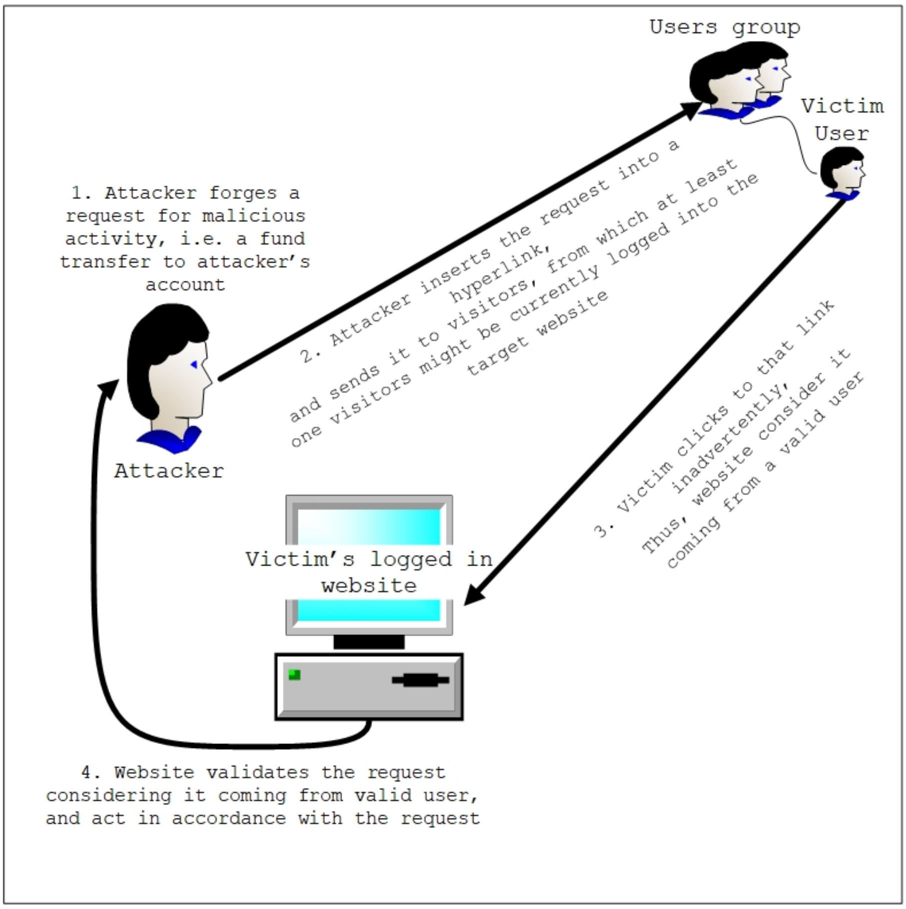

Users group
Victim User
1. Attacker forges a request for malicious activity, i.e. a fund transfer to attacker's account
Attacker
2. Attacker inserts the request into a hyperlink, and sends it to visitors, from which at least one visitors might be currently logged into the target website
3. Victim clicks to that link inadvertently, Thus, website consider it coming from a valid user
Victim's logged in website
4. Website validates the request considering it coming from valid user, and act in accordance with the request

Figure 2.2: Cross-Site Request Forgery Attack Technique

Since the attacker has no way to observe the forged request, data theft doesn't take place in this attack technique. Thus, CSRF attack is conducted based on only state-changing requests.

Figure 2.2 shows the basic techniques for obtaining this attack. Attacker can lure a victim of a web application for executing the attacker intended actions with the help of various social engineering, such as using email or chat for sending the link containing forged request. Only those, who are currently authenticated to the target website falls under the trap that the attacker conspired. If the victim is a normal user, then only state-changing actions can be performed by a successful attack. These state-changing actions may contain changing victim's password or email address, transferring funds and so on, so forth. If the victim holds some kind of super privileges from the target website, entire web application can be compromised by this attack.

2.2.3 Cross-Site Script Inclusion (XSSI)

> *"**XSSI**- A Fameless but Widespread Web Vulnerability Class"* [24] —
> Veit Hailperin in 2016

The web applications, that hold sensitive data in form of script or other contents, are mainly vulnerable to this attack. This kind of vulnerability is not very popular, that is why V. Hailperin termed it as fameless but yet widespread [24]. Fameless in a sense that, it has very few presences in the literature. It is also not present in any famous books on major security vulnerabilities. M. Zalewski first maintained this term in a very short manner [5]. Furthermore, it is not even present in OWASP listed security vulnerabilities [25]. Therefore, it is often assumed that this vulnerability is not something to worry about. Nevertheless, it is also widespread as a study showed that one-third of the surveyed data utilized dynamic script, 80% of which were exploitable by remote script inclusion [3]. An incident regarding the exploitation of this vulnerability was also reported on an web application. That vulnerable web application used authorization from a popular social networking site[26]. Thus, exploitation of this vulnerable web application lead to leak user-sensitive data provided on that popular social networking site. This incident points to the fact that, websites can be easily exploited by this attack, and personal information leakage of user-sensitive data may result in severe security issue.

This attack technique can easily be interpreted from its name. This is an attack which is conducted by inclusion of script via cross origin. The basic technique for successfully obtaining this attack is shown in Figure 2.3.

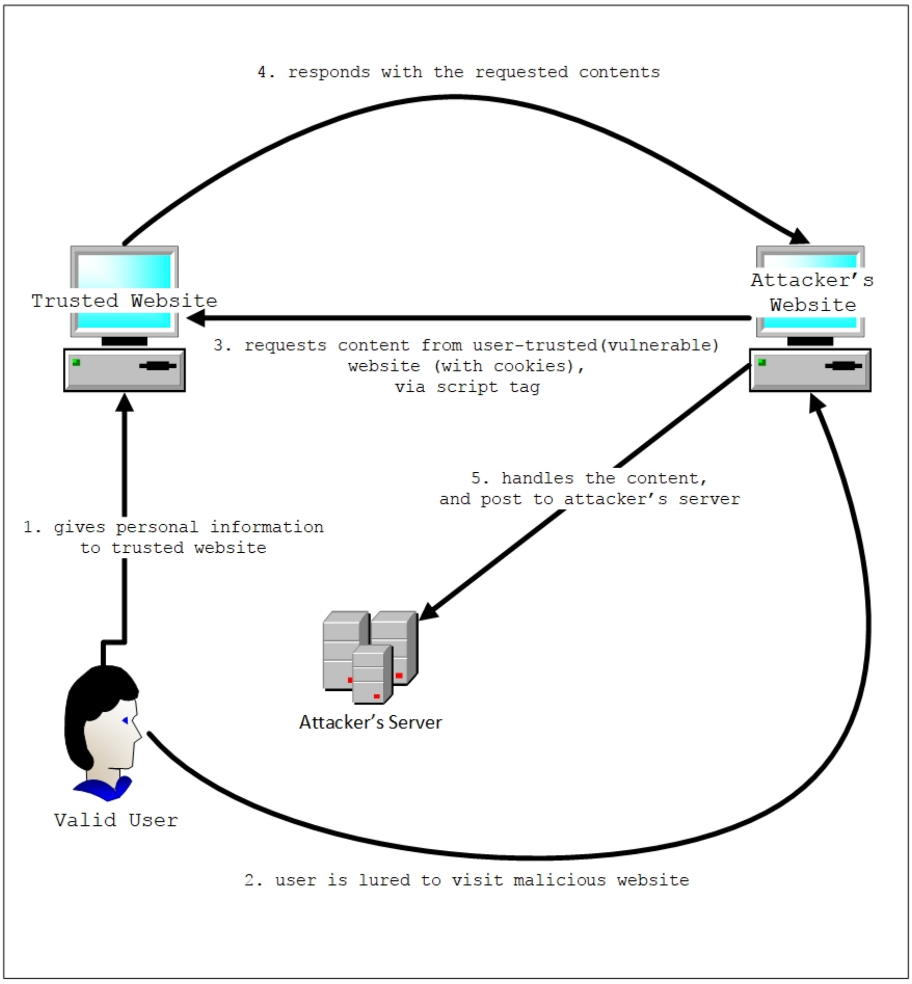

4. responds with the requested contents
Trusted Website
Attacker's Website
3. requests content from user-trusted(vulnerable) website (with cookies), via script tag
5. handles the content, and post to attacker's server
1. gives personal information to trusted website
Attacker's Server
Valid User
2. user is lured to visit malicious website

Figure 2.3: Cross-Site Script Inclusion Attack Technique

As mentioned earlier, SOP is relaxed for script inclusion, which means, one can easily request the contents from other origin via HTML script tag, pointing those to the `src` attribute. Thus, if any information is passed through that content, it can easily be tricked to post on any attackers' website. Therefore, any contents holding sensitive information directly falls under wrong hand, and thus the attack *Cross-Site Script Inclusion* takes place [27]. The included contents could be in any data format such as, JSON data, CSV file or script from other origin [28]. All the attacker needs to do is just handling the data in a way to gain the user-specified information.

XSSI Categories

Based on the presence of this attack technique in the literature, XSSI attack can be categorized in three basic ways. These ways can be termed as following:

1. Dynamic JavaScript based on authentication
2. JSONP
3. Browser's vulnerability

1. Dynamic JavaScript based on authentication: The first way is utilizing the presence of dynamic JavaScript [3]. Websites that create dynamic JavaScript, can be vulnerable to XSSI. These dynamic script files can be seen in two formats. JavaScript files that are reachable only via authentication token, fall under first category. This means after authentication, a web page may generate some new sets of JavaScript files which were not available before. Whereas, the other format of dynamic JavaScript can lie in the same script files. These script files were also present in the web page before authentication, but the contents inside those are not same depending on the authentication. If any attacker includes these dynamic scripts, several situations can be created by which data of those scripts will be accessible to the attacker. Those situations include overriding global variables, redefining global APIs, overriding global functions, altering the object prototypes, etc [3]. Therefore, an attacker can request for dynamic scripts, observe the results of execution and trick to steal sensitive data containing those scripts.

2. JSONP: Few years back data leakage via cross-origin was possible by including any JSON data in the script tag [21]. JSON data could also be interpreted by overriding the `array` constructors. However, this issue has been fixed. Now, attackers cannot gain any information by Including JSON data from other origin in the script tag. Nevertheless, today JSON data can be obtained by a technique named JSONP. JSONP is nothing but JSON with padding [29]. Therefore, attackers can easily request for JSON data from other websites by simply using JSONP. This leads to the same scheme all over again, that is leaking information from JSON file. Consequently, XSSI attack can also be executed by using JSONP.

3. Browser's vulnerability: XSSI attack can also be obtained by several browser quirks [12], which is not related to any scripts of other origins. For instance, attackers can request JSON data in `src` and specify the `charset` attribute. Hence, vulnerable browser will show an error with an encoded message. This message can be decoded with the specified `charset`, thus obtaining the original data of that JSON file. It is also possible to leak data, stored in some other file format, such as CSV, by exploiting the browser vulnerabilities. Therefore, XSSI attack can take place through utilizing these types of browser vulnerabilities, and gain user sensitive data in accordance.

2.3 Distinctions of XSSI towards XSS and CSRF

XSSI is often considered same as CSRF or XSS. XSSI is close to XSS by name, whereas it is close to CSRF by the illustration. However, the only similarity among these three attack techniques is, these are all client-side attacks [24]. Thereby, how XSSI differs from these two attacks, is described below.

2.3.1 Disseverance from XSS

XSSI is often misinterpreted as XSS. Subsequently, the terms by which these two attacks are known, are very similar to one another. However, the basic difference between the two is very simple. XSS takes place when malicious script is injected to a victim's web page, whereas, the code from victim's web page is included into a malicious page for triggering the event of XSSI [24]. Thereby, these two attack techniques are totally different in terms of the attack description.

XSS attack mainly works by injecting client-side script into the output fields of any web applications. This attack technique utilizes DOM manipulation to create various malicious activities such as altering form values, changing interface, etc. Sensitive data can also be posted to the attacker's server by switching the form action of any web applications. However, all these malicious activities are conducted with the help of injected malicious scripts to any websites.

Unlike XSS, XSSI attack doesn't include any malicious script into the targeted web applications. This attack technique is all about requesting the script contents from vulnerable web applications. By observing the execution behavior of the requested scripts, attackers can gain information presented in those scripts. Thus, XSSI is entirely related to information leakage. Consequently, this attack is not related to manipulating the vulnerable websites just like XSS, or other malicious activities mentioned above by injecting any scripts. Therefore, XSS and XSSI are two completely different attack techniques with the exception of sharing closely related names.

2.3.2 Disseverance from CSRF

As mentioned above, CSRF seems to be similar with XSSI by description. Both attack techniques include sending requests from a malicious page to a different origin, and in both cases, requests are executed in the context of the logged in users. However, the consequences of these attacks and the main goals are completely different.

XSSI is solely related to sensitive information leakage through cross-domain script inclusion. Albeit, this particular attack's consequences can lead into larger class of web attacks in the context of authenticated web user [3]. This means, a successful XSSI exploitation can lead attackers to execute other types of vulnerabilities such as CSRF or XSS related to user- authentication.

XSSI also differs from CSRF in terms of the core goal of these attacks. The main goal of CSRF is to causing state-changing actions in the name of valid users by generating request. Its goal is not related to data stealing. Therefore, contents from a response to CSRF request are not possible to read by any means. As opposed to this, XSSI does not focus to do any state-changing actions in the name of any logged in users, rather stealing information is the main target of it. These stolen data might be further perpetrated for accumulating other attacks.

2.4 Summary

Web application security is a vast domain, where numerous of security issues are continuously being discovered. These issues are also consequently being analyzed to discover necessary steps for avoidance. Nevertheless, the root cause behind all these issues are often avoided. Modern web applications have observed dramatic change towards static to dynamic. Some severe security vulnerabilities (i.e. CSRF) have also emerged with this new transformation. An attempt to investigate some of those serious security issues (i.e. XSSI) and root cause behind those, have been taken in this chapter. This discussion can help to understand the existing works related to these security issues.

Chapter 3

Literature Survey

In the literature, there are numerous of research works regarding web application vulnerabilities. Most of these focus on finding different case studies where web applications are vulnerable. Thus, those studies proposed some code of conducts, which is helpful for avoiding such vulnerabilities. Some researchers proposed frameworks or tools for making web applications less vulnerable. Based on types of works, web application vulnerability analyses and the proposed approaches can be divided into five groups. The first group of research was based on static analysis. The second group focused on DOM-based application vulnerabilities. The third one focused on vulnerabilities generated form violating the *Same-Origin Policy* (SOP). Studied based on Bangladeshi dataset contain in the fourth group. Finally, the fifth group mostly focused on other types of vulnerabilities, such as DOM-based analysis, etc. In this report, these literature contributions are discussed maintaining chronological order in each group. These categorized proposed approaches can be termed as follows:

— Static Analysis

— DOM-based Vulnerability Analysis

— SOP Violation Based Vulnerability Analysis

— Bangladeshi Dataset Based Analysis

— Other Types of Vulnerability Analysis

3.1 Static Analysis

Yao-Wen et al. [8] identified the secure information flow problem as the main problem of web application vulnerabilities. In this research work, a static analysis algorithm based on type-system and type-state, was proposed. The study targeted three major web application vulnerabilities. These were: Cross-Site Scripting, SQL Injection and General Script Injection. This research also introduced a tool named WebSSARI (Web Application Security by Static Analysis and Runtime Inspection). WebSSARI conducted a type based analysis for PHP, and after that, section of code considered vulnerable were instrumented with runtime guards. 230 open-source web application projects on SourceForge.net were also tested under the proposed algorithm for further analysis. In that analysis, 69 of those web applications were found vulnerable and developers were also notified about the vulnerabilities. In accordance with the notification, 38 projects acknowledged the findings and provided with the plans for patches.

As it was considered that, vulnerabilities in web applications were considered primarily to be connected with insecure information flow, the research work focused on assuring proper information flow. For achieving that, all the data from input fields were marked as tainted, and propagation of those throughout the application were perceived to check whether these could reach security critical function (the sink). Thus, WebSSARI kept track of variables' type-state, whether these were tainted or untainted. By applying this static analysis, WebSSARI helped to pinpoint code where runtime checks were needed. Furthermore, for data purification purpose, which was to make untainted the tainted, data was processed by a sanitization routine. Different sanitization functions were automatically injected before vulnerable functions were called. Depending on the vulnerable function, WebSSARI inserted three types of sanitization routine. These were, HTML output sanitization, database command sanitization and finally system command sanitization. Thus, WebSSARI provided runtime protection to the target web applications. The overall process was done by maintaining three files, which were predefined in the algorithm. In one file, there was preconditions to all sensitive functions (the sink). In the second one, there were all the known sanitization functions for untainted data. Finally, the last file was consisting of specifications of all the possible sources of untrusted input. Thus, the research work proposed this approach of static analysis and runtime protection in absence of user intervention.

WebSSARI used a simple tainting analysis to find cases, whether the user given input flows into sensitive functions. The algorithm used in WebSSARI, was based on static types. Nevertheless, it did not specifically cover the dynamic features in scripts. Therefore, almost all the possible vulnerabilities that might be generated from dynamic features of the scripting languages, were skipped. In modern web applications, use of dynamically generated scripts is very common. These applications face newer types of vulnerabilities, which are impossible to identify by only doing static analysis based on the information flow of type-state variables.

In the later period Livshits et al. [9] also proposed a static analysis algorithm for Java based web applications. These research work focused on finding three major web application vulnerabilities, which were: SQL injections, Cross-Site Scripting, and HTTP Splitting attacks. This paper proposed a tool based on a static analysis for detecting vulnerabilities resulted from unchecked user input. 29 security vulnerabilities, in 9 famous open-source Java web application, were found by this study. In the proposed system, users were able to provide specifications of vulnerabilities, which were automatically transmitted to static analyzers. Afterwards, the algorithm kept track of all the vulnerabilities which matched with the given specification in the statically analyzed code.

A tool was proposed by this research work which was implemented on top of Eclipse. That tool allows developer to perform security check quickly during the period of development. The first step to achieve this, was to observe the tainted object propagation. To do so, the frameworks required source description, sink description and finally derivation description of each object of the target application. The framework also required user specification of the vulnerabilities, which were translated into Java byte code, to find all the potential matches via static analysis. The result of the analysis was integrated within Eclipse, a popular Java development environment. By using the Eclipse plugin, user could easily navigate the object involving any vulnerabilities. Thus, potential vulnerabilities were made easy to observe and possible to fix as part of the development process.

The approach proposed in this study was mostly similar to WebSSARI. Like WebSSARI, this approach also focused on secure information flow. The proposed approach would help developers avoiding to develop Java web application in such a way so that, no possibilities of tainted data input making the whole application vulnerable. However, this approach does not cover the fact that many more types of vulnerabilities may occur in the web application in the runtime. This study covers the vulnerabilities that may hamper the system architecture, hence leaving the possibilities of user's sensitive data leakage related vulnerabilities.

Another static analysis effective for PHP language, was proposed by Yichen et al. [10]. The research work proposed a three-tier architecture for capturing information at the granularity level in decreasing manner. The three levels were intra-block level, intra-procedural level and finally inter-procedural level. This three-tier architecture were able to handle dynamic features of scripting language, which was not introduced by the previous literature. The significance of the approach was justified by applying this on 6 popular open-source PHP based web applications. It is stated that, 105 new security vulnerabilities were from those web applications, all of which were assumed to be remotely exploitable.

A bug detection tool was introduced by which serious vulnerabilities were automatically found with high degree of confidence. The tool aims at only detecting *SQL injection* related web application vulnerabilities. The algorithm took PHP source code as input, then parse it into Abstract Syntax Trees (ASTs). The analysis performed a standard conversion from the AST of the function body into a control flow graph, for each functions in the program. The nodes of the CFG were considered as basic blocks, and the jump relationships between two blocks were identified by the edges. Later block summary was driven where the set of variables that must be sanitized before entering block were identified. All the block summaries were combined into a function summary. The function summary included pre and post conditions for entering that function. Afterwards sanitization mechanism took place via function call. The sanitization includes many process such as casting to safe types, regular expression matching, etc. for making the untrusted input untainted. Thus the overall analysis was conducted and vulnerabilities detection were ensured.

This research work focused on PHP based web application vulnerabilities. The alarming fact that inclusion of any kind of scripts in any web application can be responsible for cross-domain information leakage, was not considered in this research. Moreover, with the increasing popularity of using third party scripts in web applications, the number of sensitive information related vulnerabilities are also increasing. Violation of browsers *Same-Origin Policy* (SOP) has introduced many other kind of web application vulnerabilities such as cross-origin information leakage, analysis on which have got less presence in the literature.

3.2 DOM-based Vulnerability Analysis

In recent year, client side functionality has enriched because of remarkable increase in complexity in JavaScript code. Thus, a proportional increase in client-side vulnerabilities with DOM-based *Cross-Site Scripting* has been occurred. Therefore, researchers have focused on the DOM-based vulnerabilities in the present situation.

Sebastian et al. [17] proposed a fully automated system which detected and validated DOM-based XSS vulnerabilities. This approach was consisted of a taint-aware JavaScript engine along with exploitation generation technique. A large-scale analysis was conducted on the Alexa top 5000 domains by utilizing the proposed approach. 6167 unique vulnerabilities were identified, which were distributed over 480 domains. This indicated that 9.6% of the total web applications hold at least one DOM-based XSS vulnerability.

The proposed system was divided into two separate components. The first component includes a modified browsing engine for dynamic taint-tracking of the tainted data flows, which was implemented for vulnerability detection. The second component was the automated vulnerability validation algorithm, that holds the information provided by the implemented browsing engine. For implementing the first component, open-source chromium browser was chosen. The JavaScript engine V8 and the DOM implementation in web-kit of the browser were patched in a way so that they were capable of conducting taint-tracking mechanism. Thus the information flow was detected and it was validated whether vulnerable or not by the automated algorithm. The system generated attack payload to validate the information flow. Finally, a chrome extension was built to communicate with the server and the application interface, which as able to automatically crawl many web applications in a limited time.

The implemented patch and the overall automated system of the study, were not later publicly available, that is why it was not possible for the developers to take advantage of the overall research work. Moreover, this research focused on DOM-based vulnerabilities, severe vulnerabilities that might be originated from dynamic script generation was not the main concern. Since dynamic script generation are very common in today's web applications, an increase number of vulnerabilities derived from those, are becoming very alarming. Research works should be conducted to put some light in this serious vulnerability issue.

Following to the previous research work regarding DOM-based vulnerabilities, Ben et al. [30] proposed an automated filter approach for DOM-based XSS, which consisted of runtime taint tracking system. The system also followed a taint-aware mechanism by which it was able to stop parsing of attacker controlled malicious content. This research work conducted an analysis before implementing the automated system. The analysis was based on the current conditions of the browser-based XSS filtering process, through which, a set of conceptual flaws were identified that allowed filter bypass in case of DOM-based XSS.

This study presented an in-depth analysis on the current approaches of the auditors for client-side XSS filtering. Mechanisms of several auditors such as, NoScript Firefox Plugin, XSS auditor, etc. It was mentioned that, most of those auditors were string-based, and limitations of string-based filters were described in depth. For the validation of the results, the mechanisms of previously discussed approach [25 million] were followed. Utilizing all the observed shortcomings, an alternative approach for client-side prevention of DOM-based XSS, were proposed. Character-level taint tracking was used in place of string-based tracking. The system was also able to precisely track the flow of malicious data into the parser through combining a taint-enhance browsing engine along with taint-aware HTML and JavaScript parsers. Thus, the system was able to precisely detect and stop injected code while parsing time.

This research presented an automated system to provide client-side protection against DOM-based XSS. However, focusing only the DOM based vulnerabilities is not enough as dynamically generated script related flaws are becoming more popular. Moreover, it is observed in the literature that popular web applications are prevalently vulnerable to several kinds of vulnerabilities, whereas, very few works have been done based on the conditions of Bangladeshi web applications, which should be increased for ensuring the security of all the users.

3.3 SOP Violation Based Vulnerability Analysis

As stated earlier, violation of the *Same-Origin Policy* can help to leak data across cross origin. Utilizing this mechanism, an exploitation technique was initially presented by Grossman [31]. The author discovered a vulnerability in Gmail, where it was possible to compromise someone's email contact list. This attack was named as *JSON Hijacking*. In this attack, a cross-domain script inclusion pointing a JSON data, was utilized. By using JavaScript's object constructor, it was possible to obtain the content of user's Gmail address book. The issue was reported to Google, and it was fixed immediately. However, the main target of this study was to highlight the exploit technique and the precautions needed to avoid such type of attacks. Although, the issue was fixed by Google, it introduced a new attack technique, which was first publicly termed as *Cross-Site Script Inclusion* (XSSI) by Christoph Kern [11]. Several other authors also used the term on later period [5] [12].

A generalized approach influenced by Grossman's technique, was introduced by Chess et al. [21], and it was termed as *JavaScript Hijacking*. In that study, 12 popular Ajax frameworks, including 4 server-integrated toolkits and 8 purely client-side libraries were analyzed. The defend mechanism against *JavaScript Hijacking* and recommendations for developers were also provided in that study.

The attack technique discussed in this research, used script-tags to include JSON objects of other origins. Thus, it was possible to gain all the JSON data by overriding the constructor to create all the objects of JSON data, and further posting it to attacker's web server via XMLHttpRequest. Web applications that make use of JavaScript as data transport mechanism via ajax call, were considered vulnerable according to this research. Nevertheless, this usage is common in the second generation of web applications (Web 2.0). For Web 2.0 applications that handle confidential data, two fundamental ways to defend against *JavaScript Hijacking* were proposed. Declining malicious requests and preventing direct execution of the

JavaScript response, were mentioned as the best way defending against *JavaScript Hijacking.* Finally, an analysis was conducted on 12 Ajax frameworks and 8 client-side libraries, out of which, only one framework was found to be preventive from this attack.

The attack technique described in this research work was mainly based on the usage of script-tag in combination with a non-standard JavaScript quirk. However, the non-standard JavaScript quirk has been removed from all the major browsers in the meantime. Nevertheless, this introduced the fact that data leakage can be possible across origin via some other way. Data leakage from JSON object was only the main focus in this study. Later it was discovered that other types of data object can also be leaked by following the similar technique. XSSI was one of those, on which a few analysis is visible in the literature.

Takeshi Terada [12] introduced five new attack techniques and browser vulnerabilities based on cross-origin script inclusion (or XSSI). These variations of XSSI attack were named as *Identifier-based XSSI.* The main idea of this paper was to develop understanding of the *Identifier-based XSSI* attack techniques and relevant countermeasures.

The five new attack techniques or browser vulnerabilities found in that research are described below:

a. Simple IE's bug leaks runtime errors: In case of error occurrence in the loaded JavaScript file of different origin, browsers normally present only fixed JavaScript error messages such as “Script error”. Thus, browsers prevent cross-origin information leakage via JavaScript error messages. However, IE9 and IE10 did not behave the same, instead, these provided the detailed error messages. By utilizing this vulnerability, the researcher was able to gain information about CSV data from different origin. However, the issue was fixed on later period.

b. JSON and other data theft using UTF-16: In the above bug, only data of limited formats, such as CSV, could be exploitable. A trick using UTF-16 encoding was discovered to expand the exploitable types of data. The trick was to use charset specification while requesting JSON data from another origin via script tag. Thus, external JSON file was loaded successfully in UTF-16, and it was possible in many browsers including IE9. This attack might not possible in IE10 or later, as those refuse to regard scripts as encoded in UTF-16.

c. Harmony proxy bug in Firefox / Chrome: Harmony is a new version of ECMAScript [32], which introduces some new language features such as JavaScript Proxy [33]. Harmony proxy could be used for cross-origin information on both in Firefox and Chrome. In response to the report on this issue, Chrome disabled JavaScript proxy feature [34], while fix from Firefox was not released yet.

d. Exhaustive search: In this paper, three brute force techniques were described that can sophisticate the attack. First technique was binary search, by which data of N bits could be identified by N times of data loading trials. The second technique was the utilization of JavaScript getter. Finally, the third one was the use of VBScript for JSON array theft on IE. All these exhaustive search techniques do not require any browser vulnerabilities. Thus, these attack techniques work on all major browsers except the third one, which works only on IE.

e. CSV with quotations theft: A trick was introduced which enables successful data theft when attacker controls some parts of the CSV file. Attacker can successfully reach the goal by taking advantage of escaping method in CSV. Doubling quotations character is defined as the escaping method in CSV. This attack works on all the major browsers.

This research work revealed five different ways of XSSI attack as well as the counter measures, that should be taken for avoiding such kind of attack. Setting appropriate content type flied in the response header was the primary counter measure. While browsers were not able to convey all the counter measures, additional measures were also suggested for the developers. These code of conducts would help avoiding such types of XSSI attack. The research work mainly focused on browser vulnerabilities. However, analysis on the severity of XSSI vulnerability containing real world web applications, was not visible in this literature.

Sebastian et al. [3] first highlighted the prevalence of the flaw by analyzing real world web applications regarding the threat XSSI. This research work showed different ways of leaking sensitive data via dynamically generated scripts. It also stated that many dynamic scripts are not properly protected against XSSI attacks. Furthermore, this study proposed protection approach that could be applicable for avoiding such kind of attacks.

This study provided an in-details analysis of dynamic script inclusion. The main purpose of this research was to identify the prevalence of dynamic scripts related web application vulnerabilities. Therefore, collecting all dynamic JavaScript from the tested websites was the first goal. A chrome browser extension was implemented to fulfill the task. The extension was used to collect all the scripts from tested sites, and then to detect dynamic code generation based on authentication credentials. The final step was the manual analysis of the gathered data, which identified the dynamic scripts depending on authentication. Out of 150 examined web applications, 81 utilized some form of dynamic script generation, and 49 of those were dynamic depending on user's session state. This indicated that, one-third of the investigated domains used state-dependent dynamic scripts. In the later findings, this research identified four types of potentially security critical data. Based on this critical information, those web applications were also exploited.

Although, this study selected Alexa top 150 websites as the dataset, it can be assumed that, analyzed result would not be the same for different datasets. Moreover, that study covers very few details about other possible vulnerabilities from dynamically generated JavaScript. However, no such analysis is conducted on any Bangladeshi web applications. Few developers are aware of the risk generated from dynamic JavaScript. Nevertheless, Bangladeshi govt. websites can be taken as a different dataset for finding the vulnerability existing in these websites.

3.4 Bangladeshi Dataset Based Analysis

Analysis on Bangladeshi web applications consisting of vulnerabilities has very few presences in the literature. Delwar et al. [19] presented an evaluation of existing **SQL injection** related vulnerabilities lying in web applications of .bd domain. The study was conducted by following black box penetration testing approach, which was done manually. Finally, the data collected were analyzed for providing guideline to website administrators.

This study claimed to be the first approach in Bangladesh that conducted analysis on .bd domains regarding **SQL injection** (SQLi) vulnerabilities. User input based **SQL injection** techniques were used to check vulnerabilities in 600 .bd domain web applications. 400 of those were found vulnerable to SQLi attack, out of which 340 were found vulnerable to **get based SQLi** and the rest (60) were vulnerable to **post based SQLi** attack. Most of these vulnerable web applications were based on various older versions of PHP language. The study suggested that vulnerabilities that were found, could be avoided by using proper user input authentication. Finally, this was proposed that, the future developers should maintain the best practice for avoiding such vulnerabilities before deployment.

This paper presented analysis of SQLi vulnerabilities existing in the web applications of .bd domain. Online web applications are recently getting very popular in Bangladesh. This study was the first that had been conducted manual on the existing vulnerabilities these web applications encounter. However, only manual analysis or black box testing is not enough to ensure web application vulnerabilities. More investigations should be done on other types of vulnerabilities such as cross-origin information leakage, since these are becoming very important in security issue of the users.

On the later period Tanjila et al. [18] proposed a similar analysis on Bangladeshi web applications. which was focused on XSS and CSRF related vulnerabilities. it was mentioned that XSS and CSRF were chosen because of those highest rank in OWASP list [25]. This research provided analysis on the data, which were collected during the experiment. The current condition of the experimented web applications along with a guideline for the future developers were also discussed in this work.

The research work followed black box testing approach, which was done manually. 500 Bangladeshi web applications were assessed for XSS and CSRF vulnerabilities. 335 out of these were found vulnerable to the targeted types of attacks. Web applications which were found vulnerable to XSS, 35% of those were vulnerable to store XSS, and 65% were vulnerable to both DOM based XSS and reflected XSS. Few websites were not vulnerable to XSS at first attempt, after imposing CSRF attack those became also vulnerable to XSS.

It was mentioned in the study that, the administrators of vulnerable web applications were communicated and very few of them reacted according to that. However, the exact number was not presented in this paper This study focused on only two types of vulnerability by black box testing which is done manually. However, only black box technique approach cannot ensure the detection of all vulnerabilities. Furthermore, it is very time consuming and impossible to recheck the false positive. If automation of whole process is not possible, at least data collection process for analysis phase should be automated to deduct the result processing time.

3.5 Other Types of Vulnerability Analysis

An experimental study on using four commercial vulnerability scanners were presented by Marco et al. [13]. The main idea of this paper was to evaluate the security vulnerabilities of web services with respect to the usage of different security scanners. The study was focused on database centric web applications, most of which were written in PHP. This research work identified security vulnerabilities in 300 publicly available web services. The main goal was to study effectiveness of security scanners and discover the common types of vulnerabilities in web services. The four tested widely used commercial scanners includes two different versions of *HP WebInspect* [14], *IBM Rational AppScan* [15], and *Acunetix Web Vulnerability Scanner* [16]. The experiment showed that a large number of web services are deployed without having minimal security testing. It was also observed that the effectiveness of the scanners in detecting vulnerabilities varies a lot.

The research followed two main approaches for testing web applications for vulnerabilities, which were, white box testing and black box testing. White box testing was consisted of the analysis of source code of web applications. This was done manually as well as by using some code analysis tool such as *FORTIFY*, *Ounce*, etc. The study also identified the fact that, exhaustive source code analysis was difficult, and all security flaws might not be detected through this. In the black box testing phase, execution of the application was observed in search for vulnerabilities. This approach was also named as penetration testing where the scanners were used to detect security flaws containing in the web services. The overall research work followed four basic steps, which were, **Preparation**, **Execution**, **Verification** and **Analysis**.

The selection of the commercial scanners and a large set of publicly available web services were included in the preparation phase. In the experiment phase, the scanners were used to scan the potential flaws in target web service. After that, manual testing was performed to ensure the vulnerabilities as part of the verification phase. This step helped to remove false positive. Finally, in the analysis phase, the obtained results were analyzed, and overall observation were systematized for future analysis. The main observation made by this research work was that, different scanners detect different types of flaws. This indicated that, selecting a suitable web scanner for web services is a very difficult task. Another observation was the high rate of false positive in the obtained results, which means, many vulnerabilities probably remain undetected. Finally, *SQL injection* was detected as prevalent in the web services, as it represented more than 84% of all vulnerabilities identified.

The study suggested future research to be focused on the development of an effective scanner for web services environments. Albeit, the study was mainly focused on the behavior of different types of vulnerability scanners, preventing web application development in vulnerable way, was not the main concern. Modern web applications are now more dependent on third party web services, which is somehow related with the use of scripting languages. Thus sensitive information leakage has now become the main concern in the web application security. This literature showed the major vulnerabilities lie in exploiting the database or harming the web application infrastructure. However, the fact was avoided that, sensitive information related vulnerabilities are also increasing in this era in a very alarming rate.

Fonseca et al. [35] presented an analysis which was conducted on the source code of the security patches of widely used web application written in weak and strong types language. An analysis of source code of the scripts that were used to attack those web applications, were also provided by this paper. Thus, this work helped to train developers as well as provided the foundation for further research on several security mechanisms such as, vulnerability scanners, static code analyzers, etc.

This research focused on *Cross-Site Scripting*, and *SQL Injection*, as these are very common and most critical vulnerabilities found in web applications. Both Source code written in many languages as well as the source code used to exploit those were also examined, so that the relevance of this kind of vulnerabilities could be understood. Firstly, source code of most commonly used weak typed language, PHP were analyzed. Subsequently, source code of string typed languages, which were Java, C# and VB were analyzed. For properly analyzing the patches, both previous version and the current version of application were considered, and the analysis was done manually. 715 vulnerabilities and 121 exploits of 17 web applications were analyzed with the help of field data on past security fixes. The proposed methodology helped to gather information on frequent mistakes that developers should stay away from. Thus. It contributed to the future developer training process for developing vulnerable free web applications.

This paper focused on the analysis based on existing field data. The derived observation was highlighted for future awareness. Nevertheless, many vulnerabilities related to dynamically generated scripting language are increasing in an alarming rate in recent time. Those vulnerabilities were not given priorities in this literature. However, more research work based on the prevalence of this type of vulnerabilities should be conducted, for ensuring the security of sensitive information provided by the user to the web applications.

3.6 Summary

It was mentioned earlier that there exist very few studies based on the vulnerabilities generated from dynamically generated script. Modern web applications have moved toward client-side operations, which lead an increase number of vulnerabilities regarding information leakage via cross-origin platform. However, no research work has been conducted on Bangladesh in this field to highlight the existence of sensitive information related vulnerabilities. It has been observed that, there are few research works regarding the vulnerabilities of Bangladeshi websites [19] [18]. However, these are not enough to fight against the recent severe situation. More empirical studies regarding different popular vulnerabilities on different websites, can help reducing the number of potential vulnerabilities in Bangladeshi websites.

Chapter 4

Methodology

An approach to detect the presence of dynamic JavaScript on any web applications and finding the exploitation technique is presented in this chapter. As stated in the previous chapters, use of dynamic scripts on any web applications can lead to extreme occurrences of information leakage. Cross-Site Script Inclusion (XSSI), is such kind of security attack, that can be easily done by utilizing the advantage of using state-dependent dynamic JavaScript on web applications. Studying existing security related literatures reveals that, this problem has not been identified as one of the major security crises yet. The prevalence of such issue once was addressed by studying only on several international web applications [3]. Therefore, an approach for Bangladeshi dataset is also required for providing acuity of the occurrences of dynamic scripts and, determining how these occurrences can lead to severe consequences of information leakage. Keeping the above factors in mind, a methodology is proposed to experiment an empirical study. This is designed to gain insights into the prevalence and exploitability of data leakage caused by the use of dynamic script generation.

4.1 Overview of the methodology

Conveying an empirical study on Bangladeshi web applications was the main goal of this research. As the foundation of the investigation target web-sites have been chosen and thoroughly examined. The overview of the total mechanism can be described by several steps. Figure 4.1 shows the flow diagram of each step. For obtaining valid authentication credentials, user needs to be authenticated to the application manually prior to the investigation. Therefore, for the sake of the study, all the websites were manually registered and were filled with personal information necessary for authentication.

An automated framework is developed which will identify the state-dependent dynamic scripts. State-dependent dynamic scripts are those, which are dynamic, depending on the authentication credentials. A chrome browser extension is built to serve the above mentioned purposes. A backend server is also developed to store the collected information by the chrome extension. This information contains the presence of dynamic scripts on the specified website. Finally, detected dynamic scripts were gathered for further analysis. This analysis includes exploitability and security of the corresponding website.

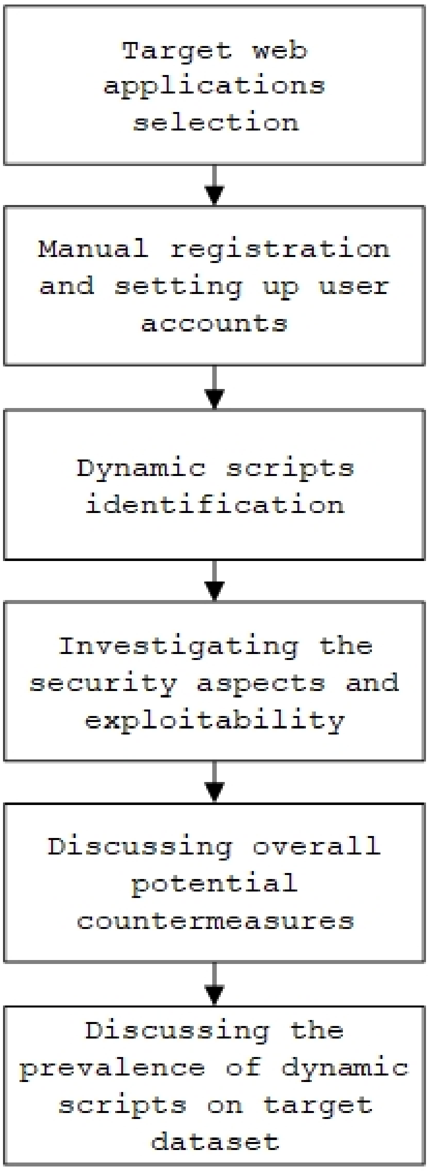

Target web applications selection
Manual registration and setting up user accounts
Dynamic scripts identification
Investigating the security aspects and exploitability
Discussing overall potential countermeasures
Discussing the prevalence of dynamic scripts on target dataset

Figure 4.1: Flow Diagram

4.1.1 Target web-sites selection

As stated earlier, detecting potential data leakage through dynamic-script generation on Bangladeshi websites is the main focus of this research. Alexa top ranked Bangladeshi websites are chosen for this purpose. An active user account is needed to each of the sites for checking whether dynamic scripts are generated depending on authentication cookies. To do so, the selected websites must fulfill the following criteria.

1. The target web application selected from"Alexa Top Sites in Bangladesh" list, must be Bangladeshi website. For example, google.com is not selected from those.

2. There must be an option to be registered for every user in the selected websites, and account registration along with login procedure must be freely available.

3. Selected websites must not be a duplicate or localized variant of an already investigated site (e.g. amazon.com vs amazon.in).

Ultimately, websites which follow the above mentioned criteria, are selected for further investigation.

4.1.2 Manual registration and setting up user accounts

An active user account or a similar state-full relationship is required for the detection of information leakage through dynamic-script generation. Therefore, all the tested websites are manually filled with personal information. Consequently, the size and the nature of dataset is kept limited. In this manual process, there are several difficulties and challenges. Most of the websites require verification through e-mail id or mobile number by sending activation link or one-time password. In some cases, verification code takes a lot of time to be sent from the server of target website. It may also happen, while authenticating on a website, the verification link sent from the server is not even accessible by any means. There may be an error sent from the server saying unsupported request. As a result, registration will not be possible in that website. Such incidents may reduce the number of target websites to a great extent.

Under the above circumstances, among Alexa top 50 websites, successful manual authentication was possible for 10 websites. Further investigation was based on the data collected from these websites.

4.1.3 Dynamic scripts identification

As the foundation of the empirical study, an automated framework is developed which will identify the state-dependent dynamic scripts. State-dependent dynamic scripts are those, which are dynamic, depending on the authentication credentials. Therefore, a chrome browser along with a backend server is implemented that fulfills two separate tasks. Figure 4.2 depicts the overall layout of the extension. The main tasks of the extension are described below.

1. Collecting scripts from target websites: Collecting all external script resources included by the target website, is the first step towards analyzing websites for dynamic scripts. The developed extension serves this purpose. Each time a new scripts node is found on the target website, the extension immediately passe it to the background server for later analysis.

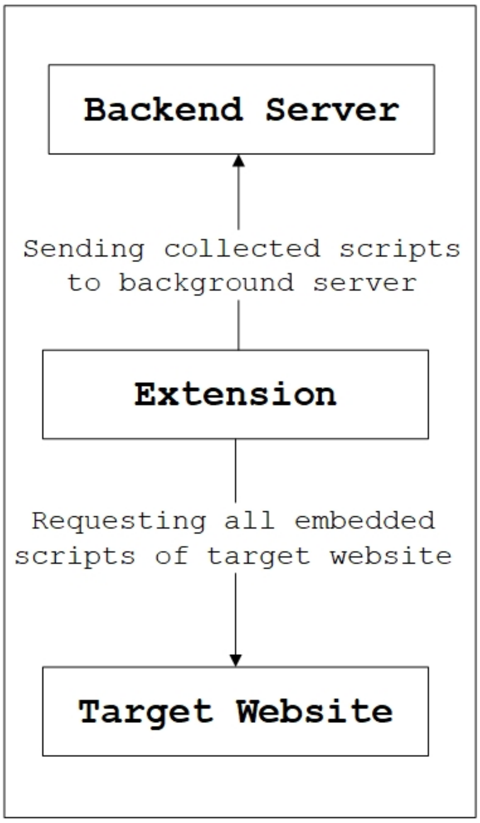

Backend Server
Sending collected scripts to background server
Extension
Requesting all embedded scripts of target website
Target Website

Figure 4.2: Extension Functionality

2. Detecting dynamic code generation on the collected data based on authentication credentials: Initially all the external script resources of target website are stored to the background server. On later period same resources are requested twice. Once with authentication cookies, and once without authentication cookies. Each time the response scripts are analyzed whether those differ. Those, which differ in between the two requests, are identified as the dynamic scripts and saved to the central database for further analysis.

Technical background of the extension and the server

For detecting dynamic scripts, both the developed extension and the backend server works together. The whole procedure is done in three separate module. The extension takes input from target website, process the data, and then send it to backend server. Figure 4.3 represents the basic architecture of the whole procedure.

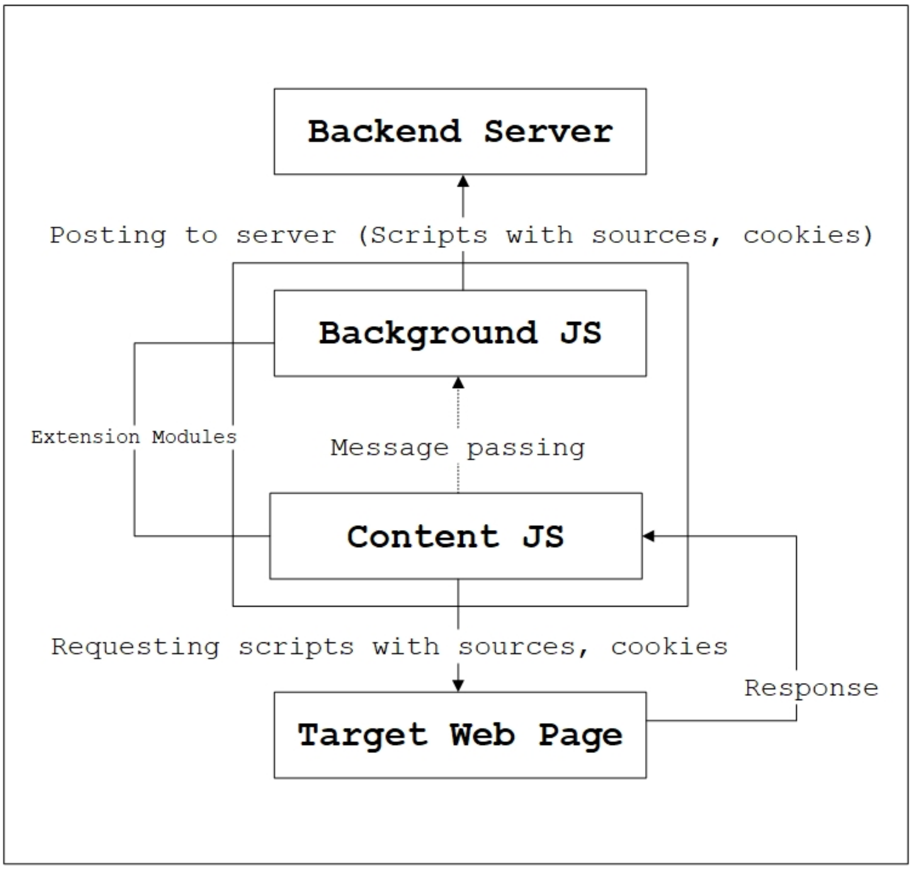


Figure 4.3: Extension's Basic Architecture

Extension's Responsibility: To perform the expected tasks, the extension needs to work in two separate modules. The core modules of the extension are, (1) background JavaScript module: this will work on background right after loading the extension to the browser, and (2) content JavaScript module: this will work on the context of current tab of the browser.

After finishing registration on the target website, the extension is loaded on that page. When the extension is loaded on the target web page, it's content script works on gathering all the external scripts sources of that page along with the cookies. Then this module uses popular `Message Passing` procedure of chrome extension and passes all the gathered data to the background module [36]. Background module receives all the data and finally post those to backend server.

It is necessary to request scripts from the target websites, regardless their transfer protocol, i.e, HTTP or HTTPS. If the scripts of an HTTPS website, are sent directly from the content module to the backend server, there will be an error saying "mixed content request" [37]. This error occurs due to the reason that, the backend server is developed on HTTP protocol,and browsers do not let share information between websites of mixed content (HTTPS and HTTP). Therefore, the total procedure is done into two phased module to avoid the complexity of mixed content request.

Backed Server's Responsibility: After the extension has completed it's responsibility, then the backend server performs to complete the whole dynamic script detection task. Figure 4.4 shows the server's task-flow after receiving the data sent by the extension.

Now that backend server holds all the necessary information i.e, cookies, scripts and sources, it attempts to the second phase of the dynamic script identification process. To detect dynamic code generation, the server first requests to the external script sources with cookies. Sent responses are then identified as `Response1` and saved to the database. The server again requests to those same script sources, but

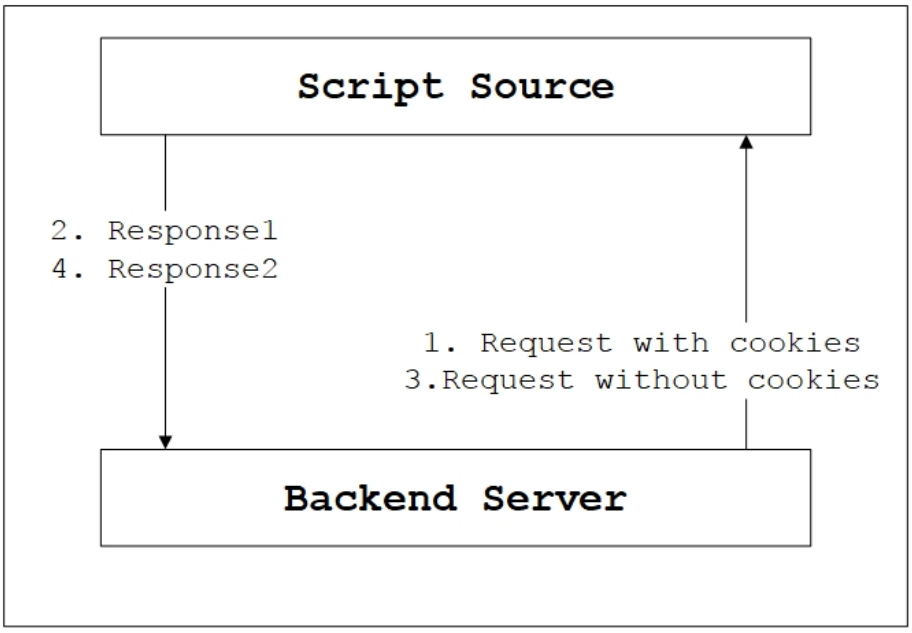


Figure 4.4: Extension's Basic Architecture

this time without cookies. The second responses are identified as `Response2`. In the second response there will be some scripts missing, which means, some scripts are not reachable without authentication cookies. These types of scripts are then directly identified as the dynamic scripts. Rest of the responses which returns data in both cases, which were identified as `Response1` and `Response2` (with authentication cookies and without authentication cookies), are then prepared for further detection procedure. Here scripts included from external sources, are the main focus of this study.

In the next step of dynamic script detection procedure, the two sets of responses are sent to the comparison module of the server. Figure 4.5 shows the activity of the server's comparison module. This module takes the two response sets. It then compare the whole block of the response scripts with the other one. If these two differ from one another, it means dynamic code has been injected based on the authentication credentials. The server then identifies that particular script sets as dynamic for further analysis. Rest scripts which remained same in both responses indicate that, those didn't change based on the authentication cookies. Thus, those

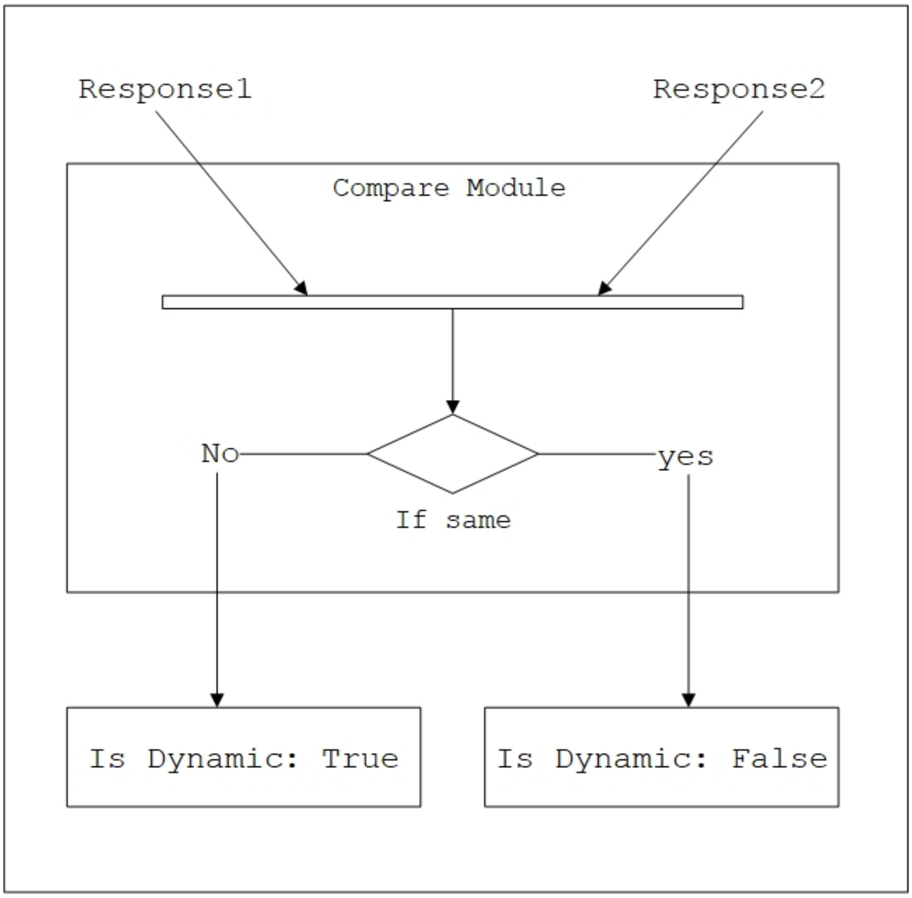


Figure 4.5: Server's Activity on Dynamic Script Detection

are identified as not dynamic and are not important for later analysis.

In this way, the total process of the detecting dynamic scripts on target website is done. After this phase of the methodology all the dynamic scripts are precisely identified and saved for later analysis.

4.1.4 Investigating the Security Aspects and Exploitability

The final step of the architecture is manual analysis of gathered data. This is done to precisely determine which scripts are dynamic in nature depending on the user's session state rather than randomness. All the gathered data are thoroughly examined in respect with the target websites. All the scripts, which are identified as dynamic, are manually analyzed whether there is a possibility of potential data leakage. If then the dynamic scripts are included to the attacker's website, attacker can manipulate those to leak data. To examine the exploitability, the developed background server is assumed to be the attacker's website and used to perform the data leakage procedure.

If the detected code uses some pattern or some way, so that information can be manipulated in a way to post to the background server (which is now serving the role of attacker's website), then websites holding those scripts will be exploited. There maybe different situations in which sensible data can be accessible to an attacker after the included script has been executed. Following are some example situations where dynamic scripts can play the role of potential data leakage.

The use of global variables in dynamic scripts

If any dynamic scripts use global variables, those variables can be accessible by any other script executed on the same web page. Therefore, whenever sensitive information is placed to any global variable inside a script, an attacker can gain access to that corresponding information.

```
//global variable used in vulnerable script
  window.value = "This is a secret value";
```

Listing 4.1: Example of dynamic script using global variables

To exploit such situation attacker needs to include the script and use that variable into the another script. With the completion of value assignment task, the attacker can view the information which was assigned to that global variables. Listing ****** shows an example of exploitation in this situation.

```
// to exploit this attacker can use following code on attacker's
    site,
     console.log(window.value);
```

Listing 4.2: Example of attacker's code to gain access to the global variable

Global API override

If any dynamic script uses any globally available API, and passes any sensitive information to such function, the function can easily be overridden by the attacker. Thus, information passes through the function can easily be gained by any attacker. Following is the example of such type of vulnerable script code, and attacker's code to exploit those. Following is a example dynamic script code, where global API can be used to do some specified task.

```
function myFunction() {
  var myVar = { secret: "secret value"};
  // Calling a predefined global function JSON.parse();
  return JSON.parse(myVar);
}
```

Listing 4.3: Example of dynamic script using global API

By observing the dynamic script code, attacker can find a way to gain access of that `myVar` object. Overriding the API is the simplest way, by which attacker can gain access to that object.

```
// to exploit this override the push function in attacker's site
JSON.parse = function(data){
    console.log(data);
}
```

Listing 4.4: Example of attacker's code to exploit dynamic Script that used global API

The use of variable prototyping in dynamic scripts

As JavaScript is a prototype-based language [38], this feature may enable attacker to gain access of the object which uses object inheritance. Object inheritance is known as prototype chaining in scripting language. Following is a example of JavaScript's prototype chaining. Here the object `secretValueList` uses `foreach` function. Thus, the function `foreach` inherits the object `secretValueList`.

```
//in vulnerable site the script contains
(function() {
  var secretValueList = ["secret1", "secret2", "secret3"];
  secretValueList.foreach(function(element)
  {
    // do something here
  });
})();
```

Listing 4.5: Example of dynamic script using prototype chain

If any attacker closely observe the source code of this dynamic script, can determine that the object `secretValueList` is an array object. Now, the attacker can use the prototype feature, and override the object by using prototype chaining.

```
//to exploit this, in attacker's site override this prototype chain
Array.prototype.forEach = function() {
  //here this = ["secret1", "secret2", "secret3"] because foreach
      function is overriden
  console.log(this);
};
```

Listing 4.6: Example of attacker's code to exploit dynamic script

Discussing overall potential countermeasures

Potential countermeasures to avoid such type of information leakage are also discussed on later analysis phase. A case study is also developed to make it more clear about how sensitive information can be stolen with minimum effort. Certain approaches which should be avoided by the developers, are also identified. Thus, the necessary countermeasures are identified, which will prevent information leakage through dynamic scripts.

Discussing the prevalence of dynamic scripts on target dataset

The main goal of the study is, how common it is to use dynamically generated scripts in Bangladeshi websites. After the implementation of the whole approach, results and findings will help to gather information about the true condition of Bangladesh. Therefore, the prevalence of dynamic scripts on Bangladeshi websites is described in the final phase of the study.

4.2 Summary

The architecture of the overall empirical study described in depth in this chapter. The core components of the architecture are explained one by one. The basic criteria and the filtering process of dataset selection are also covered. The automated architecture as well as the manual part of the study are described in depth. Moreover, the baseline of analyzing the exploitability, and the countermeasures for preventing such exploitation are discussed. The next chapter will cover results of this approach along with the comparison.

Chapter 5

Implementation and Result Analysis

This chapter aims to experimentally evaluate the performance of the proposed approach by applying it on the selected dataset. Each step described on the previous chapter is experimented with real websites. Moreover, results of each step is also described. This evaluation justifies the proposed approach, as well as helps to understand the true condition of Bangladesh in this vulnerability aspect. The overall approach is mixed with manual and automated process. An chrome browser extension along with a backend server is implemented to automate the experimentation process. Experiments were done to evaluate the performance of the extension along with the server. The server also renders the facility of an attacker's website. The total number of dynamic scripts found on each target website will highlight the main research goal. Moreover, manual analysis to exploit through all the dynamic scripts proves whether or not it is possible to steal personal information through the inclusion of dynamic JavaScript.

5.1 Implementation of Extension and Server

This section covers the implementation details of the developed chrome browser extension along with it's backend server. This tools are used in the third step of the proposed approach, as mentioned in the previous chapter. As mentioned earlier the extension is developed by using JavaScript language on top of .Net framework. On the other hand, the server is implemented by using C# programming language on top of .Net framework. In order to develop the extension following tools are used:

- **Microsoft Visual Studio Professional 2015 (version 14.0):** It is an IDE (Integrated Development Environment) developed by Microsoft Corporation [39]. It provides support for JavaScript, C#, HTML and CSS, as well as their modern successors. It is used to implement the proposed technique based on JavaScript and .Net Framework.

- **Chrome DevTools:** The Chrome DevTools are a set of web authoring and debugging tools built into Google Chrome [40]. To inspect the source code of the website under investigation, this tool is use to observe the overall performance of that site.

- **Microsoft SQL Server:** A database management system was also necessary to store information in this study. SQL server was used to develop the central database in this research. It is a database system developed by Microsoft [41].

The implementation was executed on an operating system with following configurations:

- **Processor:** Intel(R) Core(TM) i5 7200U CPU @2.50 GHz

- **RAM:** 8GB

- **Operating System:** Windows 10 Home

- **System Type:** 64-bit Operating System x64-based processor

5.2 Results of each phase

This section aims at experimenting all the steps described on the previous chapter on the selected dataset. The overall approach is evaluated with the results found and kept the data for later analysis. This analysis includes the decision making on the overall vulnerability aspects on Bangladeshi dataset.

5.2.1 Target web application selection

As mentioned in the previous chapter, the selected websites must satisfy some specific criteria. Alexa list of top sites in Bangladesh was selected primarily as the target dataset [42]. However, not all the websites in that list are eligible for this study. Therefore, a manual pre-selection procedure has been followed on the listed websites. Table 5.2.1 shows the filtering process only for the first 10 websites of that specified list. This process has been conducted on Alexa top 50 websites in Bangladesh, and total selection procedure is shown in Appendix A.

Domain Name	**Alexa Rank**	**Selection Criteria**			**Final Result**
		Bangladeshi	**Authentication**	**Duplicate**	
Google.com	1	No	Yes	No	Discarded
Youtube.com	2	No	Yes	No	Discarded
Facebook.com	3	No	Yes	No	Discarded
Prothom-alo.com	4	Yes	Yes	No	Selected
Google.com.bd	5	No	Yes	Yes	Discarded
Yahoo.com	6	No	Yes	No	Discarded
Kalerkantho.com	7	Yes	No	No	Discarded
Daraz.com.bd	8	Yes	Yes	No	Selected
Bd24live.com	9	Yes	No	No	Discarded
Wikipedia.org	10	No	No	No	Discarded

Table 5.1: Dataset Selection Procedure

The selected domains must full-fill three criteria, which are: (1) it must be Bangladeshi domain, (2) there must be an option of user authentication and it should be free, and lastly, (3) it must not be a duplicate of already selected domain. If any website of the list satisfies all these three criteria, then it is selected for the next phase.

5.2.2 Manual registration and setting-up user accounts

After completing the first phase, manual registration has been done on all the selected target websites. However, some websites such as Pothom-alo.com provided server error while registration. It took a long time to be authenticated to that site i.e, almost three days. Table 5.2 shows the final list of all the websites where manual registration is done successfully.

Selected Domain	Alexa Rank	Authentication
Prothom-alo.com	7	Successful
Daraz.com	14	Successful
Bdjobs.com	15	Successful
Grameenphone.com	20	Successful
Teletalk.com	21	Successful
Bioscopelive.com	26	Successful
Bikroy.com	27	Successful
Crazy-hd.com	31	Successful
Services.nidw.gov.bd	50	Successful
Foodies.com	not listed	Successful

Table 5.2: Final Dataset

The “Alexa Rank” column in the table indicates the ranking of the corresponding website on Bangladesh according to Alexa. It can be observed from the table that there is a selected website not listed on Alexa top list. This website is added to dataset to increase the validity of the final decision.

5.2.3 Dynamic Script Identification

After conducting the manual analysis over the dataset, and selecting the specific target dataset, The automated architecture is applied to the selected dataset. In this phase, all the target websites are loaded one-by-one, in the context of the developed chrome extension. The backend server is also kept running to save all the gathered information to database. After completing this procedure to all the target websites, the ultimate output is derived from the backend database. This output indicates the number of dynamic scripts found on each website. Table 5.3 shows total number of scripts and dynamic scripts based on authentication.

Selected Domain	Alexa Rank	Total Script	Dynamic Scripts
Prothom-alo.com	7	2	2
Daraz.com	14	23	1
Bdjobs.com	15	31	2
Grameenphone.com	20	7	0
Teletalk.com	21	1	0
Bioscopelive.com	26	20	0
Bikroy.com	27	8	0
Crazy-hd.com	31	24	0
Services.nidw.gov.bd	50	0	0
Foodies.com	not listed	53	0

Table 5.3: Final Dataset

The low number of dynamic scripts in the above table indicates the prevalence of dynamically generated scripts on Bangladeshi websites. The main goal of this research was to get insight about this fact that, how common it is to use dynamically generated scripts in Bangladeshi websites. The results show that it is very rare and infrequent to use dynamically generated scripts based on user's session state.

5.2.4 Investigating the Security Aspects and Exploitability

Now that all necessary information is stored, the manual analysis is begun. Each dynamic script of corresponding website is thoroughly examined to find a way of exploitability. Once found such pattern, attempts are taken to exploit the corresponding information. Table 5.4 shows the derived results of manual analysis

Selected Domain	Alexa Rank	Dynamic Scripts	Exploitable
Prothom-alo.com	7	2	No
Daraz.com	14	1	1
Bdjobs.com	15	2	No
Grameenphone.com	20	0	No
Teletalk.com	21	0	No
Bioscopelive.com	26	0	No
Bikroy.com	27	0	No
Crazy-hd.com	31	0	No
Services.nidw.gov.bd	50	0	No
Foodies.com	not listed	0	No

Table 5.4: Exploitability of Dataset

The result shows that, surprisingly there is no such dynamic scripts which can be exploitable by any means. All the dynamic scripts were either for banner rotation or any other design variations which take place after the user is authenticated to the site. This experiment indicates that there is no extend use of scripting languages like JavaScript, on websites of Bangladesh.

5.2.5 Discussing Potential Countermeasures

If there existed in pattern by which the the information could be exploited via dynamically generated scripts, the potential countermeasures to avoid such data leakage are to be discussed in this phase. If any website is developed strongly based on the scripting language, it is more prone to such data leakage. The first and foremost countermeasure to avoid this is, not keeping any sensitive information on any script code. Personal information should be used on static JavaScript, rather on dynamic.

5.2.6 Discussing the Prevalence of Dynamic Scripts

The main research question of this study was to find out the commons of using dynamic scripts on websites of Bangladesh. The results and findings mentioned above indicates that Bangladesh is far behind in the position of using dynamic scripts. Extended use of modern JavaScript is hardly seen in the selected websites. Unlike the existing study based on this topic [3], which was conducted on international websites, this study highlights the true condition of such usage of dynamic scripts.

5.3 Summary

This chapter intends to demonstrate the implementation and result analysis of the proposed approach. An extension along with a server is implemented to complete the overall approach of the study. The results are analyzed in depth to identify the condition of the dataset. The outcome of the study shows that the usage of dynamic scripts in Bangladeshi dataset is very rare. It also indicates that, the experimentation conducted on different dataset lead lead to different results, representing the condition of the target dataset. The next chapter shows a case study on a sample demo projec.

Chapter 6

A Case Study on Demo Website "vulnerable.com"

The result analysis of the developed architecture shows the proficiency of the proposed framework, necessary for conducting the empirical study. However, a step-by-step study may increase the understanding about the basic foundation of the study. Moreover, this will also help to justify the decision taken about the conditions of Bangladeshi websites on the focused vulnerability, XSSI. Thereby, this chapter provides a phase-to-phase analysis on a demo project, where the whole experiment is conducted. This experiment includes detecting dynamically generated scripts based on user's session state along with the exploitability related to those scripts. The processed outcomes of different phases of the experiment are also showed in this chapter in forms of snapshots.

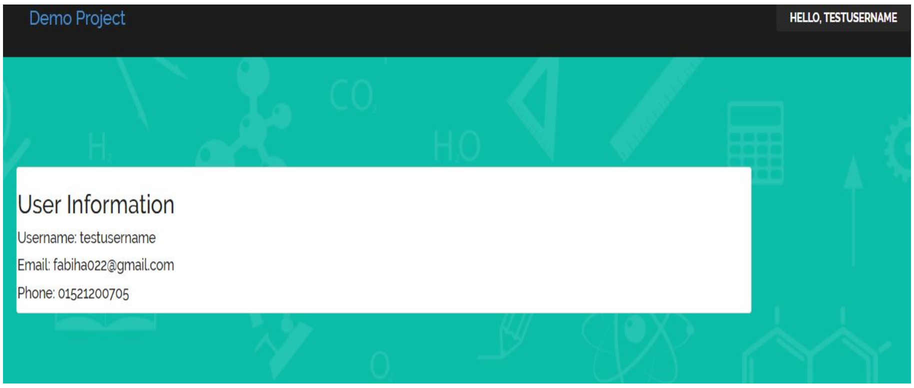
6.1 Phases of the case study

For assessment of the proposed study approach along with the developed extension, the whole process is conducted on a simple php based website. This website has the facility of user's registration process through which vulnerability depending on user's session state can be detected. After being authenticated to the site, some personal information such as user name, are displayed on the welcome page. The target of the study is, whether or not there is a possibility for an attacker to leak this personal information depending on dynamic scripts. Furthermore, if there is a possibility, then the ultimate data leakage process has been identified depending on the script's code pattern. Finally, exploitation of user's information has been taken place, making the fact clear that, use of dynamically generated scripts can lead to severe information leakage occurrence.

The main phases of the case study are: (1) Registration on the target website, (2) Inspection of source code, (3) Applying the extension on target site, (4) Analyzing the stored data, (5) Attempting towards information leakage, and (6) Observe the exploitability.

6.1.1 Registration on the target website

As mentioned in the earlier chapters, the first step to detect information leakage through dynamic script generation, is to fill the target website with personal statefull information. Therefore, manual registration has been done on the demo website. Figure 6.1 shows the after login interface. It can be observed from this procedure, that the interface has been changed based on the authentication, and some personal information such as user-name, are also shown in the after login interface.



User Information

Username: testusername

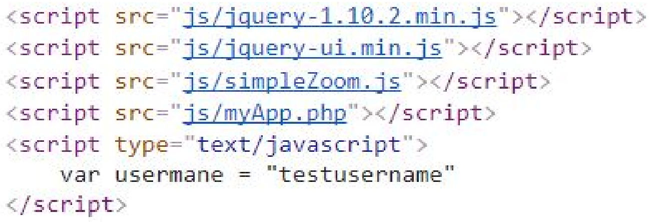
Email: fabiha022@gmail.com

Phone: 01521200705

Figure 6.1: After Login Interface

6.1.2 Inspection of source code

After registration, inspection of source code of the target website has been conducted to gather a brief idea about how the current web page is built, and how the external script resources are used. Figure 6.2 represents the snippet of the included scripts by the demo website.

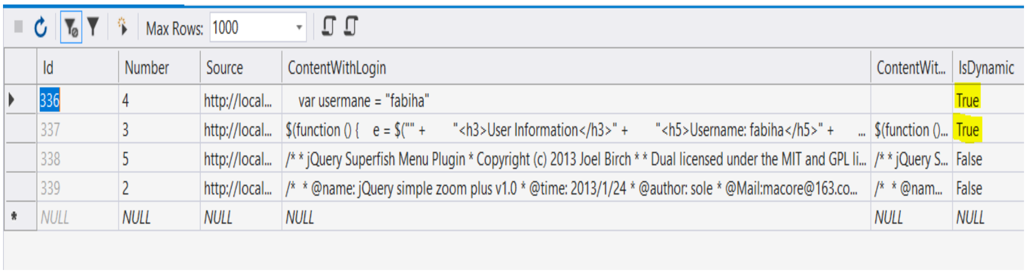


```
<script src="js/jquery-1.10.2.min.js"></script>
<script src="js/jquery-ui.min.js"></script>
<script src="js/simpleZoom.js"></script>
<script src="js/myApp.php"></script>
<script type="text/javascript">
    var usermane = "testusername"
</script>
```

Figure 6.2: Included Scripts of Target Website

6.1.3 Applying the extension on target site

Now, the extension will be loaded on the target web page. After the extension has loaded, all the included scripts shown in Figure 6.2 will be stored to the backend server. The detection procedure will take place and dynamic scripts will be identified by the server. The identified dynamic scripts are showed in Figure 6.3. It can be seen in the figure that there are two dynamic scripts found in the target website. One dynamic script is direct written in the source page, another dynamic script is included from a source link.



	Id	Number	Source	ContentWithLogin	ContentWit...	IsDynamic
▶	336	4	http://local...	var usermane = "fabiha"		True
	337	3	http://local...	$(function () { e = $("" + "<h3>User Information</h3>" + "<h5>Username: fabiha</h5>" + ...	$(function ()...	True
	338	5	http://local...	/* * jQuery Superfish Menu Plugin * Copyright (c) 2013 Joel Birch * * Dual licensed under the MIT and GPL li...	/* * jQuery S...	False
	339	2	http://local...	/* * @name: jQuery simple zoom plus v1.0 * @time: 2013/1/24 * @author: sole * @Mail:macore@163.co...	/* * @nam...	False
*	NULL	NULL	NULL	NULL	NULL	NULL

Figure 6.3: Dynamic Scripts of Demo Website

6.1.4 Analyzing the stored data

As There are two dynamic scripts found in the target demo website, the next step is to analyze the scripts very closely. Following is the source code of the dynamic script found on that website.

```
$(function () {
    e = $(\"\" +
        \"<h3>User Information</h3>\" +
        \"<h5>Username: "fabiha"</h5>\" +
        \"<h5>Email: "fabiha022@gmail.com"</h5>\" +
        \"<h5>Phone: "01521200705"</h5>\");
    $(\"#effect\").html(e);
    var options = {};
    $(\"#effect\").effect(\"bounce\", options, 500);
});
```

Listing 6.1: Dynamic Script Code

By analyzing the above code, it can be assumed that there is a object `e`, which is created by the personal information. Moreover, this personal information holding object utilizes some effect to be visible on the interface. This means, the information are to be shown on the html page, where the id of that intended html tag is `effect`. Thus, if attacker include this script on attacker's website and include a tag identified by `effect`, the expected information will be immediately posted to the attacker's website on that corresponding tag.

6.1.5 Attempting towards information leakage

To exploit the information, following html code is added to the developed server which is assumed to be the attacker's website.

```
<p id="effect">Sensitive information from targetted vulnerable
   website will be posted here.</p>
```

Listing 6.2: HTML Code on Attacker's Web-page

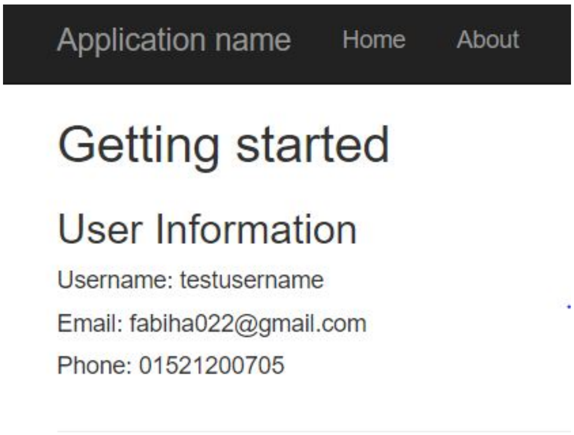
After that the intended script is included. As soon as the script is executed, all the information will be posted to the intended part of the attacker's page. Following is the code by which dynamic script is included to the attacker's website.

```
<script src="http://localhost/xssi_vulnerable_website/js/myApp.php"></script>
<script>
    var info = document.getElementById("effect");
</script>
```

Listing 6.3: Dynamic Script Inclusion on Attacker's Website

6.1.6 Observe the exploitability

Figure 6.4 represents the snippet of attacker's website, where the information of target vulnerable website is posted. By observing this, it can be assumed, how easy it is to exploit such vulnerability and get access to the user's sensitive information.



Getting started

User Information

Username: testusername

Email: fabiha022@gmail.com

Phone: 01521200705

Figure 6.4: Dynamic Scripts of Demo Website

6.2 Summary

This case study shows the basic structure of the experiment procedure. The stepwise demonstration of the approach gives a clear view of the severity and the dangers of using dynamic scripts on any websites. It also gives an idea about how the automated architecture works as the foundation of the empirical study. Manual phase of study is also well discussed in the context of this case study. This helps to approach properly in exploiting the data which is presented in the dynamic scripts. The exploitability of the information indicates, how information can be leaked through dynamically generated scripts,based on user's authentication state at minimal effort. The next chapter concludes this thesis after providing a proper future direction.

Chapter 7

Conclusion

Unlike conventional detection approaches, this research proposes to incorporate both automated and manual analysis in vulnerability detection. A framework consisting of a chrome browser extension and backend server is proposed for this purpose. The backend server also serves the purpose as attacker.com to attempt to exploit the vulnerabilities found on any dataset. In this chapter, the document is concluded by a profound but brief discussion of the research along with the future direction.

7.1 Discussion

This thesis introduces a vulnerability detection approach based on the usage of dynamic scripts on any website. An empirical study has been conducted on Bangladeshi dataset to analyze the true condition of Bangladesh in this vulnerability aspect. The study is performed by manual and automated approaches. Manual analysis includes the dataset selection process, registration process and exploitation process. As the foundation of exploitation technique an automated framework has also been introduced in this research. A chrome browser extension along with a backend server works together to gather the information of the existence of dynamic script on the investigated website. To identify the importance and the prevalence of this types of vulnerability, was the main idea of this research.

Most of the existing literature focus on identifying the other popular vulnerabilities such as XSS or CSRF. Vulnerability generated from dynamic scripts, which is known as XSSI, is hardly gotten any focus in the existing studies. This type of vulnerability is very fame-less but widespread across world wide web applications. Therefore, this study focus on this most avoided security aspect and tries to find out the actual condition of Bangladesh in the place of dynamic script usage.

From the results of the empirical study, it is highlighted that the prevalence of the usage of such kind of scripts on Bangladeshi website is very low. Extended use of scripting language is hardly seen on any web application of Bangladesh. Most of the websites depend on server for sending the total view of the page rather than sending data in form of script. Many upgraded features of scripting language are hardly seen to follow. A case study is also showed to evaluate the performance of the proposed approach. 7Thus, this research finds out the conditions of Bangladeshi website in the usage of dynamically generated scripts based on user's session state.

7.2 Future Work

The idea of detecting vulnerabilities based on the usage of dynamic scripts opens a number of directions for future research. Investigating this security aspect on different dataset leads to different types of result. This also identifies the true condition of the websites, whether these are prone to leak information via dynamic scripts or not. To automate the manual analysis of the exploitation technique can also be a future issue. This automation can lead to the achievement of automated detection and exploitation technique. The current implementation is evaluated only on ten websites. The future challenge lies in performing the overall approach on a vast scale dataset and finding a proper exploitable script. Thus, there is a scope of improvement which can be done by introducing a proper dynamic script related vulnerability detection and exploitation technique.

Appendix A

Dataset Details

Following table A shows the details of the dataset selection procedure:

Domain Name	**Alexa Rank**	**Selection Criteria**			**Final Result**
		Bangladeshi	**Authentication**	**Duplicate**	
Google.com	1	No	Yes	No	Discarded
Youtube.com	2	No	Yes	No	Discarded
Facebook.com	3	No	Yes	No	Discarded
Prothom-alo.com	4	Yes	Yes	No	Selected
Google.com.bd	5	No	Yes	Yes	Discarded
Yahoo.com	6	No	Yes	No	Discarded
Kalerkantho.com	7	Yes	No	No	Discarded
Daraz.com.bd	8	Yes	Yes	No	Selected
Bd24live.com	9	Yes	No	No	Discarded
Wikipedia.org	10	No	No	No	Discarded
Banglanews24.com	11	Yes	No	No	Discarded
Bd-pratidin.com	12	Yes	No	No	Discarded
Ntvbd.com	13	Yes	No	No	Discarded
Zoombangla.com	14	Yes	No	No	Discarded
Xvideos.com	15	No	Yes	No	Discarded

Table A.1: Dataset Selection Procedure

Domain Name	**Alexa Rank**	**Selection Criteria**			**Final Result**
		Bangladeshi	**Authentication**	**Duplicate**	
Bdjobs.com	16	Yes	Yes	No	Selected
Blogspot.com	17	No	Yes	No	Discarded
Jagonews24.com	18	Yes	No	No	Discarded
Rtvonline.com	19	Yes	No	No	Discarded
Teletalk.com.bd	20	Yes	Yes	No	Selected
Grameenphone.com	21	Yes	Yes	No	Selected
Cricbuzz.com	22	No	No	No	Discarded
Trackingclick.net	23	No	Yes	No	Discarded
Espncricinfo.com	24	No	Yes	No	Discarded
Bongacams.com	25	No	Yes	No	Discarded
Bikroy.com	26	Yes	Yes	No	Selected
Xnxx.com	27	No	Yes	No	Discarded
Bdnews24.com	28	Yes	No	No	Discarded
Nu.edu.bd	29	Yes	No	No	Discarded
Crazyhd.com	30	Yes	Yes	No	Selected
Popads.net	31	No	Yes	No	Discarded
Roar.media	32	No	No	No	Discarded
Deloton.com	33	No	No	No	Discarded
Savefrom.net	34	No	No	No	Discarded
Doubleclick.net	35	No	Yes	No	Discarded
Bioscopelive.com	36	Yes	Yes	No	Selected
Bd24report.com	37	Yes	No	No	Discarded
Softonic.com	38	No	Yes	No	Discarded
Banglarkhobor24.com	39	Yes	No	No	Discarded
Iwanttodeliver.com	40	No	Yes	No	Discarded
Deshebideshe.com	41	Yes	No	No	Discarded
Instagram.com	42	No	Yes	No	Discarded

Table A.2: Dataset Selection Procedure (Continuing from Table A)

Domain Name	**Alexa Rank**	**Selection Criteria**			**Final Result**
		Bangladeshi	**Authentication**	**Duplicate**	
Wordpress.com	43	No	Yes	No	Discarded
Gonews24.com	44	Yes	No	No	Discarded
Fiverr.com	45	No	Yes	No	Discarded
Thedailystar.net	46	Yes	No	No	Discarded
Ask.com	47	No	Yes	No	Discarded
Amazon.com	48	No	Yes	No	Discarded
Earnstations.com	49	No	Yes	No	Discarded
Nidw.gov.bd	50	Yes	Yes	No	Selected

Table A.3: Dataset Selection Procedure (End of Table A)

References

[1] *Definition of a Security Vulnerability.* Online; Accessed; 2017-07-21. Microsoft Developers Network. 2017. URL: `https://msdn.microsoft.com/en-us/library/cc751383.aspx`.

[2] *Internet Security Threat Report.* Symantec Enterprise Security, 2005.

[3] S. Lekies, B. Stock, M. Wentzel, and M. Johns. "The Unexpected Dangers of Dynamic JavaScript." In: *USENIX Security Symposium.* 2015, pp. 723–735.

[4] J. Ruderman. *Same-Origin-Policy.* Online; Accessed: 2017-10-25. URL: `https://developer.mozilla.org/enUS/docs/Web/Security/Same-origin_policy`.

[5] M. Zalewski. *The tangled Web: A guide to securing modern web applications.* No Starch Press, 2012.

[6] *Cross-site Scripting (XSS).* Online; Accessed; 2017-10-23. 2016. URL: `https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)`.

[7] *Cross-Site Request Forgery (CSRF).* Online; Accessed; 2017-10-19. 2017. URL: `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)`.

[8] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo. "Securing web application code by static analysis and runtime protection". In: *Proceedings of the 13th international conference on World Wide Web.* ACM. 2004, pp. 40–52.

[9] V. B. Livshits and M. S. Lam. “Finding Security Vulnerabilities in Java Applications with Static Analysis.” In: *USENIX Security Symposium.* Vol. 14. 2005, pp. 18–18.

[10] Y. Xie and A. Aiken. “Static Detection of Security Vulnerabilities in Scripting Languages.” In: *USENIX Security Symposium.* Vol. 15. 2006, pp. 179–192.

[11] C. Kern, A. Kesavan, and N. Daswani. *Foundations of security: what every programmer needs to know.* Springer, 2007.

[12] T. Terada. *Identifier based XSSI attacks.* Mitsui Bussan Secure Directions, Inc., 2015.

[13] M. Vieira, N. Antunes, and H. Madeira. “Using web security scanners to detect vulnerabilities in web services”. In: *Dependable Systems & Networks, 2009. DSN’09. IEEE/IFIP International Conference on.* IEEE. 2009, pp. 566–571.

[14] *HP WebInspect.* 2008. URL: `http://www.hp.com.`.

[15] *IBM Security AppScan.* 2008. URL: `http://www-03.ibm.com/software/products/en/appscan`.

[16] *Audit Your Web Security with Acunetix Vulnerability Scanner.* 2008. URL: `https://www.acunetix.com/vulnerability-scanner/`.

[17] S. Lekies, B. Stock, and M. Johns. “25 million flows later: large-scale detection of DOM-based XSS”. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security.* ACM. 2013, pp. 1193–1204.

[18] T. Farah, M. Shojol, M. Hassan, and D. Alam. “Assessment of vulnerabilities of web applications of Bangladesh: A case study of XSS & CSRF”. In: *Digital Information and Communication Technology and its Applications (DICTAP), 2016 Sixth International Conference on.* IEEE. 2016, pp. 74–78.

[19] D. Alam, T. Farah, and M. A. Kabir. "Exploring the SQL injection vulnerabilities of. bd domain web applications". In: *3rd International Conference on Advances in Computing, Electronics and Communication (ACEC 2015)*. 2015, pp. 110–114.

[20] *Getting Started: Building a Chrome Extension*. Online; Accessed: 2017-11-12. URL: `https://developer.chrome.com/extensions/getstarted`.

[21] B. Chess, Y. T. O'Neil, and J. West. *JavaScript hijacking*. Fortify Software, 2007.

[22] *Web 2.0*. Online; Accessed: 2017-10-22. URL: `http://whatis.techtarget.com/definition/Web-20-or-Web-2`.

[23] *Mash-up*. Online; Accessed: 2017-10-21. URL: `http://whatis.techtarget.com/definition/mash-up`.

[24] V. Hailperin. *Cross-Site Script Inclusion*. Online; Accessed: 2017-10-20. 2016. URL: `https://www.scip.ch/en/?labs.20160414`.

[25] *Category:OWASP Top Ten Project*. Online; URL: `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`.

[26] *Milad's Blog*. Online; Accessed; 2017-10-21. 2013. URL: `http://blog.miladbr.ir/2013/03/cross-site-script-inclusion.html`.

[27] B. Leban, M. Bendre, and P. Tabriz. *Web Application Exploits and Defenses*. Online; Accessed: 2017-10-28. URL: `http://google-gruyere.appspot.com/part3#3__cross_site_script_inclusion`.

[28] H. Veit. *The tale of a fameless but widespread vulnerability*. `https://speakerdeck.com/luh2/the-tale-of-a-fameless-but-widespread-vulnerability`. scip AG, 2016.

[29] *Send JSONP Cross-domain Requests*. Online; Accessed; 2017-10-21. JSON Community. 2013. URL: `http://json.com/2013/12/16/send-jsonp-cross-domain-requests.html`.

[30] B. Stock, S. Lekies, T. Mueller, P. Spiegel, and M. Johns. “Precise Client-side Protection against DOM-based Cross-Site Scripting.” In: *USENIX Security Symposium*. 2014, pp. 655–670.

[31] J. Grossman. *Advanced Web Attack Techniques using GMail.* Online; Accessed: 2017-10-25. URL: `http://blog.jeremiahgrossman.com/2006/01/advanced-web-attack-techniques-using.html`.

[32] *ECMAScript 6 — New Features: Overview & Comparison.* Online; Accessed: 2017-10-20. URL: `http://es6-features.org/#Constants`.

[33] *MDN web docs. Proxy.* Online; Accessed: 2017-10-23. URL: `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy`.

[34] *Security: Cross-origin information leak via ECMAScript harmony proxies.* Online; Accessed: 2017-10-25. URL: `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy`.

[35] J. Fonseca, N. Seixas, M. Vieira, and H. Madeira. “Analysis of field data on web security vulnerabilities”. In: *IEEE transactions on dependable and secure computing* 11.2 (2014), pp. 89–100.

[36] *Message Passing.* Online; Accessed; 2017-11-29. URL: `https://developer.chrome.com/apps/messaging`.

[37] J.-e. van Bergen. *What Is Mixed Content?* `https://developers.google.com/web/fundamentals/security/prevent-mixed-content/what-is-mixed-content`. Google Developers: Web Fundamentals, 2017.

[38] *Inheritance and the prototype chain.* Online; Accessed; 2017-11-29. MDN web docs. 2017. URL: `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Inheritance_and_the_prototype_chain`.

[39] *Microsoft Visual Studio.* 2017. URL: `https://www.visualstudio.com/vs/`.

[40] *Chrome Developers' Tool*. 2017. URL: https://developer.chrome.com/devtools.

[41] *SQL Server*. 2016. URL: https://www.microsoft.com/en-us/sql-server/sql-server-2016.

[42] *Top Sites in Bangladesh*. Online; Accessed; 2017-12-19. Alexa. 2017. URL: https://www.alexa.com/topsites/countries/BD.