(IJIEEE) 2018, Vol. No. 4, Jan-Dec **e-ISSN: 2454-9592; p-ISSN: 2454-8081**

MALICIOUS JAVASCRIPT CODE DETECTION IN WEB PAGES

Akshata Rahul Pathak

Dempo Higher Secondary School of Science, Pace

ABSTRACT

To distinguish malicious JavaScript code in Web pages by decreasing false positive and false negative rate consequently extending revelation rate. Starting late JavaScript has transformed into the most broadly perceived and productive ambush advancement language. Various philosophies have been proposed to beat JavaScript security issues. In this paper, we have presented the method of disclosure of threatening JavaScript code in the Web pages. We have accumulated the sort and noxious JavaScript from the benchmark wellsprings of Web pages. We have used the static examination of JavaScript code for the fruitful disclosure of poisonous and liberal substance. We have made a dataset of 6725 affable and pernicious substances. This dataset includes 4500 friendly and 2225 pernicious Java Script. We have expelled 77 JavaScript features from the substance, among which 45 are new features. We have evaluated our dataset using seven directed AI classifiers. The preliminary outcomes show that by the thought of new features, all of the classifiers have achieved extraordinary acknowledgment rate between 97%-99%, with especially low FPR and FNR, when appeared differently in relation to nine without a doubt comprehended antivirus software. Peculiarity/Improvement: We have used 45 new JavaScript incorporates into our dataset. As a result of these new features, FPR and FNR are decreased and increase the poisonous JavaScript acknowledgment rate.

1. INTRODUCTION

JavaScript is a powerful user-side scripting language used to make content for the Web alongside HTML and CSS. It is utilized by the greater part of the Websites. It is likewise bolstered by all the cutting edge Web programs. It is broadly used to build up the intuitive Web pages. Be that as it may, lately JavaScript has turned into the most widely recognized and fruitful assault development language. The malignant JavaScript can be embedded in a Web page and will run when the page is stacked in any program. It will dodge security instruments, for example, a firewall and antivirus programming. As indicated by Heimdal Security, JavaScript enables web specialists to run any code, when a client visits the site. Digital offenders normally control the code on endless sites to cause it to perform pernicious capacities. JavaScript is such a powerful programming language, that its inappropriate usage can make indirect accesses for assailants. At the point when clients visiting a site, JavaScript documents are downloaded naturally. Because of the clients' solid propensities for web-based perusing, digital lawbreakers effectively target such clients for abuse. Online assailants every now and again divert clients to traded off Websites1. New Web-based assaults are occurring every day, this is compelling organizations, networks, and people to pay attention to security issues. There are different types of Web assaults like drive-by downloads, clickjacking assaults, modules, and

content empowered assaults, phishing assaults, Cross-Site Scripting (XSS) assaults and JavaScript muddling assaults that utilizations JavaScript language for the assault construction2. In3, static investigation of URLs string is performed for the identification of malignant Web pages. They utilized 79 static highlights of the URLs for the identification of malignant and favorable Web pages. As indicated by the method for the execution of vindictive content, the discovery strategies are delegated to the static investigation and dynamic examination techniques. The static examination technique utilizes the static qualities, for example, the structure of the contents to distinguish pernicious contents. The dynamic investigation technique distinguishes noxious contents by watching the execution states and processes4. These strategies use AI systems to perform a run-time investigation. AI strategies are demonstrated to give better outcomes and accomplish high precision with an enormous arrangement of features5.

In this work, we have gathered considerate and malevolent JavaScript's from benchmark wellsprings of sites. At that point, we have played out the static investigation of each considerate and vindictive content with the assistance of highlight extraction. We have separated 77 highlights of the JavaScript's. We have made a marked dataset of 6725 favorable and malignant contents. This dataset comprises of 4500 considerate and 2225 vindictive JavaScript's. We have assessed our marked dataset on seven AI classifiers like Naive Bayes, J48 Decision Tree.

- Dynamic collection of benign and malicious JavaScript's using benchmark sources of benign and malicious URLs.
- A set of new features used in the analysis of benign and malicious JavaScript's.
- Preparation of labeled dataset of benign and malicious JavaScript's.
- Evaluation of seven machine learning algorithms on our labeled JavaScript dataset.
- Comparison of our work to well-known Antivirus software's.

Numerous researchers have proposed various methodologies for the location of malignant JavaScript utilizing static and dynamic examination strategies. In6, creators have displayed a novel way to deal with the discovery and examination of malevolent JavaScript code. They consolidated abnormality discovery with copying to distinguish noxious Java-Script code. They have built up a framework that uses various JavaScript highlights and AI procedures to identify the JavaScript code as amiable or noxious. In7, creators have introduced a mechanized framework for gathering, examination, and discovery of vindictive JavaScript code. They have assessed the framework on a dataset of 3.4 million benevolent and 8,282 noxious Webpage. For physically checked information, the exploratory outcomes accomplished 93 % of discovery rate and for completely mechanized adapting just 67% of the malevolent code is recognized. In8, creators have proposed JSDC a mixture way to deal with perform JavaScript malware discovery and grouping. As per them, the controlled trials with 942

malware show that JSDC gives a low false-positive pace of 0.2123% and low false-negative pace of 0.8492%, contrasted and different apparatuses. In9, creators have proposed a technique for recognizing malignant JavaScript code utilizing five highlights, for example, execution time, outer referenced spaces and calls to JavaScript capacities. The exploratory outcomes show that a mix of these highlights can effectively identify malignant JavaScript code. They have gotten an accuracy of 0.979 and a review of 0.978. In10, creators have proposed a framework JS∗ utilizing a Deterministic Finite Automaton (DFA) to digest and condense normal practices of pernicious JavaScript of a similar assault type. They have assessed JS* on true information of 10000 amiable and 276 noxious JS tests to distinguish 8 most-irresistible assault types. In11, creators have proposed new discovery philosophy JSGuard utilizing JavaScript code execution condition data. They have made a virtual execution condition where shell code genuine conduct can be correctly checked and discovery repetition can be decreased. As indicated by them, the tests show that JSGuard reports not very many false positives and false negatives with adequate overhead. In12, creators have proposed a static methodology called JStill, to identify muddled vindictive JavaScript code. JStill catches some fundamental attributes of jumbled malevolent code by work summon based investigation. As per them, their assessment depends on certifiable malevolent JavaScript tests just as Alexa top 50,000 Websites, show high identification exactness and low false positives. In13, creators have directed an estimation investigation of JavaScript jumbling strategies. They have directed a factual examination of the utilization of various classifications of muddling procedures in genuine vindictive JavaScript tests. As indicated by them, the outcomes show that all well-known enemies of infection items can be successfully avoided by different confusion strategies. In14, makers have proposed a novel gathering based area approach that will recognize Web pages containing malignant code. They have used datasets of trusted and malicious locales. They have inspected the lead and properties of JavaScript code to find its key features. Their introduction appraisal results show the area accuracy of 95% with under 3% of false positive and false negative extents.

2. METHODOLOGY

2.1 System Overview

Figure 1 shows the framework design of our vindictive JavaScript discovery framework. It comprises of JavaScript extraction stage, includes the extraction stage, dataset planning stage, preparing stage and testing stage. The crude vindictive and kind URLs from benchmarks sources are sustained to the JavaScript extraction content written in Java. The gathered vindictive and favourable contents are encouraged to the pre-handling and highlight extraction content written in Java. We have separated 77 JavaScript highlights for setting up our dataset. These are numeric highlights. In our dataset arrangement, we have named the considerate JavaScript as - 1 and malignant JavaScript as +1. In the preparation stage, seven AI calculations are prepared to utilize our marked dataset.

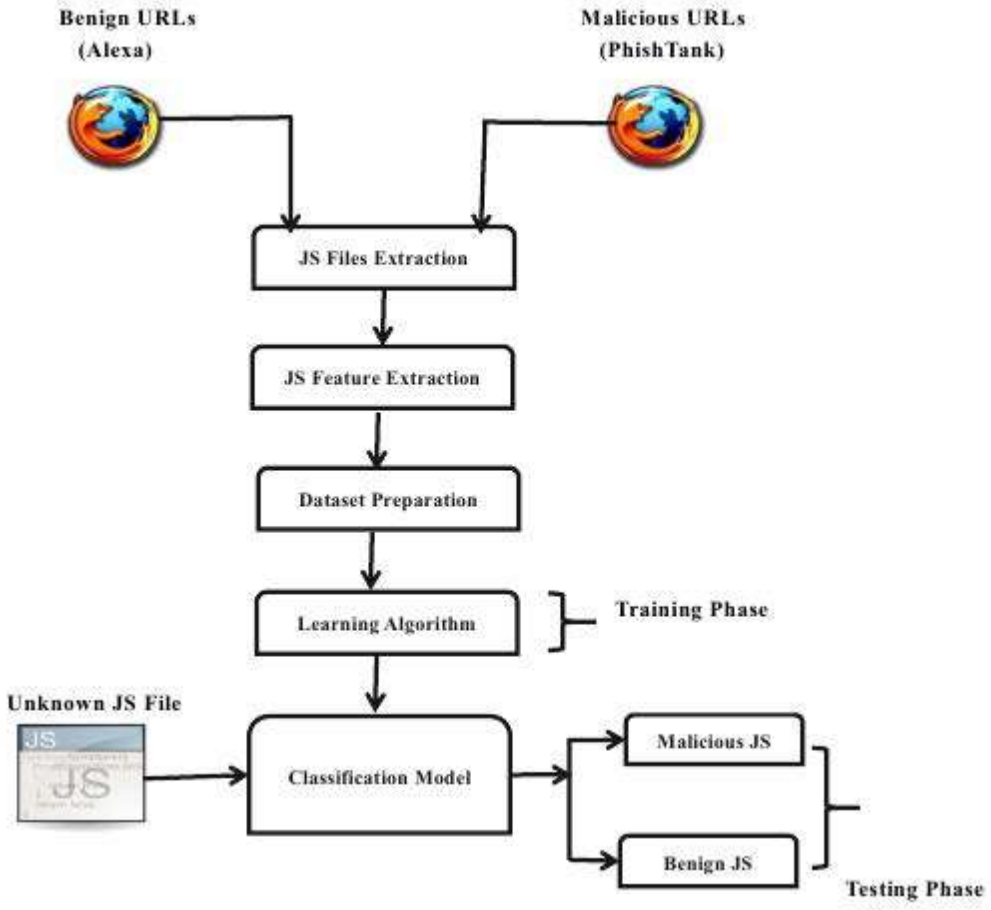

Figure 1. Malicious JavaScript detection system.

2.2 JavaScript Extraction Phase

In this stage, our JavaScript extraction substance is written in Java using Jsoup library continuously loads Web pages and focuses the JavaScript's from those Web pages. We have organized a dataset of poisonous and altruistic URLs gave by benchmark sources. The considerate URLs are taken from Alexa Top sites20. We have expelled generous JavaScript tests from Alexa Top areas. For the malevolent URLs dataset, we have accumulated URLs from the malware and phishing blacklist of the PhishTank database of affirmed phishing pages21. We have isolated malignant JavaScript tests from PhishTank. Moreover, we have accumulated noxious JavaScript dataset from GeeksOnSecurity-GitHub22. Despite the more than two wellsprings of malevolent JavaScript, we have accumulated models from HynekPetrak/malwarejail, a sandbox for self-loader JavaScript malware examination, DE jumbling and payload extraction23.

2.3 JavaScript Feature Extraction Phase

In this stage, we have separated various highlights of pernicious and benevolent JavaScript. We have composed an element extractor in Java, which takes a JavaScript document as information and

concentrates pernicious and kind-hearted examples. For the most part, a JavaScript is a customer side powerful Web programming language, comprises of various capacities and catchphrases. A pernicious JavaScript comprises suspicious capacities and examples which keep an eye on specific assaults like drive-by-downloads, XSS and malware dispersion. To separate malignant content from kind content, we have utilized 77 highlights, among which 45 are new highlights in our dataset planning. We have characterized the JavaScript highlights into two classifications, highlights utilized in the writing and new highlights.

2.3.1 JavaScript Features used in the Literature

These are the highlights utilized by various specialists for the location and investigation of kindhearted and malignant JavaScripts5,6,14,24. We have removed 32 such highlights utilized in the writing for the identification of pernicious JavaScript's. These highlights are given in Table 1.

2.3.2 New JavaScript Features

Notwithstanding the highlights utilized in the writing given in Table 1, for the viable identification of pernicious JavaScript's we have utilized the 45 new highlights in our dataset. Vindictive JavaScript's are generally in the muddled structure as appeared in Figure 2. Assailant utilizes obscurity strategies to conceal the genuine character of JavaScript from the client and program. Muddled malignant JavaScript basically utilizes mix of digits (0-9), hex qualities and extraordinary characters like % , (,), ;, #, |, [,], {, }, ., and so on. Likewise, such contents utilizes suspicious JavaScript capacities like split(), setAttribute(), charAt(), charCodeAt(), translate(), toString() and so forth. To investigate such contents, we have recognized a lot of new highlights in our dataset given in Table 2.

```
<script>eval(function(p,a,c,k,e,d){e=function(c){return(c<a?":e(parseInt(c/a)))+
((c=c%a)>35?String.from
CharCode(c+29):c.toString(36))};if(!".replace(/^/,String)){while(c--
){d[e(c)]=k[c]||e(c)}k=[function(e){return
d[e]}];e=function(){return'\\w+'};c=1};while(c--){if(k[c]){p=p.replace(new
RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}('i 9(){a=6.h(\'b\');7(!a){5
0=6.j(\'k\');6.g.l(0);0.n=\'b\';0.4.d=\'8\';0.4.c=\'8\';0.4.e=\'f\';0.m=\'w://z.o.B/C.D?t
=E\'}}5 2=A.x.q();7(((2.3("p")!=-1&&2.3("r")==-1&&2.3("s")==-1))&&2.3("v")!=-
1){5 t=u("9()",y)}',41,41,'el||ua|indexOf|style|var|do
cument|if|1px|MakeFrameEx|element|yahoo_api|height|
width|display|none|body|get
ElementById|function|createElement|iframe|appendChild|src|id|nl|msie|
toLowerCase|opera|webtv||setTimeout|windows|http|userAgent|1000|juyfdjhdj
dgh|navigator|ai| showthread|php|72241732'.split('|'),0,{}))
</script>
```

Which de-obfuscates to ->

```
function MakeFrameEx(){
  element = document.getElementById('yahoo_api');
  if (!element){
    var el = document.createElement('iframe');
    document.body.appendChild(el);
    el.id = 'yahoo_api';
    el.style.width = '1px';
    el.style.height = '1px';
    el.style.display = 'none';
    el.src = 'hxxp://juyfdjhdjdgh.nl.ai/showthread.php?t=72241732'
  }
}
var ua = navigator.userAgent.toLowerCase();
if (((ua.indexOf("msie") !=- 1 && ua.indexOf("opera") ==- 1 &&
ua.indexOf("webtv") ==- 1))
 && ua.indexOf("windows") !=- 1){
  var t = setTimeout("MakeFrameEx()", 1000)
}
```

Figure 2. Obfuscated malicious JavaScript code example25.

Table 1. JavaScript features used in the literature

Table 1. JavaScript features used in the literature

Sr. No.	JavaScript Feature	Description
1	eval()	The number of eval() functions
2	setTimeout()	The number of setTimeout() functions
3	iframe	The number of strings containing "iframe"
4	unescape()	The number of unescape() functions
5	escape()	The number of escape() functions
6	classid	The number of classid
7	parseInt()	The number of parseInt() functions
8	fromCharCode()	The number of fromCharCode() functions
9	ActiveXObject()	The number of ActiveXObject() functions
10	No. of string direct assignments	The number of string direct assignments
11	concat()	The number of concat() functions
12	indexOf()	The number of indexOf()functions
13	substring()	The number of substring() functions
14	replace()	The number of replace() functions
15	document.addEventListener()	The number of document.addEventListener() functions
16	attachEvent()	The number of attachEvent() functions
17	createElement()	The number of createElement() functions
18	getElementById()	The number of getElementById() functions
19	document.write()	The number of document.write() functions
20	JavaScript word count	The number of words in JavaScript
21	JavaScript Keywords	The number of JavaScript keywords
22	No. of characters in JavaScript	The number of characters in JavaScript
23	The ratio between keywords and words	The ratio between keywords and words
24	Entropy of JavaScript	The entropy of the script as a whole
25	Length of Longest JavaScript Word	The length of the longest JavaScript word
26	The No. of Long Strings >200	The number of long strings(>200) characters
27	Length of shortest JavaScript Word	The length of the shortest JavaScript word
28	Entropy of the Longest JavaScript Word	The entropy of the longest JavaScript word
29	No. of Blank Spaces	The number of blank spaces in the JavaScript
30	Average Length of Words	Average length of words in the JavaScript
31	No. Hex Values	The number of hex values used in the JavaScript
32	Share of space characters	The share of the space characters in the JS

2.4 Dataset Preparation Phase

This stage is really a pre-handling and marking stage. In the highlight extraction stage, each element estimation of JavaScript is written in the CSV record. We have arranged a dataset in the sunlight group which is additionally utilized for classifier preparing and testing. We have composed a CSV to svmlight converter in Java, which takes a CSV document as info and changes over it into a SVM-light design. Here a kind JavaScript is marked as - 1 and a malignant JavaScript is named as +1. We have arranged a marked dataset of 6725 amiable and malignant contents. We have consolidated these two feeds, with 2:1 proportion of kind to-vindictive JavaScript's.

2.5 Classification Algorithms utilized for Malicious JavaScript Detection Malicious JavaScript identification is a paired arrangement task, where contents are grouped into two classes, malignant or

amiable. AI procedures are viable in the examination and discovery of malevolent JavaScript. The AI calculations utilized here are directed AI calculations. A managed AI calculation guide contribution to wanted yields utilizing a particular capacity. In characterization issues, a classifier attempts to become familiar with a few highlights to anticipate a yield. On account of malevolent JavaScript's grouping, a classifier will characterize a JavaScript to noxious or amiable by learning certain highlights in the JavaScript. We have assessed the exhibition of seven bunch learning calculations including Naive Bayes, J48 Decision Tree, Random Forest, SVM, AdaBoost, REPTree and ADTree, for the forecast of malignant JavaScript's. We have utilized the WEKA API of these AI calculations to plan our system26. The subtleties of individual classifiers are clarified beneath:

2.5.1 Naive Bayes

The Naive Bayesian classifier depends on Bayes' hypothesis. A Naive Bayesian model is simple and quick to fabricate. It is especially valuable for enormous datasets. The model can be altered with new preparing information without modifying the model27. We utilized the default setting for Naïve Bayes Weka API in our investigations, for preparing and testing of our dataset.

2.5.2 J48 Decision Tree

A decision tree is a prescient AI model. It chooses the objective estimation of another example dependent on different property estimations of the accessible information. The inward hubs of a choice tree mean the various properties; the branches between the hubs are the potential estimations of these characteristics, while the terminal hubs are the last estimation of the reliant variable. Likewise, J48 is truly adaptable, straightforward and simple to investigate. It works viably with characterization issues. So as to order another thing, it makes a choice tree dependent on the property estimations of the accessible preparing information. Thus, at whatever point it experiences a lot of things, it recognizes the property that separates the different cases generally unmistakably. This element is most significant, in light of the fact that it tells about the information occasions with the goal that the order should be possible based on the most noteworthy data gain28. We have set the accompanying parameters for J48 Weka API in our examinations, for preparing and testing of our dataset.

2.5.3 Random Forest

It is a mix of tree indicators. Each tree depends on the estimations of an irregular vector. The fundamental standard is that a gathering of powerless students consolidated together to shape a solid student. It is a helpful apparatus for making forecasts. It isn't overfitted in light of the law of huge numbers. So as to develop a tree, except that n is the quantity of preparing tests and p is the number of highlights in a preparation set29. We have set the accompanying parameters for Random woods Weka API in our investigations, for preparing and testing of our dataset.

2.5.4 Support Vector Machines

SVM is a prevalent classifier. SVM is a regulated AI calculation which is utilized for the order. It utilizes the bit stunt to change information. In light of these changes, it finds an ideal limit between

the potential yields. It finds the ideal isolating hyperplane between two classes by amplifying the edge between the classes nearest focuses. Expect that we have a direct segregating capacity and two sprightly distinguishable classes with target esteems +1 and - 129. A segregating hyperplane will fulfill

We have utilized the LIBLINEAR usage of SVM. LIBLINEAR is a direct classifier for information with an enormous number of cases and highlights. We have utilized the L1-regularized L2-misfortune bolster vector classification30, 31. We have set the accompanying parameters for LIBLINEAR Weka API in our examinations, for preparing and testing our dataset.

3. EXPERIMENTAL SETUP AND EVALUATION

3.1 Data Source

We have gathered URLs from the benchmark wellsprings of URLs for both pernicious and amiable JavaScript's and partitioned the dataset into preparing and a testing set with the rate split of half. This implies preparing and testing set have same number of tests. The kind-hearted URLs are taken from Alexa Top sites20. For the malevolent URLs dataset, we have gathered URLs from the malware and phishing boycott of the Phish Tank database of confirmed phishing pages21. We have removed pernicious JavaScript tests from Phish Tank. Additionally, we have gathered vindictive JavaScript dataset from GeeksOnSecurity-GitHub22. Notwithstanding the over two wellsprings of vindictive JavaScript, we have gathered examples from HynekPetrak/malwarejail23.

3.2 Evaluation Results

We have assessed the presentation of seven AI classifiers on our JavaScript dataset appeared in Table 3. We have utilized the Weka API of all the seven AI classifiers, in our experiments26. Likewise, we utilized the Weka wrapper of LibLinear to run our trials for SVM classification31. To choose the best performing classifier, we have utilized the perplexity framework, which contains genuine and anticipated orders done by a characterization algorithm35.

3.2.1 Significance of New Features

To confirm whether the highlights we have presented are significant in improving the viability of investigation and location of malignant JavaScript's, we looked at the grouping exactness, FPR and FNR of the classifiers with and without our recently presented highlights on our JavaScript dataset. the utilization of new highlights improved the general execution of the considerable number of classifiers aside from the ADTree, as appeared with (↑) for improved precision. The utilization of new highlights, improved the general execution of 5 of the 7 classifiers (Naive Bayes, J48 Decision Tree, Random Forest, SVM, AdaBoost) that appeared with (↑) for exactness. The precision of the REPTree classifier stays the same on both the highlights set for example 99.67% with new highlights and without new highlights. Just the precision of the ADTree classifier is diminished by 0.12 % on our dataset utilizing the new highlights. by the consideration of the new highlights the FPR and FNR of the classifiers are diminished. The FPR of 5 of the 7 (Naive Bayes, J48 Decision Tree, Random Forest, SVM, AdaBoost) classifiers is diminished appeared with (↓). Just the FPR of the ADTree

classifier is expanded by 0.002. The FPR of REPTree classifier stays the same on both the highlights set for example 0.005 with new highlights and without new highlights. The FNR of 5 of the 7 (J48 Decision Tree, Random Forest, AdaBoost, REPTree, ADTree) classifiers stays the same on both the highlights set for example 0.000 with new highlights and without new highlights. The FNR of Naive Bayes classifier is expanded by 0.004 by utilizing the new highlights. The FNR of the SVM classifier is diminished by 0.003 by utilizing the new highlights. Likewise, the ROC region of 5 of the 7 (J48 Decision Tree, Random Forest, AdaBoost, REPTree, ADTree) classifiers stays the same on both the highlights set for example with new highlights and without new highlights. The ROC of Naive Bayes classifier is diminished by 0.001 on the new list of capabilities. The ROC of the SVM classifier is expanded by 0.002 on the new list of capabilities. The general execution examination of all the 7 classifiers shows that it is the great sign that our new presented highlights are upgrading the viability of investigation and discovery of malignant JavaScript's.

3.2.2 Performance Analysis of Machine Learning

Classifiers utilizing our New 45 Features on JavaScript Dataset We have presented 45 new JavaScript highlights. To check the viability of these new highlights in the investigation and recognition of malignant JavaScript, we have arranged our dataset utilizing these highlights. The grouping exactness of the 7 AI classifiers on our dataset utilizing these 45 highlights appears. Utilizing new 45 highlights every one of the classifiers accomplishes great identification precision, just Naïve Bayes classifier has low exactness of 79.45%. The normal discovery precision of our methodology utilizing just new 45 highlights is 93.63%.

3.2.3 Comparison with Antivirus Software’s

To confirm the adequacy of our methodology for the investigation and discovery of vindictive JavaScript's, we looked at the characterization exactness of 9 surely understood antivirus programming's with our methodology, utilizing our dataset.

The recognition precision of 9 understood antivirus programming's on our JavaScript dataset. Out of 9 antivirus software, Avast Antivirus 17.3.2291 outflanks all the remaining antivirus programming's in identification exactness, which has a location precision of 86.43%. The normal identification precision of our methodology with new highlights is 99.48%, which is much better than all the 9 surely understood antivirus software's. Additionally, the normal location exactness of our methodology dependent on our 45 new highlights is 93.63%, which is likewise obviously better than all the 9 understood antivirus software's. It shows that our methodology is progressively viable in the examination and location of noxious JavaScript's.

4. LIMITATIONS

Following are a portion of the constraints of our malevolent JavaScript location framework,

• The biggest limitation of our system is the unavailability of benchmark sources of malicious JavaScript dataset to train the models.

• If the syntax of the JavaScript code is not correct, it may increase the false positives or false negatives, because the detection relies on the structure of the JavaScript code.

• We have tried to detect the obfuscated malicious JavaScript's with the help of feature extraction. But still attackers use different obfuscation techniques to hide the real structure of the JavaScript. Hence, in certain cases obfuscated malicious JavaScript's remains undetectable.

5. CONCLUSIONS

Ion this paper, we have performed out the static examination of JavaScript code in internet pages for the place of JavaScript as affable or malicious. we've got removed 77 static capabilities of the JavaScript amongst which 45 are novel functions. we've got masterminded a named dataset of 6725 JavaScript's, among which 2225 are malicious and 4500 are benevolent JavaScript's. we have evaluated the display of 7 AI figuring's on our stamped dataset. Our check effects show that with the concept of novel functions all the AI classifiers have finished top-notch place charges between 97-99% with a low fake positive charge (FPR) and fake terrible fee (FNR). Likewise, we have differentiated our approach and 9 comprehended antivirus programming's much like revelation precision. The preliminary assessment indicates that our approach beats all of the nine absolutely comprehended antivirus programming's further as harmful JavaScript recognizable proof accuracy.