2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery

XSS Vulnerability Detection Using Optimized Attack Vector Repertory

Xiaobing Guo, Shuyuan Jin, and Yaxing Zhang
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
E-mails: guoxiaobing@software.ict.ac.cn, jinshuyuan@ict.ac.cn, zhangyaxing@software.ict.ac.cn

***Abstract*—In order to detect the Cross-Site Script (XSS) vulnerabilities in the web applications, this paper proposes a method of XSS vulnerability detection using optimal attack vector repertory. This method generates an attack vector repertory automatically, optimizes the attack vector repertory using an optimization model, and detects XSS vulnerabilities in web applications dynamically. To optimize the attack vector repertory, an optimization model is built in this paper with a machine learning algorithm, reducing the size of the attack vector repertory and improving the efficiency of XSS vulnerability detection. Based on this method, an XSS vulnerability detector is implemented, which is tested on 50 real-world websites. The testing results show that the detector can detect a total of 848 XSS vulnerabilities effectively in 24 websites.**

Keywords-XSS; attack vector repertory; machine learning; dynamic analysis; web crawler

I. INTRODUCTION

With the development of the web attack technologies, the web security is becoming one of the most common security issues. XSS is one of the most popular vulnerabilities in recent years, as discussed in [1]. With XSS vulnerabilities, attackers can steal the user accounts and modify the important data. Basically, a XSS vulnerability attack is an injection attack where the attackers send an XSS attack vector to the web server. Without any form of inputting validation, the XSS attack vector may occurs in the responding pages. Finally, the malicious scripts will be executed by the browser in user's computer and the attacker will gain the user's sensitive information, such as user's account numbers and passwords.

Vulnerability detection is to detect the XSS vulnerabilities in web applications and there are three main approaches applied in detecting XSS vulnerabilities, namely (i) static analysis, (ii) dynamic analysis, and (iii) hybrid analysis which based on static analysis and dynamic analysis, as described in [2].

The static analysis based approaches work with the source code of web applications and can detect the potential vulnerabilities automatically and efficiently. As it's difficult for static analysis to understand the execution logic of source code, the results may be false positives and false negatives. Gupta et al. [2] presents an excellent survey of the static analysis based approach for detecting XSS vulnerabilities and SQL vulnerabilities. Jovanovic et al. [3] proposes a method based on the static analysis to detect vulnerabilities in the web applications and the first open tool (Pixy) is implemented in their work. Shar et al. [4] [5] combine the technology of data mining with static code analysis to predict the XSS vulnerabilities and SQL vulnerabilities in web applications with static code attributes.

The dynamic analysis based approaches detect vulnerabilities with the information obtained during the application execution. The researches about detecting XSS vulnerabilities with dynamic analysis in web applications focus on fuzz testing, advanced taint tracking, as well as symbolic execution, as described in [6]. Most of the works about fuzz testing is based on attack vectors. First, build a basic attack vector repertory. Then fuzz the basic attack vectors in order to bypass filter in web servers. After that, the fuzzed attack vectors are used for test. Duchene et al. [7] detects the XSS Vulnerabilities using the model inference and evolutionary fuzzing. Based on the knowledge about the application behaviors obtained in model inference, the fuzz test inputs are generated with genetic algorithm. Z. Tang et al. [8] proposes a method based on a lexical mutation engine and the engine fuzz the seed (normal JavaScript code) to generate fuzz test inputs which are used for XSS test. The authors in [9] created an XSS attack vector repertory to detect the XSS vulnerabilities in web applications introduced by HTML5. The fuzz testing based on attack vectors can detect XSS vulnerabilities effectively with a low false negative. Well, the efficiency of this method is proportional to the size of attack vector repertory. With a wide variety of attack vectors and a lot of rules to fuzz the attack vectors, XSS attack vector repertory may have a large size, which may have thousands of XSS attack vectors and even more. Using a large-size attack vector repertory, XSS vulnerability detection may cost too much time, and the result may have a high false positive with a small amount of attack vectors.

In order to solve this problem, this paper proposes a method of XSS vulnerabilities detection which based on an optimized attack vector repertory. First, we propose a method to generate attack vector repertory automatically. This method develops an XSS attack vector grammar and builds attack vector pattern repertory, resource repertory and mutation rule repertory. With the attack vector repertory and resource repertory, the basic XSS attack vector repertory is generated. After that, mutation rules in mutation rule repertory are applied to the basic XSS attack vectors and the final XSS attack vector repertory is created. Second, we optimize the final XSS attack vector repertory using an optimization model which is built using a machine learning algorithm and trained with the historical test data produced by the XSS vulnerability detector. After that the optimized attack vector repertory is created and used for XSS vulnerability detection. Finally, we detect the XSS vulnerabilities dynamically with the optimized attack vector repertory.

This paper makes the following contributions.

- A method of generating XSS attack vector repertory. With this method, we can easily extend attack vector repertory by add attack vector patterns to attack vector pattern repertory and control the generated attack vector repertory by making mutation rule tables.
- A method of optimizing XSS attack vector repertory. Using the optimization model we can reduce the size of attack vector repertory and detect XSS vulnerabilities more efficiently.
- Base on this method, a prototype system is implemented, and tested with the real-world websites.

The rest of the paper is organized as follows. Section II introduces the background of this paper. In Section III, the XSS vulnerability detection framework is described. The method of XSS attack vector repertory generation is presented in Section IV, the optimization model is presented in Section V, and in Section VI, XSS vulnerability detection using the optimized attack vector repertory is described. Section VII shows the experiments to demonstrate the effectiveness of the method proposed in this paper and Section VIII concludes.

II. Background

XSS are often classified into three categories: reflected XSS, stored XSS and DOM based XSS. If there is a reflected XSS vulnerability in a website, attack vectors may be submitted to the web server. User's browser loads the response pages and executes the malicious scripts injected by the attacker. If there is a stored XSS vulnerability in a website, the injected malicious scripts are stored on the web server, and loaded and executed by the browser when user request the data or pages. DOM based XSS vulnerability attack does not rely on web servers and malicious scripts are executed directly in user's computers. The reflected XSS vulnerability attack is shown in Fig. 1.

In this paper, a machine learning classification algorithm is used to build an optimization model. The optimization model is trained with the historical testing data, and the model obtains knowledge about attack vector features from the training data. Based on the knowledge, optimization model optimizes the attack vector repertory and select those are used to detect XSS vulnerabilities in web applications finally. The implementation of the machine learning algorithms of Scikit-learn is used in our works, and which can be seen in [10].

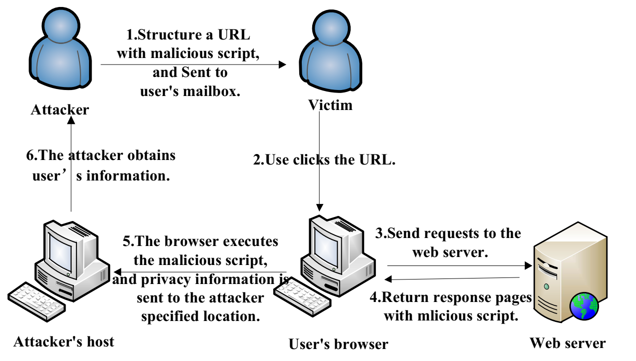
1.Structure a URL with malicious script, and Sent to user's mailbox.
Attacker
Victim
6.The attacker obtains user' s information.
2.Use clicks the URL.
5.The browser executes the malicious script, and privacy information is sent to the attacker specified location.
3.Send requests to the web server.
4.Return response pages with mlicious script.
Attacker's host
User's browser
Web server

Figure 1. Reflected XSS vulnerability attack.

The dynamic analysis is used in this paper. In order to detect XSS vulnerabilities, we perform injection tests for each injection point with XSS attack vector repertory after mutation and optimization. With the technology of web crawler, we get web pages and find injection points. The frame of web crawler of Scrapy is used in our works, and which can be seen in [11].

In general as described in [12], an attack vector depicts the approach that an adversary is used to assault a computer system or network. In this paper, an attack vector specifically refers to the user input with malicious scripts, and the malicious scripts in attack vector are called payload.

III. The Framework of XSS Vulnerability Detection

The XSS vulnerability detection contains 3 modules: attack vector repertory generation, attack vector repertory optimization and XSS vulnerability detection which use attack vectors as inputs. Fig. 2 shows the framework of XSS vulnerability detection in the paper.

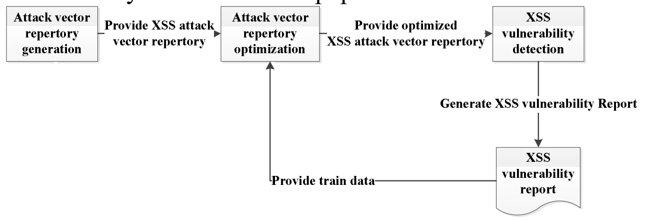
Attack vector repertory generation
Provide XSS attack vector repertory
Attack vector repertory optimization
Provide optimized XSS attack vector repertory
XSS vulnerability detection
Generate XSS vulnerability Report
XSS vulnerability report
Provide train data

Figure 2. The Framework of XSS vulnerability detection

IV. XSS Attack Vector Repertory Generation

In this section, we develop an XSS attack vector grammar to describe the whole process of XSS attack vector generation which contains basic XSS attack vector generation and final XSS attack vector generation. Following the XSS attack vector grammar, attack vector pattern repertory, resource repertory and mutation rule repertory are built. After that, we generate basic XSS attack vector repertory with attack vector pattern repertory and resource repertory, and finally, the final XSS attack vector repertory is generated by applying mutation rule to the basic XSS attack vectors.

A. *XSS Attack Vector Grammar*

Similar to the work of [7], an XSS attack vector grammar is developed in this paper, which has three parts: symbol set, attack vector pattern specification and mutation rule specification.

1) *Symbol Set*

Symbol set describes the symbols may appear in attack vectors, as shown in TABLE I. In this paper, payload has 3 categories: JS_Payload, URL_Payload and CSS_Payload.

TABLE I. SYMBOL SET OF XSS ATTACK VECTOR GRAMMAR

	Symbol	**Explanation**
1	Html_Tag	HTML tag
2	Html_Attr	HTML attribute
3	Html_Event	HTML event attribute
4	Html_AttrValue	Value of HTML attribute
5	Html_InsideTagText	Text inside HTML tag
6	Html_OusideTagText	Text outside HTML tag
7	Html_Comment	Text of the HTML comment
8	Grammar_Symbol	Symbols in HTML and JS, such as ">" and "<"
9	Separator	Separator among symbols, Such as space
10	Quote	Quote, ' or "
11	Fuzz	Fuzz text
12	Payload	Payload in XSS attack vectors
13	JS_Payload	Payload which contains JS directly
14	URL_Payload	Payload which contains JS in URL protocol
15	CSS_Payload	Payload which contains JS in CSS

2) *Attack Vector Pattern Specification*

Each attack vector pattern is representative of a class of XSS attack vectors. In this paper, the following 4 types of XSS attack vectors are considered, as shown in TABLE II.

The attack vector pattern specification describes the general form of attack vector patterns. After analyzing a lot of XSS attack vectors, we can find that an XSS attack vector is generally a HTML code fragment with malicious scripts, such as JS. So XSS attack vector pattern specification is similar to HTML syntax specification. This paper defines XSS attack vector pattern specification as follows.

[<] [Html_Tag] [Html_EventList] [Html_AttrList] [>] [Html_InsideTagText] [Html_TagList] [</] [Html_Tag] [>] [Html_outsideTagText] [<!--] [Html_Comment] [-->] [Html_TagList]

Explanations:

- Symbols, such as Html_AttrValue, Html_InsideTagText, Html_Comment and HtmlOutsideTagText, can be inserted with Payload.
- Html_TagList can be recursively nested within attack vector patterns..
- Html_EventList is a list of “Html_Event = Html_AttrValue”.
- Html_AttrList is a list of “Html_Attr = Html_AttrValue”.
- Fuzz and Separator can be among other symbols.

3) *Mutation Rule Specification*

The mutation rule specification describes mutation rules in mutation rule repertory which are applied to attack vectors in order to bypass filters. Table III shows the mutation rule specification.

B. *Three Repertories*

1) *Resource Repertory*

TABLE IV shows the content of resource repertory which is collected from Internet, such as [13]. There are three parts of resources in resource repertory.

The first part specifies the set of candidate chars of symbols in attack vectors. The second part provides the resources for basic XSS attack vector generation, for example, Html_URLAttr is the set of HTML attributes that can use URL protocol. The third part provides the resources for basic XSS attack vector mutation, for example, FuzzJS provides the set of sensitive chars for the mutation rule FuzzJS.

2) *Attack Vector Pattern Repertory*

Following the attack vector pattern specification of XSS attack vector grammar, we build attack vector pattern repertory. Table V shows attack vector patterns of the attack vectors in Table II.

For each attack vector pattern, we provide the set of candidate chars for some symbols, which are used in basic XSS attack vector generation.

Following attack vector pattern specification, users can add new attack vector patterns to the repertory, and it will generate new attack vectors in the end. So, we can continue to expand attack vector repertory in our work easily.

TABLE II. FOUR TYPES OF XSS ATTACK VECTORS IN THIS PAPER

	Type	**Example**	**Payload**
1	HTML tag “script”	<script> alert(1) </script>	JS_Payload:alert(1)
2	HTML event attribute	<body onload=alert(1)> </body>	JS_Payload:alert(1)
3	URL protocol	<a href=javascript:alert(1)> click_me </a>	URL_Payload:javascript:alert(1)
4	CSS	<body style = "background-image:url(javascript:alert(1))" > </body>	CSS_Payload:background-image:url(javascript:alert(1))

TABLE III. MUTATION RULE SPECIFICATION OF XSS ATTACK VECTOR GRAMMAR

	Type	**Explanation**
1	Name	The name of a mutation rule.
2	Type	The mutation rules are divided into 2 types: Fuzz and Encode. Fuzz means to insert fuzz text into the mutation objects and Encode means to encode the mutation objects.
3	Mutation Function	The general form of mutation function is "mutationFunction(kwargs, mutataionObject)". "kwargs" is the information about mutation object, such as the basic XSS attack vector, and "mutataionObject" is the mutation object.
4	Mutation objects	The operation objects of mutation rules, is a subset of symbol set, such as Html_AttrValue, Html_InsideTagText and Fuzz.

TABLE IV. RESOURCE REPERTORY

	Name	**Set of candidate chars**
1	Html_Tag	form, input, textarea, button, …
	Html_Attr	src, href, dynsrc, longdesc, lowsrc, action, …
	Html_Event	onclick, ondblclick, onmousedown, …
	Separator	Space, \n, \t, \r, …
	Quote	", '
	JS_Payload	"alert(1)", …
	URL_Payload	"javascript:JS_Payload", …
	CSS_Payload	"width:expression(JS_Payload);", …
2	Html_URLAttr	src, href, dynsrc, longdesc, lowsrc, action, …
	Html_CSSAttr	style, …
	Html_Attr2Tag	src: [img], longdesc: [img], ...
	Html_Event2Tag	onerror:[img], onclick:[img, button]， ...
3	FuzzJS	"/*XSS*/", "\t", "\r", "\n", …
	FuzzCSS	"/*XSS*/", "\\", "\0", …
	FuzzHtmlURLAttrValue	"\n", "\t", "\r", " ", …
	FuzzHtmlSpace	Space, \n, \t, \r, …
	FuzzBegin	"'>", "\">", "-->", "</script>", …

TABLE V. ATTACK VECTOR PATTERNS

	Attack vector patterns	**Set of candidate chars**
1	<Html_Tag> JS_Payload </Html_Tag>	Html_Tag:[script]
2	<Html_Tag Html_Event=JS_Payload> </Html_Tag>	Html_Event:[onmouseover, onloa, onclick,...]
3	<Html_Tag Html_Attr=URL_Payload> Html_InsideTagText </Html_Tag>	Html_Attr:[src, href,…]
4	<Html_Tag Html_Attr=CSS_Payload> </Html_Tag>	Html_Attr:[style]

TABLE VI. MUTATION RULES

	Name	**Type**	**Mutation objects**
1	FuzzJS	Fuzz	[JS_Payload]
2	FuzzCSS	Fuzz	[CSS_Payload]
3	FuzzHtmlSeparator	Fuzz	[Separator]
4	FuzzHtmlURLAttrValue	Fuzz	[Html_AttrValue]
5	FuzzBegin	Fuzz	[Fuzz]
6	FuzzBeginComment	Fuzz	[Fuzz]
7	FuzzBeginNoQuote	Fuzz	[Fuzz]
8	FuzzBeginSingleQuote	Fuzz	[Fuzz]
9	FuzzBeginDoubleQuote	Fuzz	[Fuzz]
10	EncodeLower2Upper	Encode	[Html_Tag, Html_Attr, Html_Event]
11	EncodeURL	Encode	[URL_Payload]
12	EncodeHtmlEntity	Encode	[Html_AttrValue, Html_InsideTagText]
13	EncodeHtmlDec	Encode	[Html_AttrValue, Html_InsideTagText]
14	EncodeHtmlHex	Encode	[Html_AttrValue, Html_InsideTagText]
15	EncodeJSUnicode	Encode	[JS_Payload]
16	EncodeCSSHex	Encode	[CSS_Payload]

3) Mutation Rule Repertory

After generating basic XSS attack vector repertory, the mutation rules are applied to the basic attack vectors and the final XSS attack vector repertory is created. The mutation rules used in the paper are shown in TABLE VI.

C. The Generation of XSS Attack Vector Repertory

The generation of XSS attack vector repertory contains two steps: basic XSS attack vector repertory generation and final XSS attack vector repertory generation. Fig. 3 shows the process of XSS attack vector repertory generation.

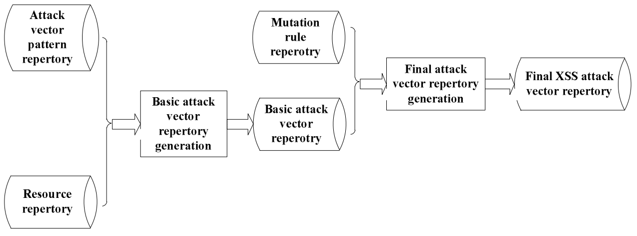
Attack vector pattern repertory
Resource repertory
Basic attack vector repertory generation
Mutation rule reperotry
Basic attack vector reperotry
Final attack vector repertory generation
Final XSS attack vector repertory

Figure 3. The generation of XSS attack vector repertory.

1) Basic XSS Attack Vector Repertory Generation

In order to generate basic XSS attack vector repertory, we have to determine the value of the symbols in each attack vector pattern with the given set of candidate chars of symbols and resource repertory. If the set of candidate chars of a symbol is given, we choose one randomly as the value of the symbol. If the set of candidate chars of a symbol is not given, we determine its value with the value of other symbols and resource repertory. For example, the resources may be used are Html_Attr2Tag and Html_Event2Tag. Html_Attr2Tag describes the set of HTML tags for each HTML attribute, and Hml_Event2Tag describes the set of HTML tags for each HTML event attribute. If the value of one symbol can't be determined, we choose one randomly from resource repertory.

The data structure (Python) of basic XSS attack vector is defined, as shown in Fig. 4.

```
Class BasicXSSAttackVector(object):
    def __init__(self, vectorType=[], vectorPatter =[],
    vectorValue=[]):
        self.vectorType = vectorType
        #The types of basic XSS attack vector.
        self.vectorPattern = vectorPattern
        #The list of symbols in basic attack vector pattern.
        self.vectorValue = vectorValue
        #The list of values of the symbol list in basic attack vector
        #pattern.
```

Figure 4. Data structure (Python) of basic XSS attack vector.

There is an example as below:

vectorType:["HtmlText","URL_Payload"]

vectorPattern:['<', 'Html_Tag', 'Html_Attr', '=', Html_'Value', '>', 'Html_InsideTagText', '</', 'Html_Tag', '>']

vectorValue:['<', 'a', 'href', '=', 'URL_Payload', '>', 'Click', '</', a, '>']

2) Final XSS Attack Vector Repertory Generation

We generate final XSS attack vector repertory by applying mutation rules to each basic XSS attack vector, and the mutation rules used in this paper is listed in TABLE VI.

The mutation rule tables which define the mutation rules applied to the basic XSS attack vectors should be made. We can made mutation rule tables for one attack vector pattern or for all basic XSS attack vectors.

The process of applying mutation rules to a basic XSS attack vector is described in Fig. 5. In the Pretreatment and Final treatment, we obtain the list of sensitive strings and the list of sensitive chars in the final XSS attack vector.

TABLE VII lists some XSS attack vectors in the final XSS attack vector repertory.

The data structure (Python) of final XSS attack vector is defined, as shown in Fig. 6.

And there is an example as below.

vectorType:["HtmlText","URL_Payload","HtmlAttrDoubleQuote"]

vectorTypeInfo:["a","href","a","<=/>":%"]

vectorMutationRlues:["FuzzBeginDoubleQuote","EncodeLower2Upper","EncodeURL"]

vector:"x"><a Href=jaVasCript:alert%283%29>click</a>"

TABLE VII. FINAL XSS ATTACK VECTORS

	Attack vectors
1	x">x><script > prompt(3) </SCripT >
2	x'><a HrEF = jaVAscRIPt:alert(3) > Click </a >
3	x--><AudiO oNerROr = write(3) sRc = x > Click </aUDIo >
4	<Body sTYlE = BackgrOUnD-Image:uRl(JavAscripT:write(3)) > Click </Body >
5	x><buTtOn onMouSeoUT = \u0077\u0072\u0069\u0074\u0065(3) > Click </buTtOn >
6	<iMG lOwSrC = jaVasCrIPT:prompt%283%29 />
7	x>x'>x">x--><A hREf = JAVAscrIPT:alert(3) > Click </a >
8	</tiTle><scRipt>alert(1)</scRipt>

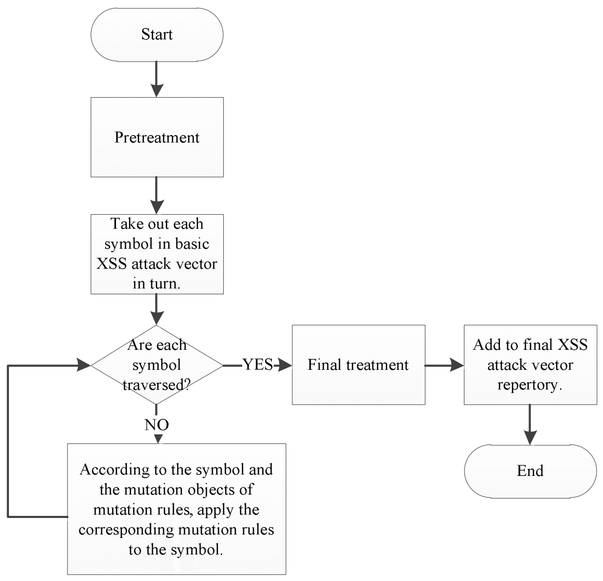
Start
Pretreatment
Take out each symbol in basic XSS attack vector in turn.
Are each symbol traversed?
YES
NO
Final treatment
Add to final XSS attack vector repertory.
End
According to the symbol and the mutation objects of mutation rules, apply the corresponding mutation rules to the symbol.

Figure 5. Process of applying mutation rules to a basic XSS attack vector.

```
class FinalXSSAttackVector(object):
   def __init__(self, vectorType = [], vectorTypeInfo = [],
vectorMutationRules=[], vector = ""):
      self.vectorType = vectorType
      #The types of final XSS attack vector.
      self.vectorTypeInfo = vectorTypeInfo
      #The type information about the final XSS attack vector,
      #including sensitive strings and sensitive chars.
      self.vectorMutationRules = vectorMutationRules
      #The mutation rules that are applied to this attack vector.
      self.vector = vector
      #The final XSS attack vector.
```

Figure 6. Data structure (Python) of final XSS attack vector.

V. XSS Attack Vector Repertory Optimization

In order to reduce the size of final XSS attack vector repertory, an optimization model is built with a machine learning classification algorithm. We extract four types of attributes of XSS attack vectors in this paper: attack vector types, mutation rules, sensitive strings and sensitive chars.

We have try seven machine learning classifiers to build the optimization model, and the XSS vulnerability detector with optimization model is tested with real-word websites which is present in Section VII.

A. *The Types of Attack Vector*

After study the code of w3af, see detail in [14], this paper classifies the attack vectors into different types according to injection places, as shown in TABLE VIII.

Each attack vector may have more than one type, meaning this attack vector can be executed successfully in more than one injection places.

The initial value of the types of attack vector is the types of attack vector pattern which is set by user. And new types of attack vector are added by mutation rules, such as FuzzBegin.

B. *Other Attributes*

The mutation rules are already introduced before. The sensitive strings are the Html_Tag, Html_Attr and Html_EventAttr used in attack vectors. The sensitive chars are the chars introduced by HTML, JS and mutation rules. In our work, the sensitive chars considered are shown in TABLE IX.

TABLE VIII. THE TYPES OF ATTACK VECTOR

Type		**Example**
Html	HtmlText	<script>alert(1)</script>
	HtmlComment	--><script>alert(1)</scirpt>
	HtmlAttrNoQuote	><script>alert(1)</script>
	HtmlAttrSingleQuote	'><script>alert(1)</script>
	HtmlAttrDoubleQuote	"><script>alert(1)</script>
Script	ScriptText	</script><script>alert(1)</script>
	ScriptLineComment	\nalert(1);
	ScriptMultiComment	*/alert(1);
	ScriptNoQuote	;alert(1);
	ScriptSingleQuote	';alert(1);
	ScriptDoubleQuote	";alert(1);
OtherTag	Title	</title><script>alert(1)</script>
	Style	</style><script>alert(1)</script>

TABLE IX. SENSITIVE CHARS

Sensitive chars
<\-=/>"'()[]{}:%&#;

VI. XSS Vulnerability Detection

In this Section, we introduce the identification of injection points and describe how to judge if there are XSS vulnerabilities or not.

A. *Identification of the Injection Points*

After creating the optimized XSS attack vector repertory, we identify the injection points in the pages crawled by web crawler. Four injection points are considered in this paper.

1) The Header of HTTP protocol. The XSS vulnerability detector sets the value of Reffer of Header to an XSS attack vector, and sends a request to the website.

2) Form inputs. Each input of forms is filled with an XSS attack vector in turns, and the XSS vulnerability detector sends requests to the website.

3) URL parameters. Each URL parameter is replaced with an XSS attack vector in turns, and requests are sent to the website by XSS vulnerability detector.

4) End of URL. The detector adds an XSS attack vector to the end of the URL, and sends a request to the website.

B. XSS Vulnerability Judgement

The response pages from website are parsed to determine the injection places of XSS attack vectors. In this paper, XSS vulnerabilities are detected if and only if the injection place and the types of the XSS attack vector are matched. And only if the injection place is one of types of the XSS attack vector, a match happens.

VII. EXPERIMENTS

Based on this method, an XSS vulnerability detector is implemented. Finally, there are 14 attack vector patterns in attack vector pattern repertory, and each attack vector pattern generates at most 5 basic XSS attack vectors randomly. 16 mutation rules in mutation rule repertory are used to mutate basic XSS attack vectors. In order to build a suitable model, the optimization model is built using seven different machine learning classifiers provided by Scikit-learn in order.

To evaluate this method of XSS vulnerability detection, experiments are conducted in this paper. The experiments perform on 50 websites that are chosen from Xssed in chronological order, see detail in [15]. First, with 92 mutation rule tables, 1105 XSS attack vectors in final XSS attack vector repertory are generated. Without the optimization model, we use the 1105 XSS attack vectors to detect the first 20 websites and generate the first XSS vulnerability report which can be used as training data to the optimization model. Second, we detect the 50 websites eight times, seven with optimization model built by different machine learning classifiers and one without optimization model. TABLE X shows the parameters of XSS vulnerability detector.

The machine learning classifiers chosen in this paper and the experimental results are shown in TABLE XI, Fig. 7 and Fig. 8. Considering the numbers of attack vectors have a great influence on the results and the number of attack vectors generated without optimization model is much more than others, the detection time for the detector without optimization model is three times than others.

According to TABLE XI, Fig. 7 and Fig. 8, we draw the following conclusions.

1) The XSS vulnerability detector can detect common XSS vulnerabilities effectively.

2) The optimization model can reduce more than 10 times of the size of XSS attack vector repertory, and the XSS vulnerability detector using optimization model can detect XSS vulnerabilities more efficiently.

TABLE X. PARAMETERS OF XSS VULNERABILITY DETECTOR

Parameter	Explanation	First	Second
num(AVP)	The number of attack vector patterns in attack vector pattern repertory.	14	14
num(OAVP)	The maximum number of attack vectors generated by one attack vector pattern.	5	5
num(BAV)	The number of basic attack vectors in the basic XSS attack vector repertory.	36	37
num(MR)	The number of mutation rules in the mutation rule repertory.	16	16
num(MRT)	The number of mutation rule tables.	92	92
num(FAV)	The number of final attack vectors in the final XSS attack vector repertory.	1105	1122

TABLE XI. THE RESULT OF EXPERIMENTS

Optimization model	Attack vectors	XSS Websites(50)	XSS vulnerabilities	Time for Each website (s)
None	1122	20	89	3000
KnighborsClassifier	46	24	848	1000
GradientBoostingClassifier	49	24	397	1000
SGDClassifier	134	24	283	1000
DecisionTreeClassifier	68	23	717	1000
LinearSVCClassifier	59	21	436	1000
GaussianNBClassifier	113	21	183	1000
RandomForestClassifier	40	20	428	1000

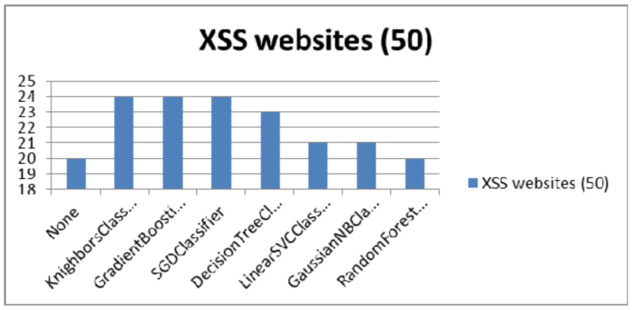

Figure 7. The Number of XSS websites detected

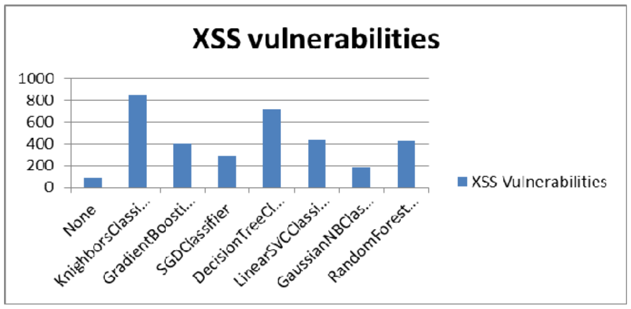

Figure 8. The Number of XSS vulnerabilities detected

VIII. CONCLUSIONS AND FUTURE WORKS

In order to detect XSS vulnerabilities, this paper proposes a method using optimized attack vector repertory. This method generates XSS attack vectors automatically, and uses an optimization model to reduce the size of attack vector repertory. After that, we detect XSS vulnerabilities with XSS attack vectors dynamically. Experimental results show that the method makes a good performance in optimizing attack vector repertory and detecting XSS vulnerabilities in web applications. In Future, we will train the optimization model using more training data and optimize the parameters of the optimization model using more testing data.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant No.61402437.

REFERENCES

[1] J. Williams and D. Wichers, "OWASP top 10–2013," OWASP Foundation, 2013.

[2] Gupta, M.K., Govil, M.C., Singh, G., Static Analysis Approaches to Detect SQL Injection and Cross Site Scripting Vulnerabilities in web Applications: A Survey, in: Recent Advances and Innovations in Engineering (ICRAIE), 2014, 9-11 May 2014.

[3] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A Static Analysis Tool for Detecting web Application Vulnerabilities," Proc. of the IEEE Symposium on Security and Privacy, May 2006.

[4] L.K. Shar, H.B.K. Tan, Mining input sanitization patterns for predicting SQL injection and cross site scripting vulnerabilities, in Proceedings–34th International Conference on Software Engineering, ICSE 2012, 2012, pp. 1293–1296.

[5] L.K. Shar, H.B.K. Tan, Predicting common web application vulnerabilities from input validation and sanitization code patterns, in: Proc. 27th IEEE/ACM Int. Conf. Autom. Softw. Eng. – ASE 2012, 2012, p. 310.

[6] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, "Modeling and Discovering Vulnerabilities with Code Property Graphs," in Proceedings of the 35th IEEE Symposium on Security and Privacy (S&P), 2014.

[7] F. Duchene, R. Groz, S. Rawat, J.-L. Richier, XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing, in: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation,2012, no. Itea 2, pp. 815–817.

[8] Z. Tang, H. Zhu, Z. Cao, S. Zhao, L-WMxD: Lexical based webmail XSS Discoverer, in: 2011 IEEE Conference on Computer Communications Workshops INFOCOM WKSHPS,2011, pp. 976–981.

[9] Yan Zhang, Xin Wang, Peng Wang, Liangkun Liu, Detecting Cross Site Scripting Vulnerabilities Introduced by HTML5, in: 2014 11th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2014,pp. 319–323.

[10] Scikit-leawrn [EB/OL]. http://scikit-learn.org.

[11] Scrapy [EB/OL]. http://scrapy.org/.

[12] Attack vector [EB/OL]. http://www.pcmag.com/encyclopedia/term/57711/attack-vector.

[13] W3school [EB/OL]. http://www.w3school.com.cn/.

[14] W3af [EB/OL]. http://w3af.org/.

[15] Xss Archive [EB/OL]. http://www.xssed.com/archive.