ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, Vol 8(27), DOI: 10.17485/ijst/2015/v8i27/87068, October 2015 ISSN (Online) : 0974-5645

Hadoop-based Crawling and Detection of New HTML5 Vulnerabilities on Public Institutions' Web Sites

In-A Kim[1], Kyu-Hyun Cho[1], Hyung-Jun Yim[1], Hwan-Kuk Kim[2] and Kyu-Chul Lee[1*]

[1]Department of Computer Engineering, Chungnam National University, 220 Gung-Dong, Yuseong-Gu, Daejeon, Korea; dodary0214@gmail.com, keicoon15@gmail.com, hyungjun25@gmail.com, kclee@cnu.ac.kr
[2]Korea Internet and Security Agency, 135, Jungdae-ro, Songpa-gu, Seoul, Korea; rinyfeel@kisa.or.kr

Abstract

HTML5 is a recent version of HTML, a programming language for web documents. It was developed to solve the problems of previous HTML versions. However, the new elements and functions of HTML5 have expanded the range of attacks that third parties can abuse. This is especially the case for public institutions which apply HTML5 in their web sites, and means that their web sites are more vulnerable to these attacks than other private websites. Public institutions' web sites consist of a larger number of web documents than other general web sites because the web sites provide information regarding policies, voting, and other events, and are connected with subordinate institutions. In this paper, because public institutions web sites consist of a large number of web documents, we used Hadoop, which is an open-source framework for distributed storage and processing. HTML5 vulnerabilities detection was processed for a large number of web documents by using distributed parallel processing. By applying distributed parallel processing for the crawling and detecting processes, we were able to improve the performance of the crawling and detecting processes for a large number of web documents connected to public institutions web sites.

Keywords: Crawling, Distributed Parallel Processing, Hadoop, HTML5 Vulnerability

1. Introduction

The previous version of the Hyper Text Markup Language (HTML) had many limitations in file upload / download, audio/video playback and implementing graphics processing, and used a variety of non-standard plug-ins such as Active X, Flash, and Silver light essentially for these purposes[1,2]. To resolve the issues with the previous HTML, HTML5, which is the next generation web standard of the World Wide Web (WWW), promises interoperability between platforms and devices and adds a lot of changes in the existing HTML specification. Among the items added in HTML5, multimedia elements make audio/video playback possible without installing non-standard plug-ins[2]. In addition to device access to features such as Geolocation, there are a lot of other changes, including Web storage, and Web messaging, in HTML5[3].

Because of the advantages that HTML5 has, as an increasing number of web browsers such as Chrome, Safari and OS platforms have become HTML5 compliant, public institutions around the world has also increased the proportion of HTML5 application sites. The UK government web sites, GOV.UK can be mentioned as a representative case of public websites employing HTML5. By applying HTML5 GOV.UK helps audiences to acquire information by providing media content without requiring other plug-ins, and enhances accessibility from a variety of devices and browsers[4,5].

However, new elements of HTML5 have expanded the range of possible attacks by third parties and new web security vulnerabilities have been discovered in web applications and sites that support HTML5. In cases of public institutions web sites, there is a higher probability of vulnerabilities with HTML5 than with other general

* Author for correspondence

web sites. Since public institutions web sites provide information and services to a public audience, all potential vulnerabilities of public institutions web sites should be prevented in advance. Since public organizations are in charge of providing and releasing information regarding policies and events, public organizations web sites need to figure a way to detect and prevent new vulnerabilities in the HTML5 application environment[6].

To detect HTML5 vulnerabilities a process of crawling and detecting the content of web documents is required which can respond from a web server. Since web sites consist of web documents connected with each other, and web documents have a relationship with other web documents by Hyperlink via a URI (Uniform Resource Identifier), all of the web documents connected to the top-level web document of web sites is the targets of crawling and detecting.

Public institutions' web sites consist of a large number of web documents providing information regarding policies, voting, and other events. In particular, the web sites have a larger number of web documents than other general web sites because of connections with subordinate institutions. In addition, public institutions' web sites contain web documents which have changing content in real-time, such as Tweets, and web documents regarding policies and events also have changes of content in the short term.

Since crawling and detecting the large number of web documents of public institutions' web sites is difficult to process by a single machine, we used Hadoop, which is a distributed parallel system for crawling web documents by multiple machines. In addition, because the quantity of crawled web documents was large, the process of detecting HTML5 vulnerabilities was also conducted on a distributed parallel system, which allowed the stable crawling and detection of a large number of web documents.

In addition, in this paper, after extracting the external links of web documents, the crawling process was repeated for the web documents of the extracted links to achieve crawling depth. The detection process was started right after the crawling process of web documents at one crawling depth. Using these processes can reduce the time gap of web documents which is changing dynamically.

In this study, we designed and developed a system which crawls a large number of web documents and detects potential HTML5 vulnerabilities based on distributed parallel process. The term detection in this paper means determining the probability of potential vulnerability in web documents.

The remainder of this paper is organized as follows. Section 2 introduces the background of the new vulnerabilities of HTML5, Apache Nutch, and related works. In Section 3, we describe the crawling process of HTML documents and the distinctive method of HTML5 documents. In Section 4, we provide methods of detecting HTML5 vulnerabilities. Lastly, we conclude with a summary of the method we propose.

2. Background

2.1 New Vulnerabilities of HTML5

HTML was first created by Tim Berners-Lee who was a physicist at the European Particle Physics Laboratory (CERN) in 1991, HTML2.0, HTML4.01, and other versions of HTML have been released as a W3C Recommendation. HTML has strong advantages in representing and structuring information, but it is weak in compatibility between platforms and the need to use other non-standard plug-ins to express audio/video/graphics. Because of these problems, W3C introduced HTML5, which added to and modified specifications of the existing HTML, and later confirmed HTML5 as its final recommendation. HTML5 added Multimedia elements, Semantic elements, Geolocation, Web Socket, Web Storage, and so on.

As a lot of specifications were added and modified from the existing HTML, new vulnerabilities in HTML5 have been discovered in web applications and sites that apply HTML5. An open source application security project OWASP (Open Web Application Security Project) selected and released HTML5 web application security vulnerabilities in 2015[7,8].

A Click Jacking attack is an attack that induces users to click on another web document which an attacker inserts regardless of the user's intention[9,10]. The Tag-based Cross-Site Scripting (XSS) attack is an attack to inject malicious script on web documents to seize the user's information or perform unusual features[11]. In addition, attackers can seize the information and control the user's browser by using Web Socket or an attacker may attempt other attacks by using Geolocation API, Web Messaging, and We Worker.

In this paper, we classify the HTML5 vulnerabilities as vulnerabilities based on tag and attribute and based on Java Script. Based on the classified vulnerabilities, we apply different detection methods for each of the classified vulnerabilities.

2.2 Apache Nutch[12]

Nutch is an open source web search engine package which aims to efficiently index the World Wide Web, such as a commercial search service. Nutch crawls a large amount of web documents based on Hadoop, which is a distributed parallel processing system, as well as crawling and indexing the small capacity web.

Nutch was first started by Doug Cutting who is a founder of Lucene and Hadoop, and Mike Capper Relais, and was implemented based on MapReduce and Hadoop Distributed File System (HDFS). Later, Hadoop was separated from the Nutch project, Nutch was included as a subproject of Lucene and separated as an independent project.

Nutch has a structure for handling a large number of web documents based on Hadoop. Hadoop is an open source framework, which supports a distributed application running on the large data processing clusters, and consists of MapReduce, HDFS, and OS-level abstractions. The primary feature of Hadoop is that the data operation is made in a computer while the actual data is located by transmitting the code for processing data to the node. Using this method, Hadoop performs data processing and reduces the transfer time of a large amount of data[13].

Nutch consists of a Searcher for retrieving the information from the query, an Indexer for generating an inverted index, a Database for storing the content of web documents, and a Fetcher for extracting hyperlinks from web documents.

In this paper, we perform the crawling process of a web document based on Nutch, which is a web document crawling project based on a distributed parallel processing for the crawling and detection of large volumes of web documents.

2.3 Related Works

As new vulnerabilities in the HTML5-based web applications and websites have been discovered, institutions and companies such as OWASP are conducting research focusing on new HTML5 security vulnerability analysis and detection. In[14,15] the researchers introduced methods for detecting the new HTML5 security vulnerabilities.

The research[14] automatically analyzed DOM based XSS vulnerabilities for a large number of web documents. The study increased the accuracy of the vulnerability detection by analyzing Java Script code at the bytes level, and HTML5's new elements based XSS vulnerability was also included. In research[15], the study automatically analyzed potential XSS vulnerabilities within web documents by constructing a repository of XSS vulnerability keywords based on the new HTML5 elements.

It automatically analyzed the DOM based XSS vulnerabilities, including new HTML5 elements intended for a large amount of web documents. However, because the study was aimed at detecting a vulnerability using the cumulative crawling process of multiple web sites, the stability of the process of crawling and analyzing a large number of web documents was not taken into account. The studies reported in[14,15] suggested methods for detecting the new HTML5 security vulnerabilities, but did not mention a method to stably crawl and process large amounts of web documents for detection.

In order to solve the common problem of these studies, in this paper, we performed crawling and vulnerability detection based on a distributed parallel processing, to increase the efficiency of processing the web documents and reduce the relative speed of a single machine process.

3. Crawling of HTML5 Web Documents based on Hadoop

3.1 Structure of Web Documents to be Detected

General web sites are connected with other web documents through a hyperlink. Typically, web sites have the most top level documents (top-level document or Root URL), and a top-level document contains several hyperlinks of one or more sub-documents, or nothing. One or more sub-documents attached to the top-level document will include hyperlinks connected to other sub-documents, and the connection relationship repeats between web documents. Therefore, web sites can be represented as a tree structure having a root document, a parent node, and a child node.

Public institutions' web sites consist of web documents regarding the policies and events of the government or international organizations, and have different features than general web sites, because the subordinate web sites of government, and web sites of countries joined in international organizations are all connected. Furthermore, HTML documents such as HTML4.01, XHTML1.1, and HTML5 can be contained among web documents of public institutions web sites and these HTML documents can be linked to other HTML5 documents. Since the top-level document to be detected may be another HTML document, and not HTML5, other kinds of HTML documents are also targets when crawling all the HTML5 documents attached to public institution websites.

In the case of public institutions web sites, as the stage of the tree increases, the number of web documents sharply increases in one crawl depth because web documents can include at least one hyperlink and all kinds of HTML documents can be the targets for crawling.

3.2 Crawling Process of Web Documents based on Hadoop

In this paper, we perform the crawling process of a web document prior to detection. The web document crawling process is divided into five steps, Inject, Generate, Fetch, Parse, Update, and operates based on MapReduce, which uses distributed parallel processing to crawl a large amount of web documents.

As shown in Figure 1, Inject stores the URL of the top-level document to storage. Based on this, the Generate process generates a list of web documents to be crawled from the Web server. Upon storing the URL of the top-level document, it generates a URL as the crawling target list.

When the information of multiple web documents is stored while crawling depth increases, it creates URLs of multiple web documents as the crawling target list. The Fetch process crawls the contents of the document after visiting web documents of the generated URL list, and the Parse process extracts hyperlinks to the outside web documents that are included in the content of the crawled document. After completing the Parse process, the Update process is used to update the information of the newly added and changed web documents.

The Inject process is carried out only early in the crawling process. Each time the crawling depth increases, the sequence of processes except for the Inject process are iterated. After the Fetch and Parse processes in one depth are completed, it saves the hyperlinks of the newly extracted external documents and creates a new crawling target list. If there are no more web documents to be crawled because the web documents crawled in the last run do not include a hyperlink, or if it reaches the designated crawling depth, the entire crawling process of web documents ends.

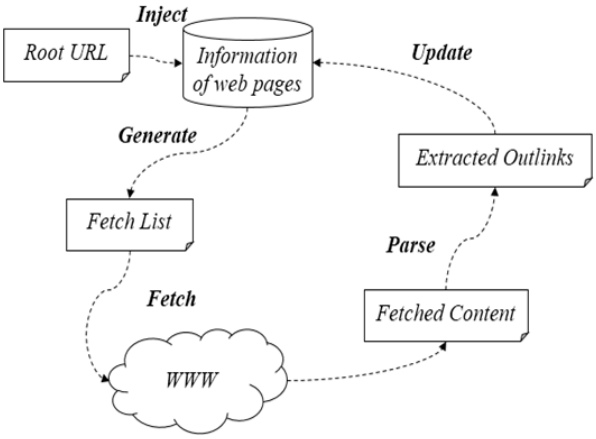

Figure 1. Crawling process of web documents based on Hadoop.

3.3 Distinctive Process of HTML5 Web Documents

In this paper, we implement a process of detecting vulnerabilities by targeting HTML5 web documents to detect HTML5 vulnerabilities, and perform a process of selecting HTML5 documents with the crawling process.

HTML documents are composed of several types, such as HTML4.01, XHTML1.1. Each type of Web document refers to a DTD (Document Type Definition) for the document type setting. DTD is the most basic specification for HTML documents and defines a structure and format for the exact status of documents. An HTML document declares a 'DOCTYPE' that is a part of a DTD reference in the top of the document, and browsers can process web documents quickly and accurately using DOCTYPE.

Table 1 shows an example of a DTD reference corresponding to the type of HTML document. For the first HTML4.01, the Strict document begins with the declaration '!DOCTYPE' and references strict.dtd, which is defined by the W3C. The XHTML1.1 document references xhtml11.dtd, which is defined by the W3C. However, in contrast to the other types of documents in Table 1 (HTML4.01, XHTML1.1), the HTML5 document has a structure ending with a <!DOCTYPE html> without a separate DTD reference.

Table 1. DTD Reference depending on the kind of web documents

Version	Example of DTD Reference
HTML5	<!DOCTYPE html>
HTML4.01	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
XHTML1.1	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

In this paper, using the features of HTML documents, we determine whether the DTD reference exists in the DOCTYPE of web documents, and select the HTML5 document. As shown in Table 2, we categorize and define crawled web documents into two kinds of document types inside the system.

If a separate DTD reference in the DOCTYPE declaration of crawled web documents is discovered, the document will be considered to be a non HTML5 document, and defines the document type as 'text / html'. If there is no DTD reference in the document, it determines the document to be an HTML5 web document and defines the document type definition as 'text / html5'.

Table 2. Content type of web documents

Contents	Explanation
text/html5	HTML5 Web Document
text/html	Other HTML Web Document (Ex. HTML4.01, XHTML1.1)

4. Detection of HTML5 Vulnerabilities Based on Hadoop

4.1 Definition of HTML5 Vulnerabilities to be Detected

In this paper, we selected the HTML5 vulnerabilities shown in Table 3 and Table 4 as those to be detected, among the HTML5 vulnerabilities that OWASP selected. The vulnerabilities to be detected were selected based on HTML5 attacks which are most commonly used, such as ClickJacking, Cross-site Scripting, DoS, Scan, Geolocation, Web Socket, and Web Worker.

Selected vulnerabilities were classified as tag and attribute vulnerability and JavaScript vulnerability based on the characteristics of its entry. Tag and attribute vulnerability means the vulnerability to attempted attacks based on the tag and attribute included in the HTML content of a web document, and we defined the core keywords for each of the vulnerabilities, as shown in Table 3. Java Script vulnerability means the vulnerability to attempted attack based on web-based script codes contained in a document, and we defined the core keywords for each vulnerability, as shown in Table 4.

Table 4. Javascript vulnerabilities

Types of Vulnerabilities		Core Keywords
DoS Attack - DoS	HashDoS	set Interval, open, send, ActiveXObject, XML-HTTP, XML Http Request
Scan Attack - SCN	Network Scan	open, ActiveXObject, XMLHTTP, XML Http Request, Date, ready State
Geolocation - GEO		coords, get Current Position
Web Socket - WSC		parse, eval, WebSocket, JSON, send
Web Worker DDoS - WWD		post Message, Worker, XML Http Request, open, send

In this paper, we defined the vulnerability patterns to be used in the vulnerability detection process based on the core keywords for each vulnerability, as presented in Table 5. A total of 24 vulnerabilities were defined by considering the total number of occurrences of a combination of the core keywords, and they included tag and attribute vulnerabilities and JavaScript vulnerability.

Table 3. Tag and attribute vulnerabilities

Types of Vulnerabilities	Core Keywords			
	Attribute1	Attribute2	Tag1	Tag2
Click Jacking Attack	Sandbox		iFrame	
Cross-site Scripting Attack	Autofocus	onfocus, onblur	input, select, text area, button, keygen	
	Formation		button, input	
	Onerror		video, audio	source
	Poster		video	
	Href		math	
	Srcdoc		iFrame	

The vulnerability patterns are composed of a vulnerability number, a vulnerability name, and keywords. P refers to the vulnerability number, from P1 up to P18 means tag and attribute vulnerability, from P19 up to P23 means JavaScript vulnerability. The vulnerability name is an abbreviation of the full name for each vulnerability and is limited to three letters.

The keywords of tag and attribute vulnerabilities and Java Script vulnerability have different pattern definitions. Tag and attribute vulnerabilities use a keyword having -E, -A and numbers. -E represents a tag (element), -A represents an attribute (attribute). Numbers suffixed to –E, -A indicate the depth of the tags and attributes. If this number is the same as the corresponding tags and attributes, it is considered to belong to the same level. In the case of JavaScript vulnerability, since the depth between keywords does not exist, unlike tag and attribute vulnerability, keywords are listed in order from the rarest keywords.

4.2 Process of Detecting HTML5 Vulnerabilities

The system performs preforms the process of crawling the web documents of public institutions' websites before starting security vulnerability detection. Security vulnerability detection is carried out in the Parse process for web documents in the crawling process. The Parse process is the process that is performed immediately after crawling the web documents of the crawling target list that was extracted in one depth. When the Parse process is finished, it increases the crawling depth to extract a new crawling target list, and performs crawling and detection.

Public organization websites contain many changeable articles such as real-time Tweets, and have a characteristic of introducing changes in a short time period, such as policy and event-related documents. In consideration of these characteristics, the method performs the detection process immediately after the crawling process in one depth, to reduce the time difference in the crawling and detection.

In addition, the crawling depth is increased, and the amount of crawled web documents increases exponentially, which means that the amount of vulnerability to be detected also increases. In order to detect security vulnerabilities for a large number of web documents, the detection process should operate based on MapReduce, as shown in Figure 2. To keep a large number of web documents discrete, web documents are distributed to mappers divided by a 'Host'. This means

Table 5. Vulnerability patterns to be detected

Number	Vulnerabilities	Keyword Patterns	Number	Vulnerabilities	Keyword Patterns
P1	JAK	-E0 iFrame -A0 sandbox	P13	XSS	-E0 input -A0 formaction
P2	XSS	-E0 input -A0 autofocus -A0 onfocus	P14	XSS	-E0 video -E1 source -A1 onerror
P3	XSS	-E0 input -A0 autofocus -A0 onblur	P15	XSS	-E0 audio -E1 source -A1 onerror
P4	XSS	-E0 select -A0 autofocus -A0 onfocus	P16	XSS	-E0 video -A0 poster
P5	XSS	-E0 select -A0 autofocus -A0 onblur	P17	XSS	-E0 math -A0 href
P6	XSS	-E0 textarea -A0 autofocus -A0 onfocus	P18	XSS	-E0 iFrame -A0 srcdoc
P7	XSS	-E0 textarea -A0 autofocus -A0 onblur	P19	DOS	setInterval send open ActiveXObject XML-HTTP XMLHttpRequest
P8	XSS	-E0 button -A0 autofocus -A0 onfocus	P20	SCN	readyState Data send open ActiveXObject XMLHTTP XMLHttpRequest
P9	XSS	-E0 button -A0 autofocus -A0 onblur	P21	GEO	coords getCurrentPosition
P10	XSS	-E0 keygen -A0 autofocus -A0 onfocus	P22	WSC	WebSocket parse eval JSON send
P11	XSS	-E0 keygen -A0 autofocus -A0 onblur	P23	WWD	postMessage Worker send open XML-HttpRequest
P12	XSS	-E0 button -A0 formaction			

that each mapper is treated with a type of web documents on the same host.

Mappers perform the process shown in Figure 3, and an HTML5 selecting process is performed before the detection process for an HTML document divided into the same host. If an input document is not an HTML5 document, the document is transferred to the hyperlink extraction process without the vulnerability detection. If the input document is an HTML5 document, it performs vulnerability detection.

Vulnerability detection is classified as either tag and attribute vulnerabilities detection or JavaScript vulnerability detection depending on the characteristics of the vulnerabilities. To do this, it represents the content of the document as a DOM tree. The system extracts the contents of all <script> tags within the document from the DOM tree and passes them to Java Script vulnerability detection. It transmits the other information of the tree to the tag and attributes vulnerability detection.

If a vulnerability is detected during the vulnerability detection process, it stores the information such as URL, the vulnerability number, the vulnerability position within the document where the vulnerability was discovered, as the intermediate results. When reducers that received the intermediate result file from mappers store the end results, the detection process is completed and the hyperlink extraction process is performed.

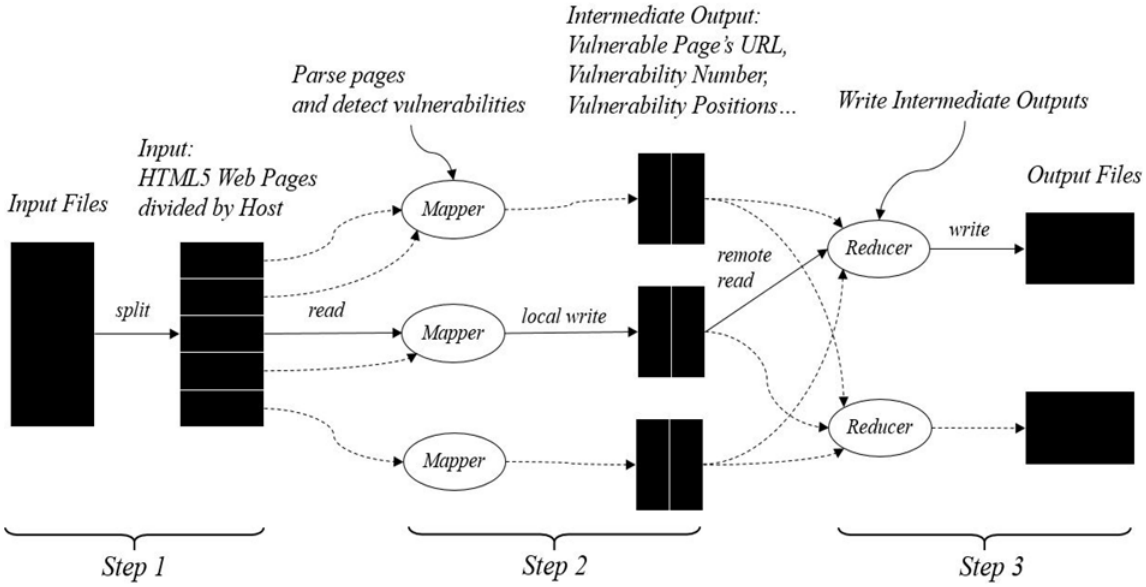

Figure 2. Detection process of HTML5 vulnerability based on map reduce.

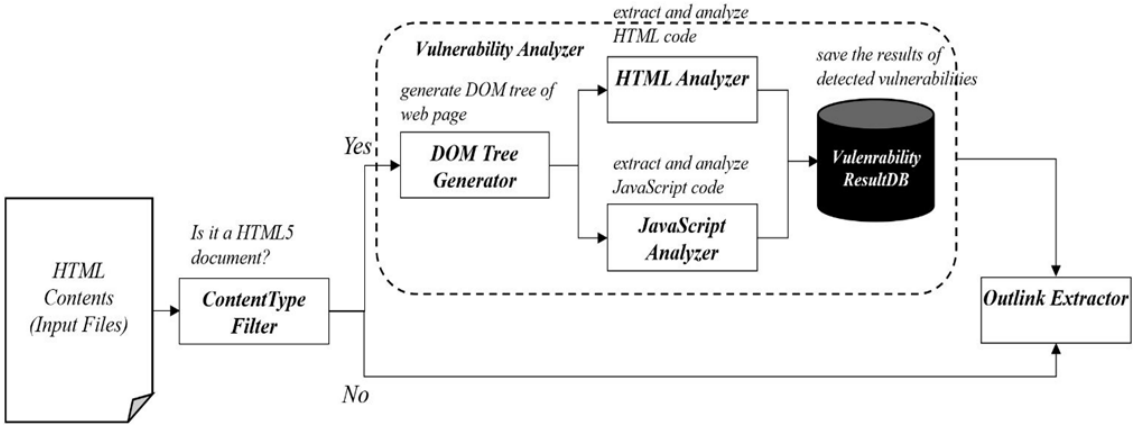

Figure 3. Architecture of HTML5 vulnerability detection.

4.3 Detection Method of HTML5 Vulnerabilities

In this paper, we classified vulnerabilities as tag and attribute vulnerabilities and Java Script vulnerability depending on the features, by applying different analytical methods. In the case of tag and attribute vulnerabilities detection, the detection process is performed based on a DOM tree, while in the case of Java Script vulnerability detection, it performs a keyword search in accordance with their scarcity.

4.3.1 Detection Method of Tag and Attribute Vulnerabilities

Tag and attribute vulnerabilities detection is performed based on a DOM tree. DOM is a W3C standard formula that represents the neutral structured document on the platform and language, and it is possible to access the elements of an HTML document through it. It can generate a tree structure using a DOM tree because a well-structured HTML document has a hierarchical structure, and there are elements that make up the structure of the HTML document.

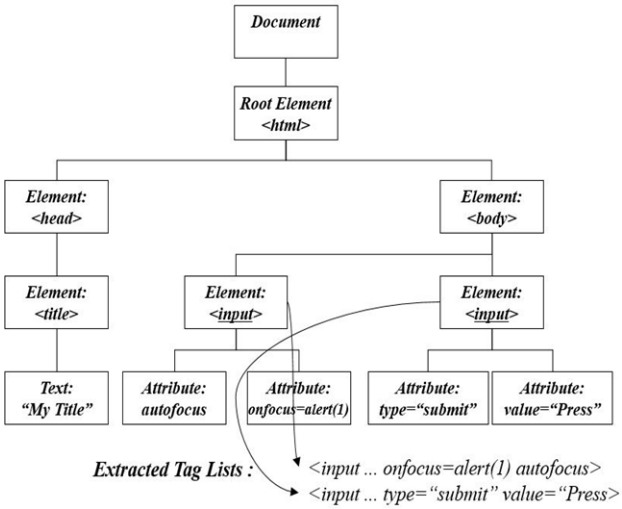

Figure 4. Example of tag and attribute vulnerabilities detection.

The tag and attribute vulnerabilities should be searched by looking for the attributes in a specific tag. In the case of HTML documents, since nested tags form a hierarchical structure, it is difficult to perform the detection using a simple keyword search. Using a DOM tree is suitable for a hierarchical approach with tag and attribute vulnerabilities detection because the access time can be reduced for child nodes for nested tags[16]. Vulnerability detection places the input document into a DOM tree and passes the <script> tag list to JavaScript vulnerability detection after extracting the keyword list in the tag, and then passes the contents of the rest of the tree to tag and attribute vulnerabilities detection.

In tag and attribute vulnerabilities detection, it retrieves the first tag of the defined vulnerabilities and extracts lists of matching tags from the tree, searches the attribute again with the same depth from the extracted tag list. For example, in the case of 'P2 XSS: -E0 input -A0 autofocus -A0 onfocus', as presented in Figure 4. it searches the vulnerable tag 'input' in the DOM tree, and extracts two tags lists, <input ... onfocus = alert (1) autofocus>, <input ... type = "submit" vlaue = "Press">. Since 'input', 'autofocus', and 'onfocus ' have the same number '0', it can be seen that autofocus and onfocus belong in the input tag. It the searches vulnerable properties 'autofocus', 'input' in the extracted tag list. If it detects all the tags and attributes defined as the vulnerability, it stores information about the vulnerability as the resulting file.

4.3.2 Detection Method of JavaScript Vulnerabilities

In the case of JavaScript vulnerabilities, it extracts the JavaScript code in the <script> tag from the DOM tree created in the initial depth of detection and performs detection. If the object created by the script specific area is used in another script region, all of the JavaScript code separate from each other in a document are the targets to be detected.

In the case of JavaScript vulnerability detection, if all of the keywords defined as vulnerabilities exist in a document, it is considered to be a vulnerability. For this process, it searches keywords by scarcity to minimize the number of searches. From P19 to P24 of Table 5 are lists of the high-scarcity order of the keywords of vulnerabilities to be detected.

The JavaScript keyword vulnerability detection should perform a keyword search for each vulnerability in the content of the script extracted from the input document. In the detection process, if the keywords to be detected do not exist in the script, it recognizes it as false and immediately starts the detection of the next vulnerability. If the keywords to be detected do exist in the script, it stores the information about the vulnerability as a resulting file.

5. Conclusion

In this paper, we carried out a new HTML5 potential security vulnerability detection process for the web documents of public institutions websites. Public institutions web sites consist of a large number of web documents that are unlike other general web sites, because such web sites provide information regarding policies, voting, and other events, and are connected with subordinate institutions. We therefore performed the crawling and the detection process based on the distributed parallel processing system Hadoop. By performing the crawling and detection using distributed parallel processing, we could reliably handle a large number of web documents of public institutions web sites.

In addition, we classified the vulnerabilities to be detected as either tag and attribute vulnerabilities or JavaScript vulnerability based on the characteristics of its entry, and applied different detection processes for the classified vulnerabilities.

6. Acknowledgment

This work was supported by the ICT R&D Program of MSIP/IITP. [B0101-15-0230, The Development of Script-based Cyber Attack Protection Technology].

7. References

1. Suk-Chull K, Jung-Sub P. HTML5 security issues in the next generation web standards environment. Korea Institute of Information Security and Cryptology. 2015; 24(4):44–55.
2. Lee S-H, Yim WG, Kim JH. Issues of views and information security perspective of the next generation of web standards HTML5. Korean Society for Internet Information. 2012; 13(2):47–54.
3. Ian H, Google Inc. HTML5: A vocabulary and associated APIs for HTML and XHTML W3C Recommendation. W3C; 2014.
4. Frances B. Using HTML5 for GOV.UK. Blog Government Digital Service; 2013.
5. USA. Whitehouse. Digital Government, Building a 21st century platform to better serve the American people. Available from: https://www.whitehouse.gov/sites/default/files/omb/egov/digital-government/digital-government.html.
6. Jaehun J, Anat H. The influence of information security on the adoption of web-based integrated information systems: an e-government study in Peru. Information Technology for Development; 2015 Jan 13. DOI 10.1080/02681102.2014.979393
7. HTML5 Security Cheat Sheet. OWASP. Available from: https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet
8. Kang S-C. Security Issues in the New HTML5 web services environment. Internet & Security Focus; 2013.
9. Clickjacking. OWASP. available from: https://www.owasp.org/index.php/Clickjacking.
10. Gustav R, Elie B, Dan B, Collin J. Busting frame busting: A study of clickjacking vulnerabilities on popular sites. IEEE Oakland Web 2.0 Security and Privacy Workshop; 2010.
11. Cross-site Scripting (XSS). OWASP. Available from: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
12. Khare R, Cutting D. Nutch: A flexible and scalable open-source web search engine; 2004.
13. Dean J, Ghemamwat S. MapReduce: simplified data processing on large clusters. Communications of the ACM - 50th anniversary issue, 2008; 51(1):107–13.
14. Dong G, Zhang Y, Wang X, Wang P. Detecting cross site scripting vulnerabilities introduced by HTML5. In: 2014 11th International Joint Conference on Computer Science and Software Engineering; 2014 May 14-16; Pattya, Thailand. 2014. p. 319–23.
15. Jin X, Hu X, Ying K, Du W, Yin H, Peri GN. Code injection attacks on HTML5-based mobile apps: Characterization, Detection and Mitigation. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security; 2014 Nov 3-7; Scottsdale, Arizona, USA; 2014. p. 66–77.
16. Chakrabarti S. Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction. In: Proceedings of the 10th international conference on World Wide Web; 2001 May 1-5; Hong Kong; 2001. p. 211–20.