A WEB CONTENT ANALYTICS ARCHITECTURE FOR MALICIOUS JAVASCRIPT DETECTION

JongHun Jung, Chae-tae Im, Soojin Yoon, hcbae

Internet Incidents Response Architecture Team
Korea Internet & Security Agency
{jjh2640, chtim, sjyoon, hcbae}@kisa.or.kr

ABSTRACT

Recent web-based cyber attacks are evolving into a new form of attacks such as private information theft and DDoS attack exploiting JavaScript within a web page. These attacks can be made just by accessing a web site without distribution of malicious codes and infection. Script-based cyber attacks are hard to detect with traditional security equipments such as Firewall and IPS because they inject malicious scripts in a response message for a normal web request. Furthermore, they are hard to trace because attacks such as DDoS can be made just by visiting a web page. Due to these reasons, it is expected that they could result in direct damages and great ripple effects. To cope with these issues, in this article, a proposal is made for techniques that are used to detect malicious scripts through real-time web content analysis and to automatically generate detection signatures for malicious JavaScript.

KEYWORDS

Script-based Cyber Attacks; Forward-Proxy Server; Malicious Java Script API; Deep Content Inspection; API Call Trace.

1. INTRODUCTION

Recent introduction of Ajax and HTML5 technologies has enabled dynamic representation of web content, providing compatibility between a client and a server in web environment. However, the efforts to deal with new security vulnerabilities in these technologies, such as the awareness, countermeasure technology development, and standardization are still insufficient. In particular, web-based attacks using malicious scripts can bypass traditional security equipments, such as IDS, IPS and Web Firewall, because, unlike conventional malicious code attacks, they do not download an executable file directly, but they still can be made by combining normal built-in APIs in JavaScript. And also, it is getting harder to detect these attacks as they employ traffic encryption and script obfuscation. The Figure 1 illustrates how a DDoS attack can be made with JavaScript just by accessing a web page. Furthermore, due to the accelerated introduction of HTML5, it is expected that cyber attacks exploiting vulnerabilities of new tags and APIs will grow rapidly.

In this article, a proposal is made for techniques that are used to detect malicious scripts through collection of HTTP Web traffics and static/dynamic analysis, and to generate a detection signature automatically. Chapter 2 shows trends in related studies. Chapter 3 describes techniques

Dhinaharan Nagamalai et al. (Eds) : CCSEA, DKMP, AIFU, SEA - 2015
pp. 11–19, 2015. © CS & IT-CSCP 2015 DOI : 10.5121/csit.2015.50202

that are used to collect and analyze web content for detection of malicious JavaScript. Chapter 4 describes more compact techniques that are used to generate a detection signature automatically with less false positive rate. Finally, Chapter 4 concludes the article.

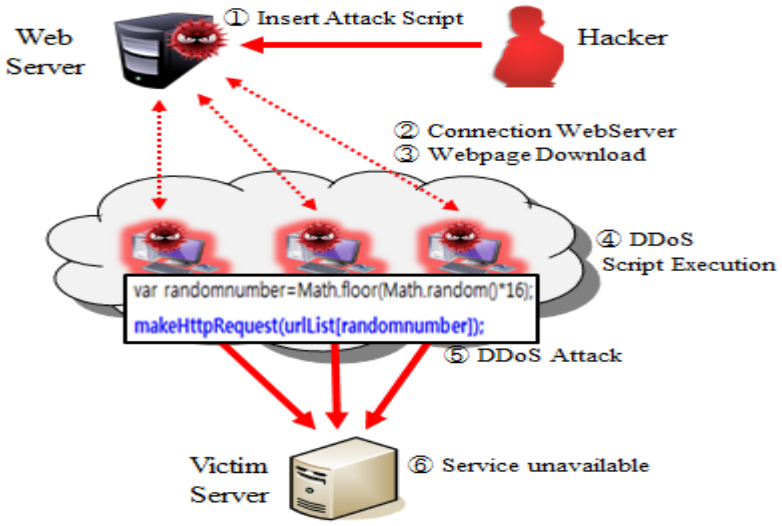

Figure 1. A DDoS attack using JavaScript code

2. RELATED WORK

2.1. WebShiled

Using a modified browser that consists of DOM API, HTML/CSS Parser and JavaScript Engine only, parse web content in proxy, turn it into the form of DOM Structure, and store it. Send the DOM Structure in the string format to the client. Send a script to the client only if it turns out that the script is safe after running it in the modified browser. However, prevention of exploitation of new vulnerabilities in HTML5 is insufficient.

2.2. A signature for Malware Detection

The method of Automatic generation of a signature for malware or worm can be divided into 5 categories: vulnerability-based, content-based, content-shifting, semantic-aware and honeypot-based. Among these, the content-based is the one that is proposed in this article.

In the content-based method, a signature target set is determined based on traffic and the same malicious behavior, and then a signature is generated based on the content.

A content-based signature [1] can be divided into Longest Common Substring, Longest Common Subsequence, Conjunction Signature and Bayes Signature. For the Longest Common Substring and Longest Common Subsequence, one retrieves the longest common substring and longest common subsequence respectively from the target set. For the Conjunction Signature, one uses a set of strings that appear in all targets, as a signature. For Bayes Signature, one checks whether a string in a sample appears in the malicious, and then determines whether the sample is malicious or not based on the percentage of malicious strings.

2.3 BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)

BIRCH is an algorithm for hierarchical clustering for a large database. BIRCH allows addition of a new value in a clustered tree as a new entity is added, eliminating the need of re-clustering.

BIRCH creates a CF (Clustering Feature) tree that has distance information for all leaves under a single node. As a new entity is added, it searches for the closest node. It adds the entity to the cluster of the closest node if the distance is same or shorter than threshold, or creates a new cluster and adds the entity to it if the distance is same or longer than threshold.

3. WEB CONTENT ANALYSIS TECHNOLOGY

For real-time detection of malicious JavaScript, one collects HTTP traffics by configuring a proxy server, and parses a HTML document and crawls a link to external resource in order to generate content for analysis. One performs static analysis, such as pattern-matching of web content, and dynamic analysis, such as checking whether obfuscated or not and the HTML5 tag percentage, to determine if the content is malicious. If a malicious script is found, remove the function that actually causes malicious behaviors before sending the script to the client. The Figure 2 shows the proposed system architecture that can be used to detect malicious scripts at network level.

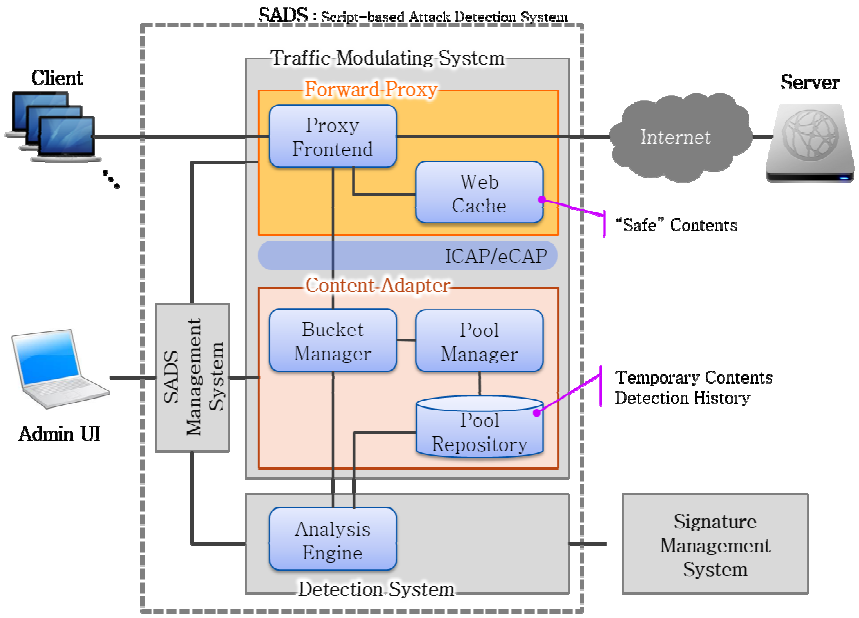

Figure2. System Architecture for Real-time Detection of Malicious Scripts

It consists of modules: i) Forward-Proxy, ii) Web Content Generation Module, iii) Analytics Engine (Static/Dynamic Analysis), and iv) System Control Module.

3.1 Forward-Proxy and Content Adapter

For collection of web traffics, Squid-Proxy Server is configured in the in-line form between clients and Web Server, where all HTTP Request and Response packets are collected and Internet Content Adaptation Protocol (ICAP) is used to pass the received HTTP traffics to Web Content Control Module. Then, Web Content Control Module extracts the external JavaScript link data

contained in the document, using HTML Parser received from Proxy Server, and collects resources for the link with a separate crawler to generate web content for analysis.

3.2 Web Content Analysis

The term 'web content' refers to the entire document that includes both a HTML document and external resources. As shown in the Figure 3, web content goes through the fast static analysis process that performs pattern-matching based on Yara-RuleSet[2]. However, because some sources, such as those obfuscated, require additional analysis, they go through the dynamic analysis process that uses Rihno Browser Engine to run the script and extract call trace data for detection.

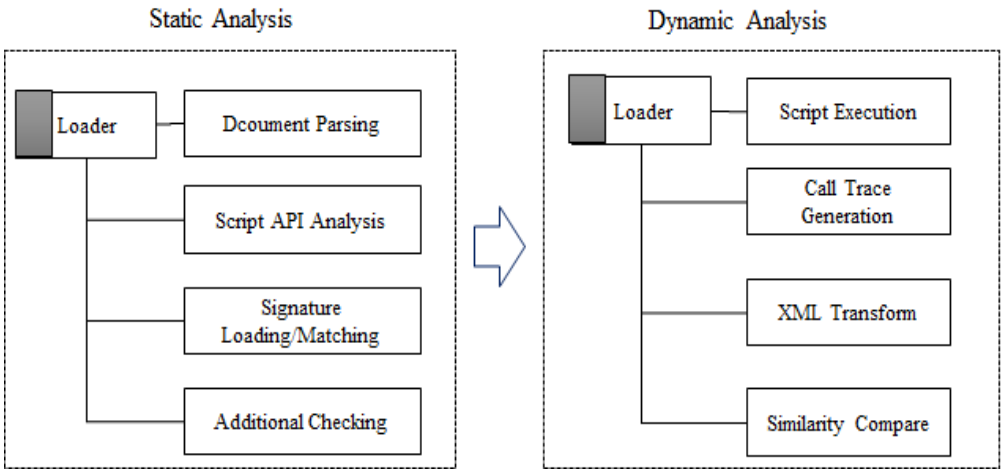

Figure3. Configuration of Web Content Analytics Engine

An input data set is in the format of JSON that consists of a HTML document, external JavaScript, and meta data (IP, port, protocol, domain, etc.). First, extract the primary key token to classify the type of the malicious behaviour. Table 1 shows summary of basic keywords contained in each malicious behaviour

Table 1. Examples of Basic Keywords for Each Malicious Behaviour

Malicious Type		The Keyword
DoS Attack	**HashDoS** **XML HttpObject DoS**	setInterval, open, send, ActiveXObject, XMLHTTP, XMLHttpRequest
Scan Attack	**Network Scan**	open, ActiveXObject, XMLHTTP, XMLHttpRequest, Date, readyState
	Port Scan	
Geolocation		coords, getCurrentPosition
Web Socket		parse, eval, WebSocket, JSON, send
Web Worker DDoS		postMessage, Worker, XMLHttpRequest, open, send

Look up the signature for a malicious behaviour and then perform signature-matching check to determine whether it is malicious or not.

Additionally, score the JavaScript obfuscation and the percentage of HTML5 new tag usage in the entire document, and then perform dynamic analysis if the score is the same or above the predetermined level. JavaScript obfuscation check is performed because most of malicious JavaScript codes are obfuscated, and it is hard to determine whether it is malicious just by doing signature–matching during static analysis. The Figure 4 illustrates process of the JavaScript obfuscation check[3]. As these are main characteristics of the obfuscated JavaScript, if a special character in a JavaScript string is frequently used, if there is a string with abnormal length, or if the entropy score of characters in the JavaScript is low, score them and if the total score is the same or above the cutoff, consider it obfuscated and perform dynamic analysis additionally.

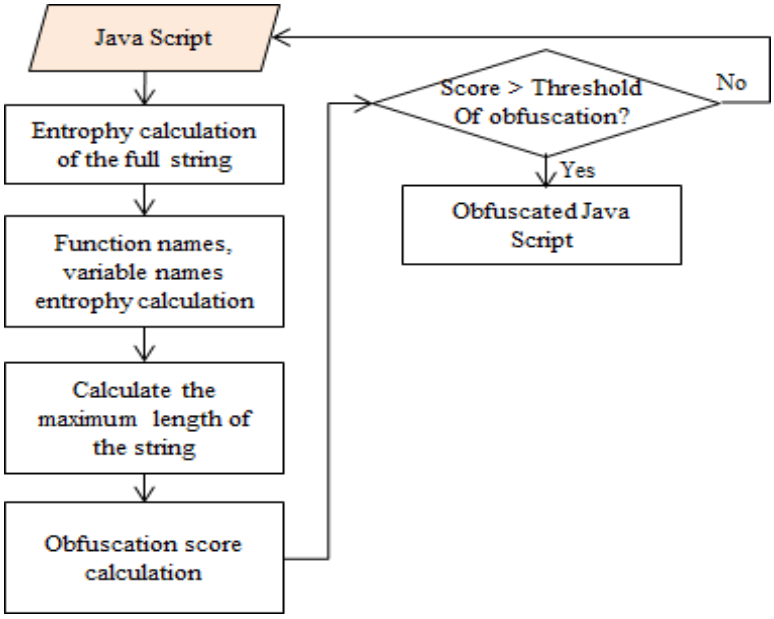

Figure4. JavaScript Obfuscation Analysis Process

And also, the usage of HTML5 tags is checked to detect malicious scripts such as jacking or cross-site scripts exploiting new tags of HTML5 (Canvas, Audio, Video). It has been arranged to perform dynamic analysis if a weight for each HTML5 tag is applied and the score is the same or above the predetermined level.

During dynamic analysis in real world situation, a malicious JavaScript is executed using open source-based Rhino JavaScript Engine with a built-in sandbox, and JavaScript API Call Trace data is extracted and stored in XML data format.

The Figure 5 shows the Function Call Trace data for Port Scan malicious JavaScript, converted to XML format.

```
<root>
<document.write>
<P1>&lt;div id=&quot;comments_thread&quot;&gt;Comments.&lt;/div&gt;</P1>
<Loc>Sample1:14398</Loc>
</document.write>
<setInterval>
<P1>100</P1>
<P2>function startRequest() {
createXMLHttpRequest();
xmlHttp.onreadystatechange = handleStateChange;
xmlHttp.open(&quot;GET&quot;, settingUrl, false);
xmlHttp.send();}</P2>
<Loc>Sample1:15232</Loc>
</setInterval>
<XMLHttpRequest.open>
<P1>GET</P1>
<P2>http://192.168.159.133</P2>
<P3>false</P3>
<Loc>Sample1:15622</Loc>
</XMLHttpRequest.open>
<XMLHttpRequest.send>
<Loc>Sample1:15665</Loc>
</XMLHttpRequest.send>
</root>
```

<setInterval>

<XMLHttpRequest.open>

<XMLHttpRequest.send>

Figure 5. Trace Data of a Port Scan Malicious Script

In this article, SimHash Algorithm[4] is proposed for comparison of JavaScript Function Call Trace similarities. SimHash utilizes Local Sensitive Hashing (LSH) for similarity comparison, and LSH maximizes conflicts between similar items rather than avoiding them. That is, the algorithm generates similar results for similar items. Using this function, regardless of the input value size, generate FingerPrint in an array in bit form just like the outcome of a normal hash function, and then use the hamming distance to measure the similarity.

4. TECHNIQUE OF GENERATING A SIGNATURE DEDICATED FOR DETECTION

In this article, the malicious script, malicious type, obfuscation status, meta data and other data received from the analytics engine are used for automatic generation of signature for malicious JavaScript. It is proposed that a detection signature can be automatically generated by clustering with a malicious script from the registered malicious script pool, generating the combined signature, and refining the signature. Figure 6 illustrates the process of signature generation.

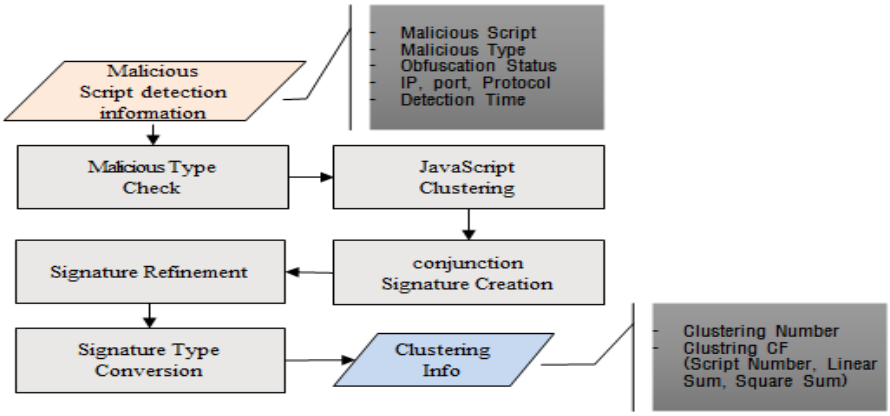

Figure6. Automatic Generation of the Detection Signature for a Malicious Script

4.1 Malicious Script Clustering

In this article, it is proposed to use the script clustering technique for automatic generation of the detection signature for a malicious script. The goal of clustering is to streamline the signature itself and improve the false positive rate by grouping malicious scripts showing similar behaviors, and thus preventing extraction of unnecessary tokens. For each token of malicious JavaScript, calculate the Term Frequency-Inverse Document Frequency value and vectorize it. The TF-IDF[5] weight is a statistical figure that is used to evaluate the importance of a certain term in a document, and it can be calculated as the product of Term Frequency and Inverse Document frequency.

The Term Frequency simply indicates how often a term appears in the document, and the Inverse Document Frequency provides general importance of the term.

$$\mathrm{tf}_{\mathrm{i,j}} = \frac{\mathrm{n}_{\mathrm{i,j}}}{\sum_{\mathrm{k}} \mathrm{n}_{\mathrm{k,j}}} \tag{1}$$

- $n_{i,j}$ indicates the number of times that Term t_i appears in document d_j.

$$idf_i = \log \frac{|D|}{\left|\{d_i : t_i \in d_j\}\right|} \tag{2}$$

- $|D|$ indicates Total Document Numbers
- $|di : ti \in d_j|$ indicates number of documents in which term t_i appears

$$tfidf_{ij} = tf_{ij} * idf_i \tag{4}$$

- TF-IDF weight is calculated by multiplying the TF and IDF.

Using the vector created with TF-IDF, perform hierarchical clustering in the Complete-linkage Cluster method. By improving BIRCH Algorithm for hierarchical clustering, quantify the vector distance and meta data (time similarity, IP, port and protocol), and then take their sum as the similarity score to determine whether malicious script clustering can be done.

Figure 7 shows the clustering process through modified distance calculation. For clustering purpose, the score of significant meta data similarity is applied to distance measurements between basic vectors in order to form a clustering tree.

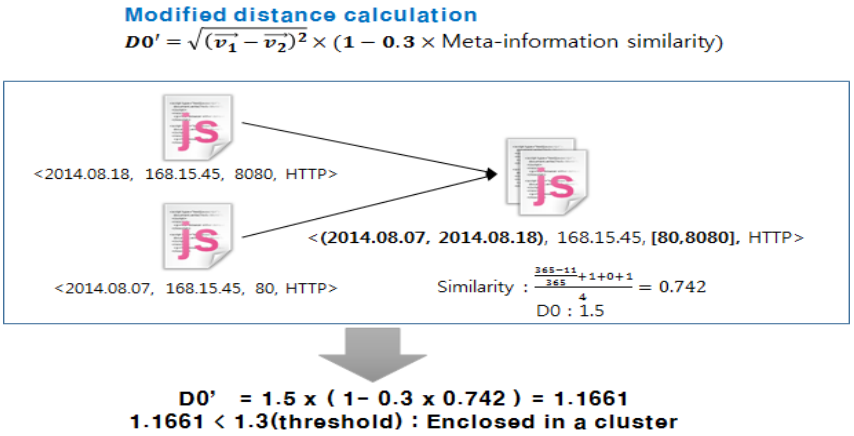

Figure7. Modified Distance Calculation Formula and Meta Similarity Application

4.2 Generating a Conjunction Signature

Extract a common token from a malicious script file within the allowed distance in a cluster to generate a conjunction signature. Convert a token in the same form, such as IP, to a regular expression before processing. The Table 2 shows the combined signature generated with Port Scan JavaScript.

Table 2. Examples of Conjunction Signature for Port Scan Detection

output, targetIP, endtime, starttime, ate, appendChild, break, wordWrap, createElement, onRequest, ActiveXObject, majorPort, (?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?) \ \.) {3 }(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)XMLHttpRequest, style, open, innerHTML, Array, XMLHTTP, true, onreadystatechange, restime, Microsoft, Close, send, scanRes, getElementById

4.3 Refining the Signature and Verifying the False Positive Rate

Verifying performance based on the detection signature generated shows that the number of unnecessary tokens or the false positive rate grows depending on the number of malicious script samples. Accordingly, an additional refinement of the signature is carried out by comparing with a token extracted from a normal web document, and eliminating the one that is duplicate or less than a certain length (3 characters).

The Table 3 shows the number of tokens and the false positive rate after the signature is refined. In this specific example, the signature has been compared against 28 malicious JavaScript codes and 300 JavaScript codes collected randomly for false positive verification. Group 3 shows the results after signature refinement. It can be seen that the false positive rate and the length of the signature generated (the number of tokens) have been significantly improved.

Table 3. The Results of Signature Refinement – False Positives

Section	Group 1	Group 2	Group 3
Average Detection Rate	100%	100%	100%
Average False Positive Rate	0%	8.8%	0%
Average Number of Tokens	146.7	89.5	115.9
The Number of Groupings	4 JavaScript Codes	9 JavaScript Codes	-
Refinement	×	×	○

5. CONCLUSION

In this article, a proposal has been made for techniques that are used to detect malicious JavaScript and to automatically generate detection signatures. While it shows good results if the signatures generated using the proposed techniques are employed to detect malicious scripts and measure the latency time, it requires additional experiments on a larger pool of samples and higher volume of traffics. Furthermore, to deal with security vulnerabilities of new APIs in HTML5, it is planned to expand the scope of the proposed dynamic analysis and conduct related

studies on detection of malicious behaviors by monitoring behaviors caused by JavaScript running,

ACKNOWLEDGMENT

This work was supported by the ICT R&D Program of MSIP/IITP. [14-912-06-002, The Development of Script-based Cyber Attack Protection Technology]

REFERENCES

[1] Z. Li, M. Sanghi, Y. Chen, M. Y. Kao, and B. Chavez, "Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience.", IEEE Symposium on Security and Privacy, May 2006.
[2] YARA Documentation,http://yara.readthedocs.org/en/latest/index.html
[3] Xu, Wei, Fangfang Zhang, and Sencun Zhu. "The power of obfuscation techniques in malicious JavaScript code: A measurement study." Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on. IEEE, 2012.
[4] Charikar, Moses S. "Similarity estimation techniques from rounding algorithms." Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. ACM, 2002.
[5] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval", Journal of Documentation, Vol.28, No.1, 1972, pp.11-21.