An Efficient Technique for Detection of Suspicious Malicious Web Site

K. Pragadeesh Kumar[1], Dr.N.Jaisankar[2], N.Mythili[3]
[2]School of computing science and Engineering, VIT University, Vellore, India
[1 &3] School of Information Technology and Engineering, VIT University, Vellore, India

***Abstract*—In today's web world web sites became attackers' main target. Since days before virus signatures had been used to detect malicious web pages. In this paper the malicious web pages will be detected using a prototype system based on the concept of abnormal visibility, also it detects the exact location of malicious code in the source code. The proposed prototype system uses a Web Spider which captures the entire link URLs associated with the web page. HTML parser will parse the web pages and convert the code into data structures recognized by the Detector. The Detector will match the structure with the abnormal visibility fingerprints and locates possible malicious code. The system proves higher performance, higher efficiency and lower maintenance cost, almost all malicious web pages are detected and the malicious codes encoded in the JavaScript. The system provides security alarm for end-users before visiting malicious web pages.**

I. Introduction

With the rapid extension of web applications, the attacks through websites have become common. When downloaded the malicious code can cause undesirable changes on the host system and affect normal operations. The prototype system use Web spider and HTML parser to get and parse the Web pages from target site. The system will parse the codes of the Web pages and convert them into the data structures that can be recognized by the detection engine, then match these structures with the abnormal visibility fingerprints and locate the possible malicious codes in the source codes of pages. For the reason that the attackers often use JavaScript to generate malicious codes dynamically, this system can also interpret and execute JavaScript scripts.

Cross-site scripting is gaining popularity among attackers as an easy exposure to find in Web sites. Left unattended, your Web site's ability to operate securely, as well as your company's reputation, may become victim of the attacks. This project is proposed to raise the awareness of this emerging threat and to present a solution implementation for Web applications to avoid this kind of attack.

There are wide variants of malware affecting the websites. The iframe HTML code to has been used mostly, to inject maliciousness into the websites using the iframe tags. The code may be injected into HTML, PHP, ASP or tpl source files. Themes or templates of Content management systems are the infectious source where in the malicious tags are integrated. The virus will also modify .htaccess and hosts files. The infection is not a server-wide exploit, it will only infect sites on the server that it has passwords to. The BadWare distribution is a widely known unavoidable peeking into the host servers and it is an inevitable need to have a solution

'Iframe virus' is a type of badware. Badware distribution has been expanded beyond traditional channels like email viruses to harder-to-avoid methods like automated "drive-by downloads" that are launched by compromised web pages.

The iframe variants will also sneak through JavaScript. iframe tags may not be seen in plain text in the source because it is encoded. If the encoded script code is decoded, it will contain code to invoke iframe via JavaScript.

Our goal is to develop a prototype system which has to detect the malicious iframe tags in the HTML code and in encoded JavaScript. The Malware detection tool will look into the HTML code and decode JavaScript too if any, each time when a webpage is loaded to look for any maliciousness. For detection we use the concept of 'Abnormal Visibility Recognition'. The concept relies on analyzing the property of iframe tags, especially for width and height value to meet the threshold value. The java HTML parser will have major part of job which has to be done.

II. Related Work

M. Almgren, H. Debar, M. Dacier, findings aimed at presenting an intrusion-detecting tool for protecting the web servers from attacks. The paper discuss on providing a tool that can run at real time for detecting undesirable changes and keep track on suspicious hosts. The design is flexible and uses virus signature matching for detection and try to reduce number of false alarms. The intrusion detection system have been discussed has the task of monitoring the usage of information systems to detect the apparition of insecure states either by authorized users or any external parties. The attack generally target on operating system and services other than web server.

The tool allows the search for flexible attack signatures in any field of the server logs. By grouping these into classes, similar attacks can be generalized under one name to save time. It also allows different alerts to be merged, and it will perform refined checks if certain conditions are met. The signatures are not limited to matching simple cgi programs, but are extended to detection of denial-of-service attacks. The design of the tool is modular to allow it to be extended in the future. It is portable between different platforms, and can run in real time.

C. Seifert, I. Welch, P. Komisarczuk, used 'Honey pot' technique to detect attacks. It focuses on malicious web servers, which they interact with by driving a web browser on the dedicated honeypot system. Honeyclient detects successful attacks by monitoring changes to a list of files, directories, and system configuration after the Honeyclient has interacted with a server.

Alexander Moshchuk, Tanya Bragin, Damien Deville, Steven D. Gribble, and Henry M. Levy. SpyProxy intercepts and evaluates Web content in transit from Web servers to the browser. SpyProxy executes active Web content in a safe virtual machine before it reaches the browser. Because SpyProxy relies on the behavior of active content, it can block zero-day attacks and previously unseen threats. Thus the method overcomes the 'strider Honeymonkey problem, Zeroth day attack. The system shows some delay in execution but its negligible to some extent.

The proposed system has been proven to be more efficient than the existing system that uses virus signature matching. The signature matching technique will have disadvantages of maintaining huge database of signature and the host is vulnerable to attacks until the patch has been released. The proposed system uses a significant method- Abnormal Visibility Recognition, for maliciousness detection. The system maintains a relatively fewer abnormal visibility modes which is more stable than signatures. The method examines only few attributes of HTML tags instead of maintaining huge signature database. The method records the location of suspicious code too, that helps the admin to rectify the maliciousness.

The disadvantage may be the parser has to look into the same HTML code each time when the URL is entered were the time consumption causes little inconvenience but there is no other way as the Web Pages are dynamic and cross-site scripts can interrupt at any time when any hoax is done.

III. PROPOSED WORK

We design and implement a prototype of malicious Web pages detection system based on abnormal visibility recognition. The architecture of the prototype system is depicted in Figure 1. The proposed system consists of four main components:

1. Web spider.
2. HTML parser.
3. JavaScript engine.
4. Abnormal visibility detector.

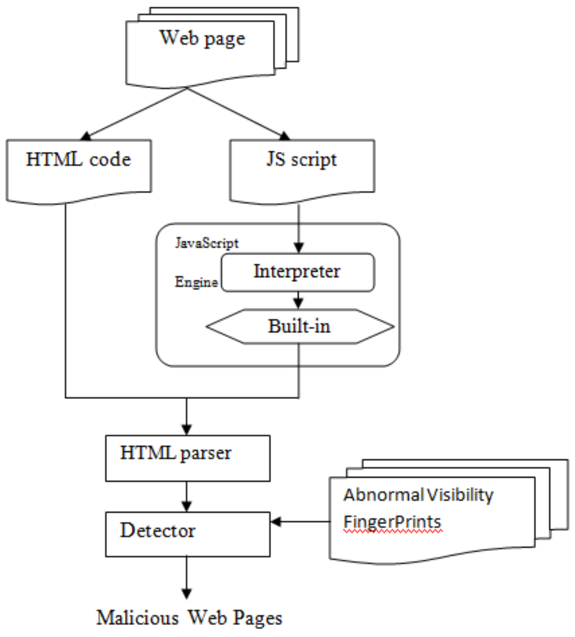


Figure 1. Architecture of the system

The web spider will automatically crawl from the target website via the HTML source code and collect the possible URL links associated with the site. The link URLs are made into a queue and will be fed into the HTML parser. The spider starts crawling once when initial URL link is given. The program will collect all links including URLs from e-mail links until any error or broken link is found. Reports will be given for any errors.

As soon as a recognized URL is given, the Spider crawls through the site for examining for good and bad links. It will indicate for broken links too.

```
spider = new Spider(this);
spider.clear();
base = new URL(url.getText());
spider.addURL(base);
spider.begin();
```

The HTML parser will parse the contents of each link. This is the specific module of more significant process, in which unique class has to be defined to determine erroneous constructs. Dependency: solely depend on previous module's output source. In which broken links will be skipped off from parsing as the report is intimated already. Java contains support both for accessing the contents of URLs and parsing HTML. The "processURL" method, which is called for each URL encountered, does this. Reading the contents of a URL is relatively easy in Java.

```
URLConnection connection = url.openConnection();
    if ( (connection.getContentType()!=null) &&
       !connection.getContentType().toLowerCase()
              .startsWith("text/") ) {
      getWorkloadWaiting().remove(url);
      getWorkloadProcessed().add(url);
      log("Not processing because content type is: " +
         connection.getContentType() );
      return;     }
```

A special-purpose JavaScript interpreter is implemented to get the execution output of browser-end scripts that are often used to generate malicious code dynamically by attackers. Our JavaScript engine consists of a Rhino-based JavaScript interpreter and some necessary simulation built-in objects. The scripts extracted from original Web page will be fed to JavaScript engine; the interpreter will interpret and execute them. When the scripts need to output some data to browser, e.g. HTML codes generated, the built-in objects will receive the data like a real browser. After interpreting, the output data collected by built-in objects will be integrated with HTML codes of original Web page and passed to parser. As soon as the links obtained simultaneously the HTML code for each link will be scrutinized for suspicious tags, as assisted by 'HTMLEditorKit' class of java.

Already as defined the detector will analyze the tag attributes by matching it with the abnormal visibility fingerprints. For plain HTML codes, the detector will check the width and height attributes of related HTML tags directly. For JavaScript scripts, detection will be performed to the tags in the results of interpretation and execution. The width and height values will be compared with a threshold. If they are less than the threshold, we consider that there is an abnormal visibility in the Web page and it would be regarded as a possible malicious page. If the tag value for a given page is less than the threshold value set, it is detected to be a possibly malicious web page. So far, the prototype system has three kinds of fingerprints as follows:

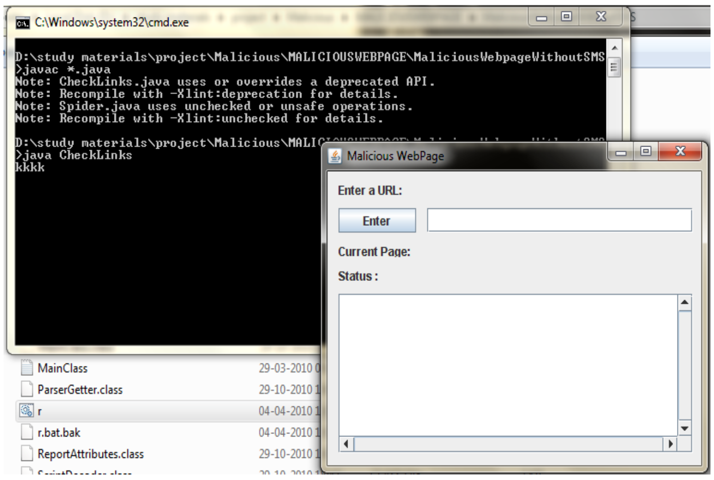
1) Abnormal width or height.
2) Abnormal display: none display style.
3) Abnormal iframe generated by scripts

A detector is used to detect abnormal visibility. The system determines whether a tag is abnormal by matching its attributes values. To calculate the values of width and height attributes set in percentage format, we define a generalized display screen area as a calculation baseline whose area is 1027*768 pixels.

IV. RESULTS AND DISCUSSION

The spider begins processing when the begin method is clicked. To allow the example program to maintain its User Interface, the spider is started up as a separate Thread. Clicking the "Begin" button begins this background spider thread. When the background thread begins, the run method of the "CheckLinks" class is called. The run method begins by instantiating the Spider object. This can be seen here:

```
spider = new Spider(this);
spider.clear();
base = new URL(url.getText());
spider.addURL(base);
spider.begin();
```



```
C:\Windows\system32\cmd.exe
D:\study materials\project\Malicious\MALICIOUSWEBPAGE\MaliciousWebpageWithoutSMS
>javac *.java
Note: CheckLinks.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: Spider.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

D:\study materials\project\Malicious\MALI
>java CheckLinks
kkkk
```

MainClass	29-03-2010
ParserGetter.class	29-10-2010
r	04-04-2010
r.bat.bak	04-04-2010
ReportAttributes.class	29-10-2010

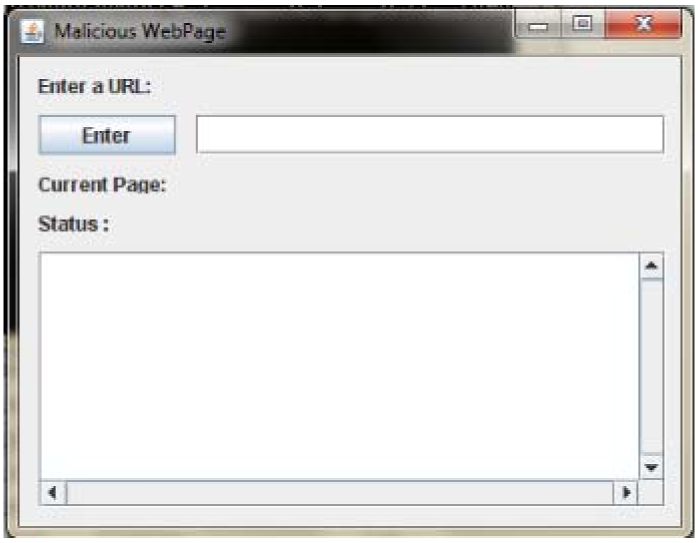
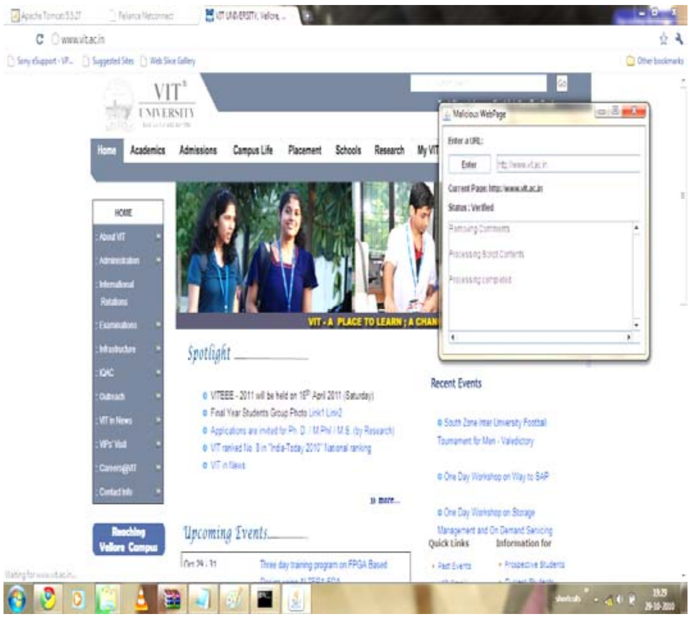


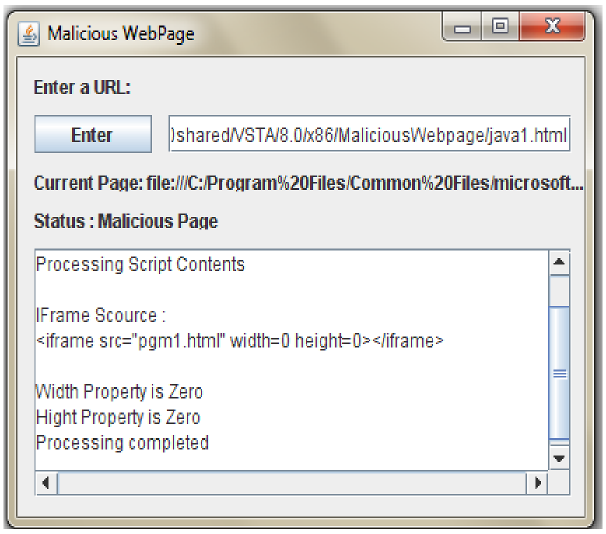
Figure 2. User interface after execution.



Current Page: http://www.vit.ac.in
Status : Verified
Removing Comments
Processing Script Contents
Processing completed

Figure 3. Processing a harmless website

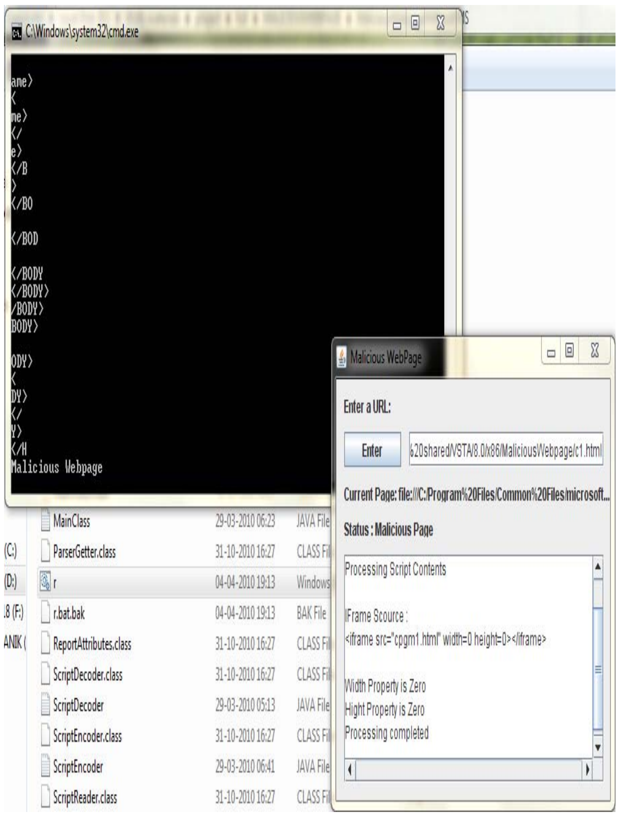
Determine a malicious website



```
Malicious WebPage
Enter a URL:
[Enter]  )shared/VSTA/8.0/x86/MaliciousWebpage/java1.html
Current Page: file:///C:/Program%20Files/Common%20Files/microsoft...
Status : Malicious Page

Processing Script Contents

IFrame Scource :
<iframe src="pgm1.html" width=0 height=0></iframe>

Width Property is Zero
Hight Property is Zero
Processing completed
```

Width or height attributes of 'iframe' tags

For determination on of a malicious site, using the defined properties we have developed a set of example to hide some display features of the webpage. Thus we can check easily whether the system detects the maliciousness.



```
C:\Windows\system32\cmd.exe
ame>
<
me>
</
e>
</B
>
</BO

</BOD

</BODY
</BODY>
/BODY>
BODY>

ODY>
<
DY>
</
Y>
</H
Malicious Webpage
```

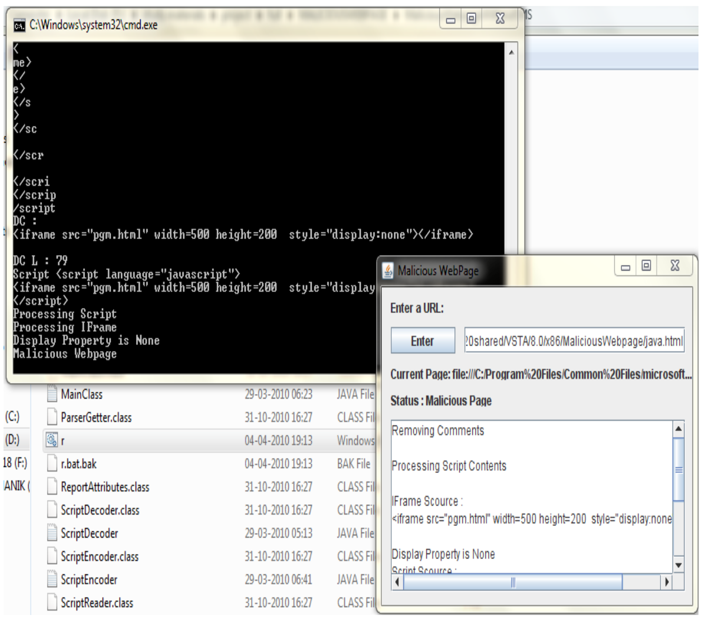
```
MainClass              29-03-2010 06:23   JAVA File
ParserGetter.class     31-10-2010 16:27   CLASS Fil
r                      04-04-2010 19:13   Windows
r.bat.bak              04-04-2010 19:13   BAK File
ReportAttributes.class 31-10-2010 16:27   CLASS Fil
ScriptDecoder.class    31-10-2010 16:27   CLASS Fil
ScriptDecoder          29-03-2010 05:13   JAVA File
ScriptEncoder.class    31-10-2010 16:27   CLASS Fil
ScriptEncoder          29-03-2010 06:41   JAVA File
ScriptReader.class     31-10-2010 16:27   CLASS Fil
```

```
Malicious WebPage
Enter a URL:
[Enter]  %20shared/VSTA/8.0/x86/MaliciousWebpage/c1.html
Current Page: file:///C:/Program%20Files/Common%20Files/microsoft...
Status : Malicious Page

Processing Script Contents

IFrame Scource :
<iframe src="cpgm1.html" width=0 height=0></iframe>

Width Property is Zero
Hight Property is Zero
Processing completed
```

Figure 4. Showing malicious page in case of width and height property is zero both

Set the display style of the 'iframe' tags to "display: none"

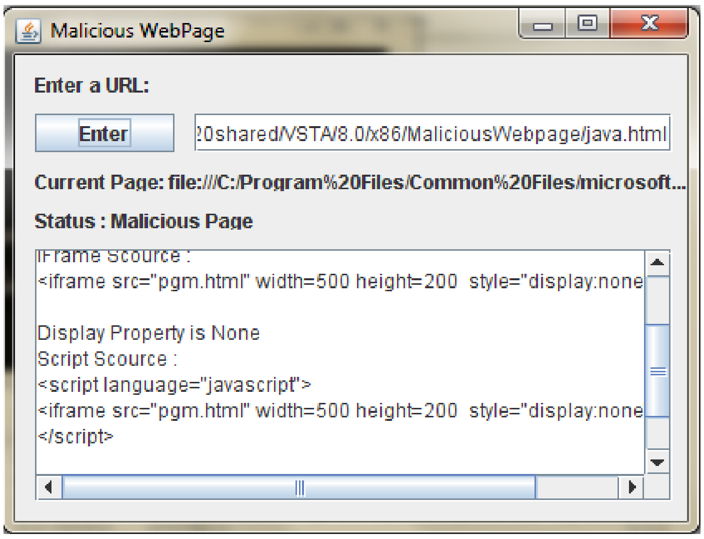
Our method can record the location of the tags when parsing the Web pages so as to locate the malicious codes in the source codes accurately.



```
C:\Windows\system32\cmd.exe
<
me>
</
e>
</s
>
</sc

</scr

</scri
</scrip
/script
DC :
<iframe src="pgm.html" width=500 height=200  style="display:none"></iframe>

DC L : 79
Script <script language="javascript">
<iframe src="pgm.html" width=500 height=200  style="display
</script>
Processing Script
Processing IFrame
Display Property is None
Malicious Webpage
```

```
MainClass              29-03-2010 06:23   JAVA File
ParserGetter.class     31-10-2010 16:27   CLASS Fil
r                      04-04-2010 19:13   Windows
r.bat.bak              04-04-2010 19:13   BAK File
ReportAttributes.class 31-10-2010 16:27   CLASS Fil
ScriptDecoder.class    31-10-2010 16:27   CLASS Fil
ScriptDecoder          29-03-2010 05:13   JAVA File
ScriptEncoder.class    31-10-2010 16:27   CLASS Fil
ScriptEncoder          29-03-2010 06:41   JAVA File
ScriptReader.class     31-10-2010 16:27   CLASS Fil
```

```
Malicious WebPage
Enter a URL:
[Enter]  20shared/VSTA/8.0/x86/MaliciousWebpage/java.html
Current Page: file:///C:/Program%20Files/Common%20Files/microsoft...
Status : Malicious Page

Removing Comments

Processing Script Contents

IFrame Scource :
<iframe src="pgm.html" width=500 height=200  style="display:none

Display Property is None
Script Scource :
```



```
Malicious WebPage
Enter a URL:
[Enter]  20shared/VSTA/8.0/x86/MaliciousWebpage/java.html
Current Page: file:///C:/Program%20Files/Common%20Files/microsoft...
Status : Malicious Page

IFrame Scource :
<iframe src="pgm.html" width=500 height=200  style="display:none

Display Property is None
Script Scource :
<script language="javascript">
<iframe src="pgm.html" width=500 height=200  style="display:none
</script>
```

Figure 5. Showing malicious code location in the source code by decoding javascript.

V. Conclusion and Future Enhancements

In this paper, based on the analysis and statistics of Web malicious codes, abnormal visibility and propose a corresponding detection method is introduced to detect malicious Web pages effectively and efficiently. A prototype detection system is implemented based on abnormal visibility recognition. The experiments show that the system can detect a Web site with high performance, and can detect almost all the Web pages containing malicious codes. The system can be used to monitor the security state of target Web sites and provide security alarm for end users before visiting malicious Web pages. In the future, we will improve and refine the abnormal visibility fingerprints to avoid false negative as much as possible.

REFERENCES

[1] G. McGraw and G. Morrisett. "Attacking malicious code: report to the Infosec research council," IEEE Software, Vol. 17, No. 5, pp. 33-41, 2000.

[2] M. Christodorescu, and S. Jha. "Testing malware detectors," Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis 2004, pp. 34- 44, Boston, MA, USA, July 2004. .

[3] I. S. Ja J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith. "Detecting malicious code by model checking," Proceedings of the 2nd International Conference on Intrusion and Malware Detection and Vulnerability Assessment, Vol. 3548, pp. 174-187, Vienna, Austria, July 2005. [CrossRef] .

[4] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant. "Semantics-aware malware detection," Proceedings of the 2005 IEEE Symposium on Security and Privacy , pp. 32-46, Oakland, CA, USA, May 2005.

[5] J. Bergeron et al. "Static Detection of Malicious Code in ExecuTable Programs," Symposium on Requirements Engineering for Information Security, Indianapolis, Indiana, USA, March 2001.

[6] M. Christodorescu and S. Jha. "Testing malware detectors," Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis 2004, pp. 34-44, Boston, MA, USA, July 2004.

[7] M. Young, Alexander Moshchuk, Tanya Bragin, Steven D. Gribble, and Henry M. Levy. "A Crawler-based Study of Spyware on the Web," In Proceedings of the 2006 Network and Distributed System Security Symposium, pages 17-33, February 2006.

[8] Bin Liang et al. Malicious Web Pages Detection Based on Abnormal Visibility Recognition, IEEE, 2009.

[9] Provos, N., McNamee, D., Mavrommatis, P., Wang, K., Modadugu, N. "The Ghost In The Browser Analysis of Webbased Malware, " First Workshop on Hot Topics in Understanding Botnets April 10, 2007, Cambridge, MA.