

Design and Implementation of Dynamic and Efficient Web Crawler for XSS Vulnerability Detection

Ao Chai^{1,a}

¹College of Computer and Control Engineering, Nankai University, Tianjin, 300071, China

^aemail: chaiao12@163.com

Keywords: Crawler; Parallel; XSS; Vulnerability

Abstract. With the continuous development of Web technology and the sustained popularization of the Internet, the security problems of the Web has been attracting more and more attention from the network workers. According to the OWASP (Open Web Application Security Project), XSS (Cross-Site Scripting) has been among the top three vulnerabilities in a number of threats. Therefore, it is very meaningful to design XSS vulnerability automatic mining tools. Crawler, as a web data scratching tool, has been favored by search engine workers. In this paper, we introduce a dynamic parallel crawler aiming for XSS vulnerability detection, and the basic principle of XSS. Based on them, we designed an efficient XSS detection tool. Finally, an experiment shows that our tools can detect XSS vulnerabilities efficiently.

Introduction

With the development and popularization of the Internet, E-mail, Blog, Forums, Post Bar, etc. has become an indispensable part in people's work and life. While these services bring convenience to people, there are still more and more potential risks. XSS (Cross-Site Scripting) attack, for example, is widespread in the Internet, and has been security risks to the Web. It allows the attacker to inject the script into the pages, which can cause serious damage to user privacy. According to the newest Symantec Internet Security Threat Report [1], XSS is listed as No.5 of "Top 10 Vulnerabilities Found Unpatched on Scanned Web Servers". Meanwhile, in the security vulnerability list published by OWASP (Open Web Application Security Project), XSS ranks as top three [2].

In order to detect the presence of XSS vulnerabilities in the Web system, many security researchers have been studying various automation tools. The XSS mining tool based on crawler is an efficient and active detection method [3], which is commonly used to assess the security of Web applications. In this paper, we designed a dynamic multi-threaded crawler tool to help XSS vulnerability detection, which uses an improved Bloom Filter algorithm for URL duplicate checking.

Efficient parallel network crawler design

The basic principle of crawler and the basic structure of an efficient parallel crawler

Web crawler [4] usually starts with a collection of URL, which called a seed set (each file on the network has an address, i.e. URL). The crawler will first fetch the URLs from the collection, which have been ordered in a queue. In a certain order, the crawler downloads the pages, analyze the page content, collect useful information, and extracts the new URLs then stored them in the queue, which is mentioned earlier, to download later. Repeat the above procedure until the URL queue is empty or satisfies a crawling termination condition, so as to traverse the target Web.

In this paper, we introduce an efficient dynamic parallel network crawler. It consists of a crawler controller to manage all crawler threads, dynamically creates the required crawlers and destroys idle crawlers. Meanwhile, it has an URL resolving module. Before crawlers download the pages, the module maps all the URL to IP address, so that the burden of crawlers can be greatly reduced and the efficiency can be dramatically improved. At the same time, we use an improved Bloom Filter

algorithm to check the duplication of URLs. The function modules and the architecture are shown in the following figure, where the Crawler Controller, the URL Resolver, and the URL Update module must access the URL Frontier in a mutually exclusive way.

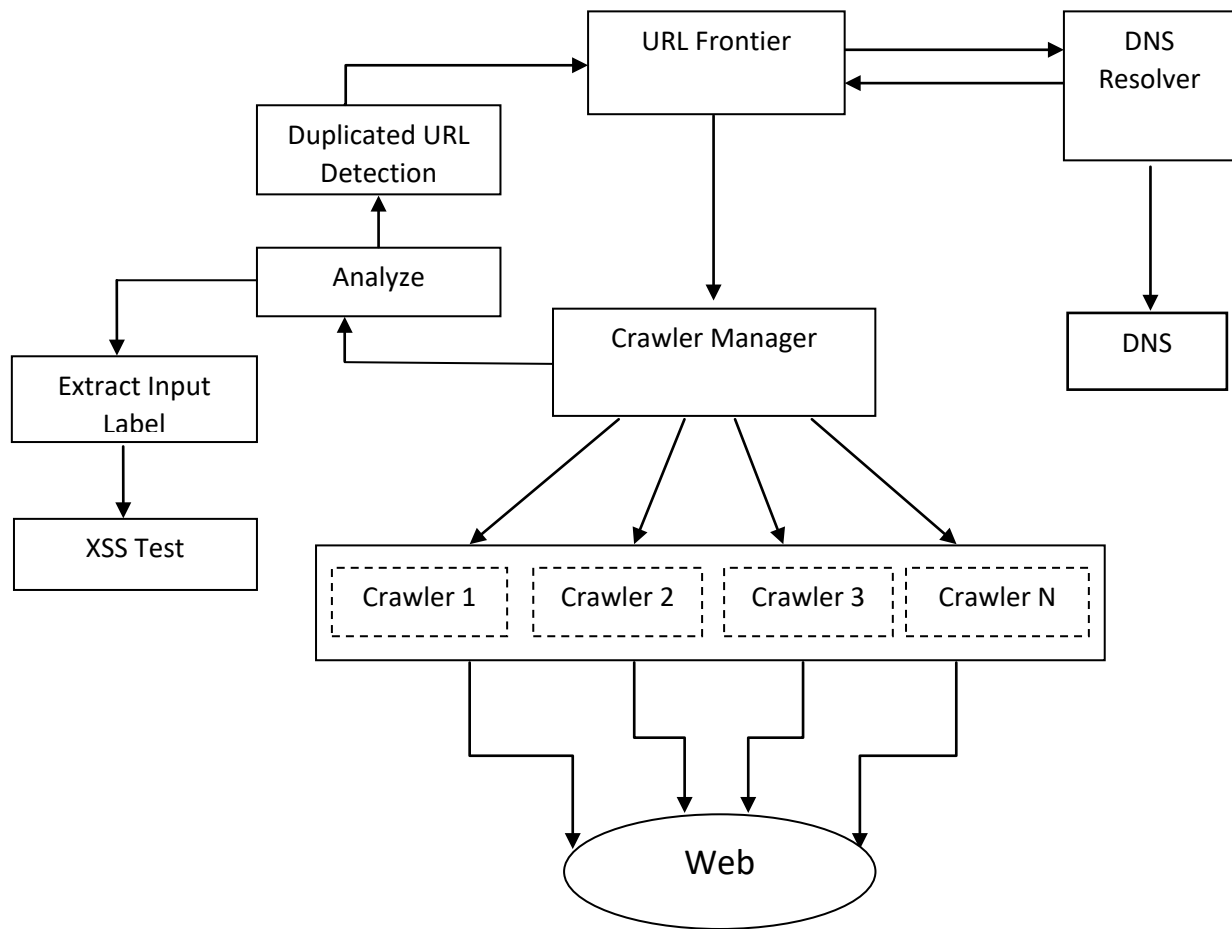


Fig.1. The architecture of the dynamic parallel crawler

Parallel module

If there are lots of crawler threads accessing the URL Frontier exclusively, it's likely to cause the URL Update process busy, according to the "Readers-Writers problem" [5]. So here, we set up a crawler controller to fetch URLs from URL Frontier in a batch mode. The controller is in mutual exclusion with the other two modules while visiting the URL Frontier. Due to the unpredictability of the network environment [6], it is difficult to predict the time that each crawler thread spends, as well as the appropriate number of crawler threads. Therefore, our crawler controller also has a function that create new threads and destroy idle threads. Both of them depend on the number of requested URLs as well as the Internet situation.

The HTTP requests must be set when crawlers access the Internet, such as HTTP headers, cookie settings, etc. After the crawlers download the pages, they analyze the HTML, extract the input tags and links contained within the page, then send the links into an URL buffer exclusively.

The pseudo-code of the crawler controller is given below:

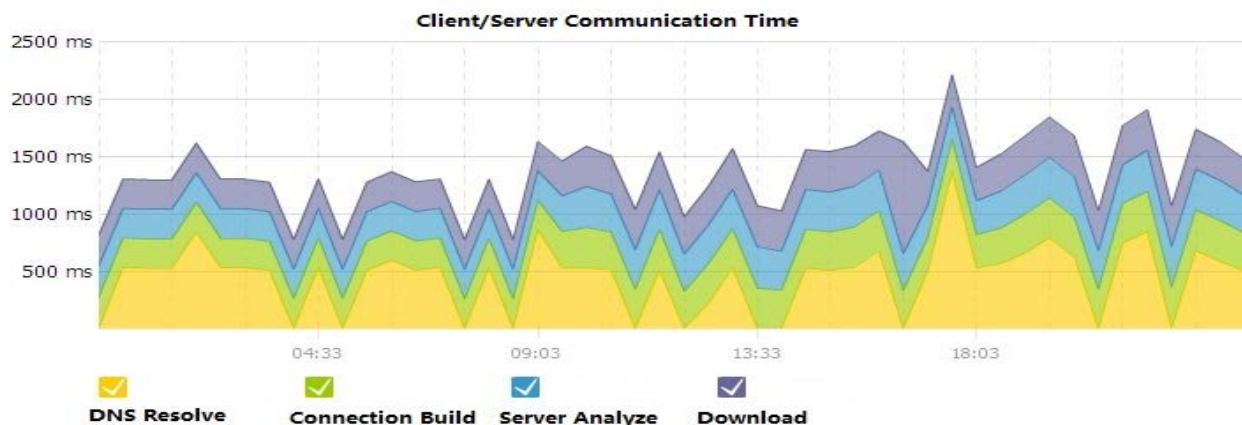
```
void CrawlerController() {
    while(TRUE) {
        if(point-length!=0){
            down(&mutex_URLFrontier);
            Read(URLFrontier,50);
            up(&mutex_URLFrontier);
        }
        else {
            up(&mutex_URLFrontier);
            continue;
        }
        while(buffer!=null) {
            if(IdleCrawler==0) Create(NewCrawlerThread,URL);
            else Assign(CrawlerThread,URL);
        }
        if(IdleCrawler!=0) Kill(CrawlerThread);
    }
}
```

The pseudo-code for the crawler is as follows:

```
void Crawler() {
    Assemble(HTTPHead);
    Request(URL);
    urls=ExtractNewURL(document);
    down(&mutex_URLBuffer);
    write(urls,URLBuffer);
    up(&mutex_URLBuffer);
    Sleep();
}
```

URL Resolving module

Typically, the server, where the HTML page located in, has a domain name. The domain name must be translated into the corresponding IP address before browsers or crawlers can access. The Internet provides a specific service, which is called DNS (Domain Name System), to map the domain name to the IP address. The experiment shows that when the client sends an HTTP request to the server, DNS resolving time consumes at least 30% of the total time. Further, due to the network situation, the request would be also lost at certain possibility [7].



Therefore, we set up a DNS resolving module specifically in the crawler system, which works in parallel with the Crawlers. It visits the Internet DNS server, maps the URL and IP, and saves

them in pair in the URL Frontier. In this way, each crawler no longer has to visit the DNS server but can directly access the target site. Thus, the efficiency of crawler access is improved. The pseudo-code is as follows:

```
void IPResolver() {
    while(true) {
        down(&mutex_URLFrontier);
        Read(URLFrontier);
        up(&mutex_URLFrontier);
        Call(DNS to resolve IP of URL);
        down(&mutex_URLFrontier);
        Write(URLFrontier);
        up(&mutex_URLFrontier);
    }
}
```

Parallel URL duplicate checking module

When crawlers work in parallel mode, they will encounter duplicate URLs inevitably. If the crawlers take time to access the repeated URLs, the efficiency of the system will be seriously influenced. So how to prevent the crawler from accessing the same pages, that is, how to quickly determine whether a URL is already in the URL Frontier, is a very crucial part.

We use an improved Bloom Filter algorithm. The traditional Bloom Filter algorithm was proposed by Burton Howard Bloom in 1970. It is based on hash to detect whether an element exists in a particular set. It is a space- and time-efficient data structure[8].

It sets an array of i bits. Initially, each bit is set to be 0. Then it sets j completely independent hash function.

$$h_k = H_k(key) (1 \leq k \leq j) \quad (1)$$

Each hash function $H_k(key)$ can map the element which is to be detected to a bit of the array. Now, the element is operated by j different hash, get j results $h_1, h_2, h_3 \dots h_j$. Then set the corresponding j positions in the array to be 1. When testing the next element, do the same operation, and observe whether j new positions are already set to be 1 in the array. If so, it represents the element is in the collection. Figure:

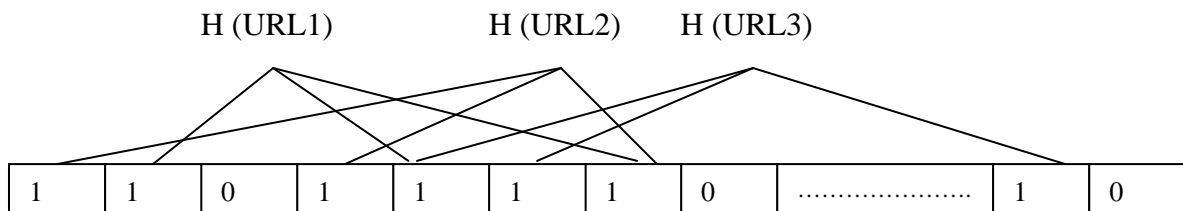


Fig. 2. The principle of Bloom Filter

It is not difficult to see that, due to the characteristics of the hash function, the Bloom Filter algorithm has a certain error rate, that is, the elements that are not actually within the collection may be considered as existed, (on the other hand, the elements which exist in the collection can be truly detected), so the algorithm can result in False Positive. In the case that hash functions are properly selected, it is obvious that the array size i and the number of hash functions j determine the probability of False Positive. According to the literature [8], we have the exact probability of False Positives below:

$$\sum_i \Pr(q = t)(1 - t)^j \approx (1 - E[q])^j = (1 - (1 - \frac{1}{i})^{jn})^j \approx (1 - e^{-jn/i})^j \quad (2)$$

We improved the traditional Bloom Filter so that it can support parallel computing. We set hash functions to make sure each of them has separated result. Namely, all the hash have j/i consecutive bits in the array. Therefore, these hashes can be calculated and checked simultaneously. Figure:

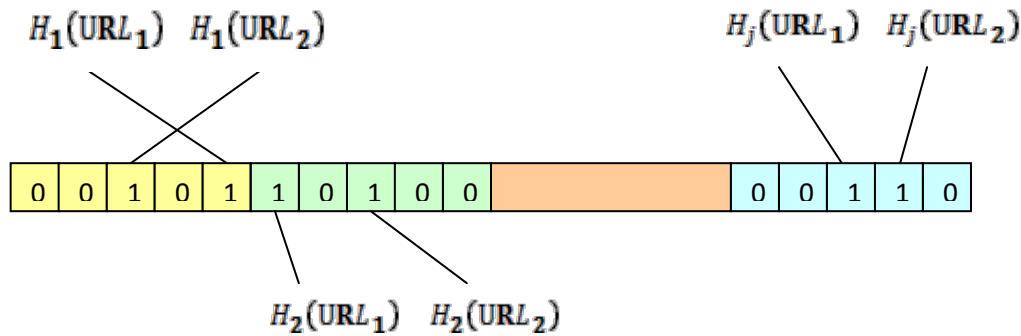


Fig.3. The principle of Parallel Bloom Filter

We deduce the False Positive rate of this improved Bloom Filter as follow. Since a bit in the array can only be calculated by one hash function and its probability is (j/i) , then the probability that the bit is not selected is $(1 - j/i)$. If there are n elements added to the collection, the probability that a bit in an array is zero after storing the whole elements should be $(1 - j/i)^n$. We have $(1 - j/i)^n \approx e^{(-jn/i)}$. By

$$(1 - \frac{j}{i})^n \leq (1 - \frac{1}{i})^{jn} \quad (3)$$

As n increases, they are infinitely close. So the probability of False Positive in improved Bloom Filter is only slightly higher than the traditional one. However, considering that our algorithm supports parallel, this small sacrifice is still worthwhile.

XSS basic principles and variation rules

XSS introduction

XSS (Cross-Site Script) is an application-level security issue which allows the attacker to inject client-end scripting into a Web pages. Combining with other vulnerabilities, it can lead to a worm attack that causes serious harm to user's privacy. Many security workers have been studying various approaches of detection and defense. According to the characteristics of XSS and its attack methods, security workers divided into three main types generally:

- Reflected XSS Attack: This type of XSS is the most common, but also the most widely used by hackers. The attacker attaches the malicious script to the URL parameter. When the user clicks on the link, the code will execute at user's host.
- Stored XSS Attack: With lower trigger cost, this type of XSS is more threatening than reflected XSS, and can even affect the security of Web server.
- DOM-based XSS Attack: By modifying the page DOM node information to form the XSS attacks. It often needs to be constructed for specific JavaScript DOM.

Transform the primitive attack vectors

Generally speaking, most websites have XSS filters, which use the keyword blacklist strategy and the regular expression to filter the input data [9]. Although it's an effective method, we can still transform our attack vectors in various way.

There are three main approaches for the variation of attack vectors:

1. Original Vectors Encoding

With several ways of encoding mechanism, original vectors can be combined into variety forms malicious attack vectors. Therefore, the encoded vectors can bypass the filter and try to inject the script into the code. Here we focus on three main parts:

- HTML entity encoding
- Tag attribute encoding
- Special attribute data

2. Confuse the special characters in original vectors

- Change the letter case in tags: the letter case in the tags does not influence the operation of the code, but it can make the filter's keywords identification confused.
- Change slashes to spaces: most of the time, slashes play the same function as spaces in HTML, and it can bypass filter easily.
- Insert newlines or tabs: the browsers will skip these special characters in the operation of the code, meanwhile, the special characters may also bypass the filters.
- Change or add quotation marks: there are commonly four ways to indicate values in tags: single quotes, double quotes, anti-quotes, and no quotes. Using the mixed or unpaired quotes can easily lead to the result that certain events bypass the filter and become a new attribute itself successfully.

3. Recombine the original vectors:

Some attack vectors can be nested within other vectors, then they can bypass the filter. Many filters tend to ignore the content in attributes. As for some tags, due their own characteristics, it's easy to insert other labels into their interior and attributes. Such as <embed>, <iframe> and <object>. One of the methods is to add/delete the angle brackets from tags, resulting in a content overflow and then bypassing the filter. Such as the new attribute "srcdoc" in HTML5, you can insert a new label, triggering a XSS attack.

Experiment

Problems before the experiment

Nowadays, lots of websites installed anti-crawler system for the consideration of security and some moral and legal regulations. However, from the nature of the white hat workers, as well as under their moral constraints, i.e., not to collect user's name, address, telephone or E-mail, we can break through some restrictions for security reasons.

(1) Change of HTTP Request Header

In the HTTP protocol, when the client sends a request to the server, a set of attributes and configurations is transmitted. HTTP defines more than ten kinds of types of request header and attribute fields, the following seven are common:

Host	Connection	Accept	User-Agent	Referrer	Accept-Encoding	Accept-Languages
------	------------	--------	------------	----------	-----------------	------------------

The major difference between request sent by a browser and sent by Python Crawler would be as follow:

Attribute	User-Agent	Accept-Encoding
Python crawler content	Python-urllib/3.7	identity
Look into content normally	Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.8.1.11) Gecko/20071127 Firefox/2.0.0.11	gzip, deflate

Some websites can tell whether the request is from a crawler through this, then refuses to respond. Here we use Python's "requests" module [10] to modify the headers in the code. The Python code is as follows:

```
session=requests.Session()
headers={"User-Agent":"Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11",
"Accept":"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8"}
url="http://www.baidu.com"
req=session.get(url,headers)
```

(2) Cookie and access features

Some websites need to identify the user's identity by Session Tracking [11]. For example, the user must log in to browse certain pages. So we need to use Python's selenium module [12] to set the request cookie.

Some other websites, however, determine whether the request is from a crawler by observing the its interaction with the server. For example, loading pages so fast, the server will consider the request comes from a crawler, not human. So we can also set the request interval, random time is the most ideal. The Python code is as follows:

```
time.sleep(random.randint(3,10))
```

(3) The hidden input tags

Several websites set some hidden input tags to prevent crawlers from accessing. These input points, under normal circumstances, the is invisible to the human user, but can be found by crawlers. If these tags are automatically input content, it is reasonable for the server to believe that is a crawler visiting, rather than a person. In general, there are three ways to hide the page input tags:

- Set the label attribute to "hidden"
- Set the CSS property "display: none"
- Move the input field outside of the screen, while hiding the scroll bar

To deal with this, we need to use the Python selenium module [12] is_displayed () function to scan the page first.

Test result

In order to prove the effectiveness of our crawler system, we chose two well-known portals for our scanning test. Meanwhile, we operated the system with two popular vulnerability mining tools: Acunetix and XSSer and an open source crawler tool Blue Leech for comparison. The first portal has 6112 valid URLs, and the second portal has 21091. (We do not disclose the name of the site, given privacy.)

Our experiment is based on the platform of Linux Ubuntu 16.04.2, the Python version is 3.5.3. The result as following:

	Acunetix	XSSer	Blue Leech	Our Tools
#1Portal Scanning	3 hours 31 mins	3 hours 20 mins	4 hours 5 mins	3 hours 9 mins
#1Portal Vulnerabilities	23	16	-	21
#2Portal Scanning	17 hours 21 mins	15 hours 20 mins	20 hours 57 mins	15 hours 19 mins
#2Portal Vulnerabilities	12	9	-	12

The above experimental data proves that our crawlers, which is designed for XSS vulnerability mining, has a high efficiency. More important, while our tools is running, the system resource occupancy rate is relatively small. This is one of the important advantages of our tool.

Conclusion

In this paper, we proposed a dynamic parallel crawler system, which is designed for XSS vulnerability detection, and introduced the important modules in detail, including crawler controller module, parallel duplicate checking module, DNS resolving module and so on. Finally, we tested the XSS vulnerability detection tool, indicating the tool has certain advantages in scanning websites, which involves speed and system resource occupancy. However, XSS attacks means have been emerging in an endless stream, not only based on HTML, but also on Flash, ActiveX or Silverlight. This brings the XSS vulnerability mining endless challenges.

References

- [1] Symantec Internet Security Threat Report: Trends for July December 2007 (Executive Summary)
- [2] Top 10 2013 [Online]: https://www.owasp.org/index.php/Top_10_2013
- [3] Priti Singh, Kirthika Thevar, Pooja Shetty, Bushra Shaikh. Detection of SQL Injection and XSS Vulnerability in Web Application. International Journal of Engineering and Applied Sciences, March 2015.
- [4] Mini Singh Ahuja, Dr. Jatinder Singh Bal, Varnica. Web Crawler: Extracting the Web Data. International Journal of Computer Trends and Technology, volume 13 number 3, Jul 2014.
- [5] Andrew S. Tanenbaum, Herbert Bos. Modern Operating Systems. Mar 20th, 2014.
- [6] James Kurose, Keith Ross. Computer Networking: A Top-Down Approach (7th Edition) May 6, 2016.
- [7] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, Robert Morris. DNS Performance and the Effectiveness of Caching.
- [8] Sasu Tarkoma, Christian Esteve Rothenberg, Eemil Lagerspetz. Theory and Practice of Bloom Filters for Distributed Systems.
- [9] https://www.owasp.org/index.php/Testing_for_Stored_Cross_site_scripting_%28OTG-INPVAL-002%29
- [10] <http://docs.python-requests.org/en/master/>
- [11] Pranjali Gondane, Dinesh. S. Gawande, R.D. Wagh, S.B. Lanjewar, S. Ugale. Securing Web Application from SQL Injection & Session Tracking. International Journal of Engineering Science and Innovative Technology. Volume 2, Issue 3, May 2013.
- [12] http://www.seleniumhq.org/docs/03_webdriver.jsp