An Efficient Black-Box Vulnerability Scanning Method for Web Application

Haoxia Jin[1], Ming Xu[1(✉)], Xue Yang[1], Ting Wu[1], Ning Zheng[1], and Tao Yang[2(✉)]

[1] Internet and Network Security Laboratory, Hangzhou Dianzi University, Hangzhou, China
{151050013,mxu,153050004,wuting,nzheng}@hdu.edu.cn
[2] Key Lab of the Third Research Institute of the Ministry of Public Security, Shanghai, China
yangtao@stars.org.cn

Abstract. To discover web vulnerabilities before they are exploited by malicious attackers, black-box vulnerability scanners scan all the web pages of a web application. However, a web application implemented by several server-side programs with a backend database can generate a massive number of web pages, and may raise an unaffordable time consuming. The root cause of vulnerabilities is the mal-implemented server-side program, instead of any certain web pages that generated by the server-side program. In this paper, an efficient black-box web vulnerability scanning method – handler-ready – is proposed, which highlights the scanning on the server-side programs – *handlers* – rather than concrete web pages. Handler-ready reduces the HTTP requests of massive web pages to a small number of *handlers*, and gives the *handlers* an even chance of being scanned. Therefore, the handler-ready can avoid being stuck with massive web pages that generated by the same *handler* when scanning. The experimental result shows that the proposed scanning method can discover more vulnerabilities than traditional methods in a limited amount of time.

Keywords: Web application · Black-box vulnerability scanner

1 Introduction

Web applications are the most popular way of delivering services via the Internet. The complexity of modern web application has caused massive vulnerabilities in web applications, and, in fact, the number of reported web applications is growing sharply [1].

Web vulnerabilities threaten the security and privacy for both citizens and enterprises. For example, an attacker exploited a vulnerability of CSDN's (China Software Developer Network) website in Apr. 2010, and published the database in Dec. 2011, which contains over 6 million user informations [2].

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2018
I. Romdhani et al. (Eds.): CollaborateCom 2017, LNICST 252, pp. 444–455, 2018.
https://doi.org/10.1007/978-3-030-00916-8_42

An approach for fighting security vulnerabilities is to discover defects before malicious attackers find and exploit them by conducting automated black-box vulnerability scanning. Black-box vulnerability scanners observe the application's output in response to a specific input, and verify whether a vulnerability exists.

However, virtually any website that serves content from a database use one or more server-side programs to generate pages on the website, leading to a site considering of several clusters of pages, each generated by the same server-side program [3]. Taking stackoverflow.com for example, which has over 13 million web pages. A thorough black-box scanning against 13 million web pages is impossible and unnecessary. A pre-defined maximum pages to be crawled may neglect some of the functionalities of web application not being scanned. Alternatively, the impact of the massive server-side generated web pages on efficiency can be mitigated by our method.

In addition, web applications are released or updated with new features all the time, which may also introduce new vulnerabilities. It is necessary to discover their vulnerabilities with a rapid reaction.

In this paper, we propose the method – handler-ready – to improve the efficiency of black-box web vulnerability scanning against the massive automatically generated web pages under the time-restrict situation. Handler-ready is based on the motivation that the web pages generated from the same server-side program have similar patterns of their HTTP requests, and scanning on a small number of web pages could find vulnerabilities as much as scanning on all the web pages.

In summary, the main contributions of this paper are following:

- An efficient scanning method. It avoids being stuck in massive pages that related to the same server-side handler.
- Proposing the *sharing-the-same-handler problem* and the relevant concepts.
- A method of the handler learning along with the method of request sampling.
- An evaluation of both efficiency and effectiveness of the proposed method under the time-restrict situation. The experimental result promisingly shows that the efficiency is improved with no decrease in effectiveness.

The rest of the paper is organized as follows. We discuss related work in Sect. 2. In Sect. 3, we describe proposed handler-ready method along with its important formulations for *request* and *handler*. The evaluation of various scanning methods for effectively discovering web vulnerabilities are presented in Sect. 4.

2 Related Work

Another approach for fighting web application vulnerabilities is using white-box vulnerability scanners, which requires a web application's source code or target code. However, white-box scanners are commonly programming-language-specified, which reduce the scope of target web applications. In addition, there is the problem of substantial false positives [4]. Finally, the source code or target

code of the applications itself may be unavailable. In contrast, black-box vulnerability scanners observe the application's output in response to a specific input to verify the existence of vulnerabilities.

Automated black-box web application vulnerability scanning has been a hot topic in research for many years. A number of tools have been developed to automatically discover vulnerabilities in web applications, produced as academic prototypes [5,6], and open-source projects, such as skipfish, w3af, and OWASP Zed Attack Proxy.

The evaluation of [7] acknowledges challenges of web application vulnerability scanning in depth. [8] makes a comparison of the efficiency and effectiveness of vulnerability discovery techniques. A common theme of web vulnerability scanning is to improve effectiveness. [9,10] emphasizes the importance of crawling and application's internal states, and introduces a state-aware scanning method, which captures the web application's state changing, to find more vulnerabilities under different states. Different with these works, our aim is improving the efficiency, while not downgrading the effectiveness.

3 Method: Handler-Ready

Traditional scanning method takes following steps to scan a web application:

1. **Crawling.** The crawling process collects the target application's web pages automatically or manually. The automatic crawling often fails to trigger the AJAX requests which dominated by the client-side dynamic codes. Therefore, a common practice is manually browsing the target application first, then launching the automatic crawling.
2. **Scanning.** The scanning process constructs and sends some sophisticated HTTP requests to a collected web page's every injection point (such as a text input box, a parameter inside the URL) for every type of vulnerabilities. After the HTTP request received and processed by the target application, the scanning process receives the HTTP response of its constructed HTTP request, verifies whether a certain type of vulnerability has been found.
3. **Reporting.** The discovered vulnerabilities along with the relevant information are represented to the user.

The challenge is that the scanning processes often costs unaffordable time consuming, due to the massive number of web pages collected in the crawling process. These pages are generated by the very same server-side programs [3]. However, the number of web application's handlers is limited despite the massive number of server-side generated web pages. In addition, the duplicated web pages may reports spurious vulnerabilities that related to the same root cause [10]. Therefore, we introduce our efficient scanning method, handler-ready, which does two extra steps before scanning launched:

1. **Handler-Learning.** It reduces the HTTP requests of a massive number of web pages to a small number of *handlers* by utilizing the state-of-art frequent pattern mining algorithm.

2. **Request-Sampling.** It generates a partial sequence of HTTP requests for scanning such that the *handlers* have even chance of being scanned and avoid being stuck.

The steps of the proposed handler-ready method are shown in Fig. 3. Only two steps are inserted into the traditional steps. Therefore, handler-ready is ready to plug into existing scanners (Fig. 1).

Fig. 1. Handler-ready steps for vulnerabilities scanning.

The traditional method can also be interpreted by the proposed method, whose "handler-learning" treats every web page come from the same handler, and whose "request-sampling" just sample every web pages for scanning. The commonly used setting of "maximum children to crawl", which modifies the behavior of the crawling process, can also be a special "request-sampling" that samples a web page unless the number of its siblings sampled in the same directory not exceeds the threshold of "maximum children to crawl". Therefore, in our evaluation, we treat all traditional methods as different "request-sampling" methods, and compared them with the proposed method.

3.1 Modeling: The *Request* and *Handler*

In this section, we define the *request* and the *handler*, along with their *relations*.

Definition (Request). A *request* u is a map $u : K \to V$ from the key-set K to the value-set V. The set of all possible *requests* is denoted by U.

Where K is the set of keys $\{k_{\texttt{method}}, k_{\texttt{scheme}}, k_{\texttt{port}}\} \cup \{k_{\texttt{host,1}}, ..., k_{\texttt{host,h}}\} \cup \{k_{\texttt{path,1}}, ..., k_{\texttt{path,p}}\} \cup \{k_{\texttt{qs,}*}\}$. V is the set of all values of any possible HTTP requests, augmented with token $\perp$ that denotes the empty value. In describing any request u, u maps the keys of K to the corresponding values occurred in the original HTTP request, or the default empty value $\perp$ unless the corresponding values are not exist.

Example 1. Given a request

$$u_1 = \texttt{GET http://example.com/foo/bar?hello=world},$$

it has following mapping:

·	$k_{\texttt{method}}$	$k_{\texttt{scheme}}$	$k_{\texttt{port}}$	$k_{\texttt{host,2}}$	$k_{\texttt{host,1}}$	$k_{\texttt{path,1}}$	$k_{\texttt{path,2}}$	$k_{\texttt{qs,hello}}$	·
u_1	GET	http	$\perp$	example	com	foo	bar	world	$\perp$

Notice that the *keys* $\{k_{\texttt{host,1}}, ..., k_{\texttt{host,h}}\}$ are encoded reversely to the domain name's segments of a web application's for the alignment of the multiple subdomains that the web application may employ.

Definition (Handler). A *handler* r is a map $r : K \to W$, mapping each key in K to a value in W. Let R be the set of all possible *handlers*. Where W is the set of values V extended with regular expression characters. For simplicity, in this paper, we only consider $W = V \cup \{\star\}$, where $\star$ represents the wildcard character that can match any non-empty value in V. We define the *match* relation between *handlers* and *requests*, the *handler partial ordering* relation between *handlers*, and the *supports* of *handlers* as follows:

- **Match.** A *handler* r matches *request* u, written as $r \oplus u$, if for every *key* k, either $r(k) = \star$ or $r(k) = u(k)$.
- **Handler Partial Ordering.** A *handler* r' is less general than r, written as $r' \leq r$, if for every *key* k, $r'(k) \neq r(k) \Rightarrow r(k) = \star$. This is saying that a *handler*, which is more specific, is less general than other *handlers*.
- **Supports.** The support of r in U, denoted *supp*(r), is the set of *requests* $\{u|\forall u(r \oplus u) \wedge \neg\exists r'(r' \neq r \wedge r' \leq r \wedge r' \oplus u)\}$. This is saying that a *request* u belongs and only belongs to *supp*(r), if r is most preceding one of the *handlers* that *match* the *request* u.

Given sets of *handlers* R' and R'' that match all the *requests*, the sets $\{supp(r)|\forall r \in R'\}$ and $\{supp(r)|\forall r \in R''\}$ are partitions(i.e. equivalence classes) on U. Therefore, the *similarity* $\Theta(R', R'')$ is calculated by Jaccard Similarity: $\Theta(R', R'') = \frac{sum(\min(M_{R'}, M_{R''}))}{sum(\max(M_{R'}, M_{R''}))}$, where $M_{R'}, M_{R''}$ are the upper triangular matrixes of R', R'''s relation matrixes.

Our definition of *handler* coincides with the definition "pattern" or "script" of works about URL classification, such as [3,11]. In addition, our definition of *handler partial ordering* relation enables the ordering between *handlers*, which is not defined in the previous works. As show in Example 2, the *handler partial ordering* is useful when modeling *handlers* which seems very similar.

Example 2. Given following instances:

·	$k_{\texttt{method}}$	$k_{\texttt{scheme}}$	$k_{\texttt{port}}$	$k_{\texttt{host,2}}$	$k_{\texttt{host,1}}$	$k_{\texttt{path,1}}$	$k_{\texttt{path,2}}$	$k_{\texttt{path,3}}$	$k_{\star}$
u_1	GET	http	$\perp$	stackoverflow	com	questions	666	subrasi	$\perp$
u_2	GET	http	$\perp$	stackoverflow	com	questions	233	lol	$\perp$
u_3	GET	http	$\perp$	stackoverflow	com	questions	tagged	java	$\perp$
u_4	GET	http	$\perp$	stackoverflow	com	questions	ask	$\perp$	$\perp$
r_1	GET	http	$\perp$	stackoverflow	com	questions	ask	$\perp$	$\perp$
r_2	GET	http	$\perp$	stackoverflow	com	questions	tagged	$\star$	$\perp$
r_3	GET	http	$\perp$	stackoverflow	com	questions	$\star$	$\star$	$\perp$

On the stackoverflow.com, r_1 represents a web page for asking a new question; r_2 represents the web pages about questions tagged with the value indicated by $k_{\texttt{path,3}}$; and r_3 represents a question specified by $k_{\texttt{path,2}}$ and $k_{\texttt{path,3}}$. They have similar URL formats, but represent totally different functionalities.

There are $r_1 \leq r_3, r_2 \leq r_3, r_1 \neq r_2 \neq r_3$, $r_1 \oplus u_4$, $r_2 \oplus u_3$, and $r_3 \oplus u_i, i = 1, 2, 3, 4$. Although r_3 *matches* $u_{1,2,3,4}$, $u_{3,4}$ are not *supports* of r_3. The *supports* of r_1, r_2, and r_3 are $supp(r_1) = \{u_4\}$, $supp(r_2) = \{u_3\}$, and $supp(r_3) = \{u_1, u_2\}$.

The definitions of *handler partial ordering* and *supports* provide the ability to model multiple *handlers* that very similar to each other.

Assumption. We assume the web application is a *handler-dispatcher machine* that a *request* u is dispatched to the *handler* r if $u \in supp(r)$.

It should be note that our assumption *handler-dispatcher machine* meets the implementations of popular web servers, such as Microsoft IIS, Apache HTTPD, Nginx, as well as web application frameworks, such as SpringMVC, Struts, and Django. These implementations employ a regular expression rule based mechanism for request matching, and support the ordering between rules in the configuration file. All these features can be interpreted by our model. For web applications implemented by dynamic web pages and even static websites, the *handler-dispatcher machine* can still work.

Problem. The proposed problem is called *sharing the same handler problem*. Given a subset of *requests* U', find the a subset of *handlers* R', such that $\Theta(R, R') \rightarrow 1.0$.

Where R is the true set of *handlers* decided by the server-side programs or web server's configuration. Solving the *sharing the same handler problem* is the very aim of the handler-learning process of the proposed handler-ready method.

3.2 Handler Learning

As shown in Example 1 and Example 2, a bundle of *requests* is literally encoded into a multi-dimensional database. Therefore, a frequent closed itemset mining algorithm can be used to learn the patterns of *requests*. The handler-learning process follows the following steps:

1. **Itemset Encoding.** Transform a *request* u to itemset u' by Eq. (1).
2. **Frequent Pattern Mining.** Mine frequent itemsets $\{u'', ...\}$ from $\{u', ...\}$.
3. **Handler Constructing.** A frequent pattern u'' can be converted to a handler $r_{u''}$ by Eq. (2).

$$u' = \{e_k, v_{k,u(k)} | \forall k \in K, u(k) \neq \perp\} \tag{1}$$

$$r_{u''}(k) = \begin{cases} u(k), & \text{if } e_k \in u'' \wedge v_{k,u(k)} \in u''. \\ \star, & \text{if } e_k \in u'' \wedge v_{k,u(k)} \notin u''. \\ \perp, & \text{otherwise.} \end{cases} \tag{2}$$

In the Eq. (1), e_k represents the existence of *key* k, and $v_{k,u(k)}$ represents that the *value* of *key* k is $u(k)$. The choosing algorithm for frequent pattern mining is FPClose [12], which is a state-of-art algorithm, and its output is so called "closed itemset pattern". The closed itemset pattern is not included in another pattern having exactly the same occurrence. This feature make sure that the learned *handlers* are representative.

FPClose needs a parameter `minsup`, which is the minimum occurrence of a pattern. In our problem, FPClose with `minsup` $= 5\%$ produces patterns that not trivial. For large web application that have many functionalities, the `minsup` may be smaller to preserve more details. The *requests* that not matched by the FPClose produced patterns, are grouped together, and form a special *handler*, which is also used as a *handler* in the hereafter request-sampling process.

3.3 Request Sampling

In this section, we introduce our sampling method, which generates the sequence of *requests* for vulnerabilities scanning. Our sampling method needs following principles to do sampling:

- The chances of all *handlers* being sampled should be equivalent, as we don't know whether a given *handler* is more vulnerable than another.
- The chances of all *requests* of a *handler*'s *support* being sampled should be even. As we are agnostic with the values' meaning of *requests* even when their sets of keys are same, the only thing we can do is giving them the equivalent chance of being sampled.

To meet the above principles, the request-sampling take a sampling threshold p, a given set of *requests* U' and the learned *handlers* R' as input, and follows the following steps:

1. Shuffle the order of the *requests* in the set $supp(r)$ for all $r \in R'$.
2. Traverse every $r \in R'$ in turn, and sample a $u \in supp(r)$ for scanning.
3. Stop when the ratio p of *requests* of U' are sampled.

4 Evaluation

The purpose of the proposed method is to detect vulnerabilities of web applications. In this section, we evaluate the effectiveness of our handler-ready method in terms of the following questions:

- Whether efficiency improved by employing handler-ready?
- Whether more vulnerabilities discovered by handler-ready when scanning same number of objects that scanned by the traditional method?

Table 1. Evaluated applications.

Application	Description	Version	Lines
Gallery3	A photo hosting	3.0.2	26,622
Vanilla Forum v2.0	A discussion forum	2.0.17.10	43,880
WackoPicko	An intentionally vulnerable web application	2.0	900
WackoRESTful	WackoPicko's URL rewrite version	2.0	941
WordPress v3	A Blog hosting platform	3.2.1	71.698

4.1 Preparations

Table 1 provides an overview of the evaluated web applications. Most of them are evaluated in [10]. WackoRESTful is WackoPicko's URL rewrite version modified by us. All of them are varied in size, complexity, and functionality.

In order to collect the data sets, OWASP Zed Attack Proxy 2.6.0 is used to crawl and to scan pages on the web applications listed in Table 1. The sequence of *requests* scanned by the OWASP Zed Attack Proxy is the input of our handler-ready method, and is also the other sampling-methods' input. After the sequence of *requests* being scanned is collected, the handler learning process is started to extract handler information.

The statistics of handler learning against evaluated applications are shown in Table 2, and the labeling is made by us manually to calculate the *similarity*. The column "Similarity" shows the *similarities*, which are not very close to 1.0. These *similarities* is actually saying the ratios of the relations of the *requests* that dispatched to the same *handler* with 100% confidence. A bigger *similarity* may potentially introduce a higher scanning efficiency. Therefore, the proposed handler-ready can still improve the scanning efficiency against Vanilla Forum v2.0 despite its small *similarity*.

The column "Distribution" shows the distribution of $||supp(r)||/||U'||$ of the evaluated applications, and suggests that a minority of *handlers* could matches the largest portions of *requests*. The processes handler-learning and request-sampling of handler-ready are necessary to give all *handlers* even chance of

Table 2. The statistics of the evaluated applications after handler-learning.

Application	**Requests**	**Handlers**	**Similarity**	**Distribution**
Gallery3	1,710	38	0.5536	
Vanilla Forum v2.0	1,353	42	0.1988	
WackoPicko	144	19	0.4120	
WackoRESTful	276	22	0.4166	
WordPress v3	253	24	0.7884	
Total	3,736	145	N/A	0% 5% 10% 15% 20% 25% 30%

being scanned rather than the chances positively correlated to the number of matched *requests*.

4.2 Sampling Methods

To simulate the time-restrict situation, the following are evaluated sampling methods:

- **handler-ready$_p$.** The proposed method. The first ratio p of request-sampling's result are used for scanning.
- **DFS$_p$.** The first ratio p of the lexicographical order of *requests*.
- **owasp-zap$_p$.** The first ratio p of original sequence of *requests* scanned by OWASP Zed Attack Proxy.
- **max-children$_m$.** The sequence of *requests* scanned by OWASP Zed Attack Proxy when setting the parameter "maximum children to crawl" to m.

where p is the ratio of *requests* being scanned, and m is the maximum children to crawl under a certain directory.

4.3 Vulnerabilities Scanning Results

To evaluate the performance of the handler-ready for vulnerabilities scanning under the p and m, we conduct two sets of comparison experiments:

- Comparisons between handler-ready$_p$, DFS$_p$, and owasp-zap$_p$. The threshold p is ranging from 0.1 to 1.0.
- Comparisons between handler-ready$_{p_m}$ and max-children$_m$. Where p_m is the threshold related to m, such that $||\text{handler-ready}_{p_m}|| = ||\text{max-children}_m||$. Without loss of generality, the parameter m is ranging from 0 to 20.

handler-ready$_p$ vs. DFS$_p$ vs. owasp-zap$_p$. The vulnerabilities reported by three sampling methods under different threshold p are shown in Fig. 2. In most cases, the proposed handler-ready$_p$ performs better than other methods.

For Gallery3 and WordPress v3, the true positives are omitted, because they have no true positives found in this experiment. It is interesting that Vanilla Forum v2.0 has two reflected XSS vulnerabilities and three external redirect vulnerabilities discovered during our experiment. As all of these vulnerabilities are placed in the management module of the application, the damage of these vulnerabilities being exploited will be very harmful. The reflected XSS vulnerabilities can be exploited by a malicious user to collect another user's(web master's) credential.

The sampling method owasp-zap$_p$ is actually a breadth first traversing of *requests*. owasp-zap$_p$ and DFS$_p$ are often being stuck, for they often enter a directory with massive web pages derived by the same server-side handler. In contrast, the proposed handler-ready$_p$ does not.

Comparing with the traditional scanning methods, handler-ready has a 40% efficiency improvement against the evaluated web applications with no decrease in effectiveness.

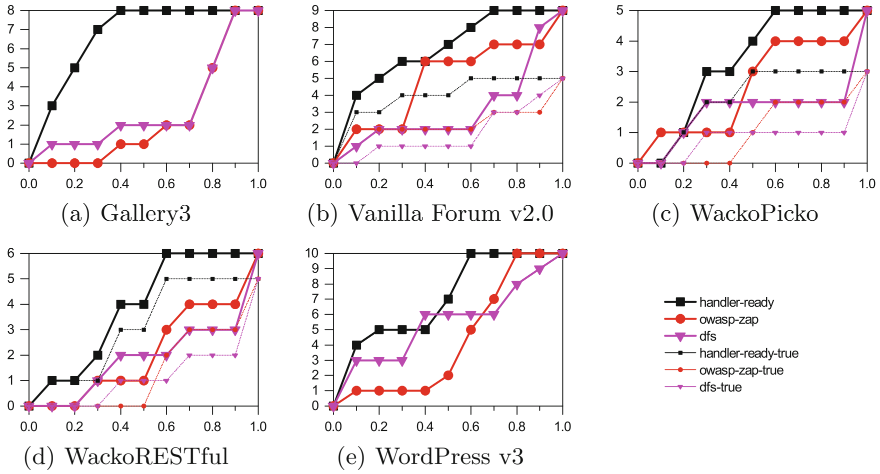

Fig. 2. Vulnerabilities reported under different threshold p. Handler-ready-true, owasp-zap-true, and dfs-true are three methods' true positives respectively.

handler-ready$_{p_m}$ vs. max-children$_m$. The vulnerabilities reported by two methods under different parameter m and m's related p_m are shown in Fig. 3.

The proposed handler-ready$_p$ performs better than "maximum children to crawl" method, i.e. max-children$_m$. As shown in Fig. 3(a), max-children$_m$ still neglects vulnerabilities that should be discovered when $m = 20$, which is quite large. Although handler-ready seems not superior than max-children against Vanilla Forum v2.0, the former can still find true positives earlier than max-

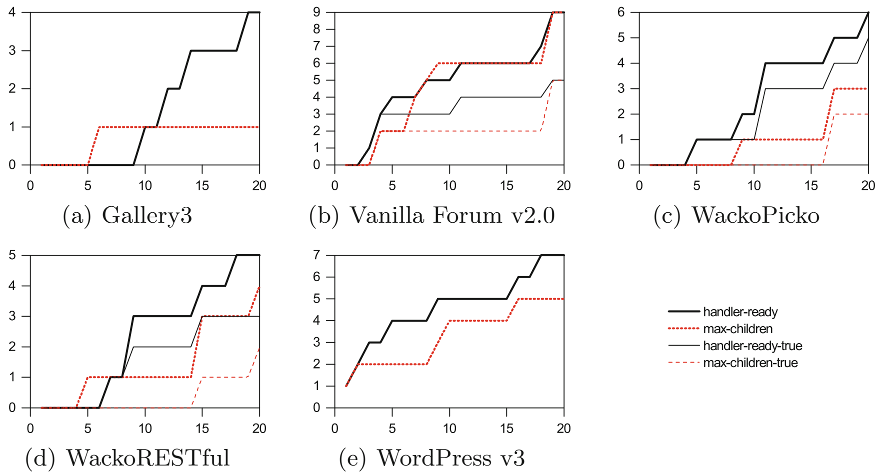

Fig. 3. Vulnerabilities reported under different parameter m and p_m. Handler-ready-true and max-children-true are two methods' true positives respectively.

children. The reason is that "maximum children to crawl" prunes the *requests* that have vulnerable children.

Under the time-restricted condition, handler-ready can discover more vulnerabilities than the commonly used "maximum children to crawl" approach with the same number of the object being scanned.

5 Limitation

Although the web applications nested in the web browser are unable to manipulate other request fields (such as Request Headers) to identify different handlers, web applications' developers still have chance to hide handler tokens in the Cookie or the payload of the POST request as mentioned in [13]. And HTTP-based APIs (such as API for mobile APP) may manipulate these fields to identify the handler and parameters. In the future, we will study how to model these fields and make our method fully-handler-ready.

6 Conclusion

New web vulnerabilities emerge quickly and threaten the security and privacy for both citizens and enterprises. Web applications update all the time to import new features, which may also introduce new vulnerabilities in the meanwhile. It is necessary to discover web vulnerabilities with a rapid reaction. However, traditional scanning methods do not scan quickly against web applications, which utilize server-side programs to generate massive web pages from a backend database.

The proposed method, handler-ready, learns the pattern of requests for the massive automatically generated web pages, and samples the requests that worth to scan under the time-restrict situation. Sampled requests are used to scan the web application. The experimental result shows that the proposed method can significantly improve the efficiency under the time-restricted condition.

Acknowledgements. This work is supported by the National Key R&D Plan of China under grant no. 2016YFB0800201, the Natural Science Foundation of China under grant no. 61070212 and 61572165, the State Key Program of Zhejiang Province Natural Science Foundation of China under grant no. LZ15F020003, the Key research and development plan project of Zhejiang Province under grant no. 2017C01065, the Key Lab of Information Network Security, Ministry of Public Security, under grant no C16603.

References

1. Martin, R.A., Christey, S.: Vulnerability type distributions in CVE. MITRE Report (2007)
2. China Software Developer Network (CSDN) leaked 6 million user information. http://www.williamlong.info/archives/2933.html

3. Blanco, L., Dalvi, N., Machanavajjhala, A.: Highly efficient algorithms for structural clustering of large websites. In: WWW, pp. 437–446 (2011)
4. Medeiros, I., Neves, N.F., Correia, M.: DEKANT: a static analysis tool that learns to detect web application vulnerabilities. In: ISSTA, pp. 1–11 (2016)
5. Felmetsger, V., Cavedon, L., Kruegel, C., Vigna, G.: Toward automated detection of logic vulnerabilities in web applications. In: USS, pp. 143–160 (2010)
6. Halfond, W.G.J., Choudhary, S.R., Orso, A.: Penetration testing with improved input vector identification. In: ICST, pp. 346–355 (2009)
7. McAllister, S., Kirda, E., Kruegel, C.: Leveraging user interactions for in-depth testing of web applications. In: RAID, pp. 191–210 (2008)
8. Austin, A., Holmgreen, C., Williams, L.: A comparison of the efficiency and effectiveness of vulnerability discovery techniques. IST **55**(7), 1279–1288 (2013)
9. Doupé, A., Cova, M., Vigna, G.: Why Johnny can't pentest: an analysis of black-box web vulnerability scanners. In: DIMVA, pp. 111–131 (2010)
10. Doupé, A., Cavedon, L., Kruegel, C., Vigna, G.: Enemy of the state: a state-aware black-box web vulnerability scanner. In: USS, pp. 523–538 (2012)
11. Hernndez, I., Rivero, C.R., Ruiz, D., Corchuelo, R.: CALA: classifying links automatically based on their URL. JSS **115**, 130–143 (2016)
12. Grahne, G., Zhu, J.: Fast algorithms for frequent itemset mining using FP-trees. TKDE **17**(10), 1347–1362 (2005). https://doi.org/10.1109/TKDE.2005.166
13. Shezaf, O.: Rest assessment cheat sheet. http://tinyurl.com/mkqd8br