ECE653 PROJECT REPORT

UNIVERSITY OF WATERLOO

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Automated Website Fuzzing with WFuzz

Xinyi Ma (20814843)
Jiahao Shi (20683205)
Shuoyuan Chen (20683201)

Date: August 20, 2020

Abstract

Fuzzing is an essential precautionary method to prevent cyber attacks against website hosts. Conventional website fuzzing tools such as WFuzz is laborious and requires enormous concentration. Therefore, we propose in this report an automated website fuzzer, AutoWFuzz, which scans the whole website and report all found errors automatically. Our tool achieves high coverage of errors in an efficient manner, saving website maintainers huge amount of time.

1 INTRODUCTION

Cyber-security has always been an issue of great concern in website development. Web applications are widely used for information exchange, but they are quite vulnerable. If the vulnerabilities are not detected and avoided, they will lead to serious consequences such as information leakage and injection attacks. In this way, it will be easy for hackers to get sensitive information and conduct Internet crime.

Although there are many ways like static analysis on source code to find vulnerabilities in the website, Website Fuzzing is the most widely used method to expose vulnerabilities in web applications. Fuzzing testing can generate a wide range of invalid and unexpected inputs, which are often overlooked by the developers, and monitors the application for exceptions. [2] The goal of fuzzing testing is to induce unexpected behavior of the web application. But we found that random fuzzing and manual fuzzing tools are not automated while fuzzing a website with thousands of URLs and will result in a huge workload. Therefore, a better method including fuzzing and analysis needs to be implemented for the issue.

Since manually testing a web application can be time-consuming and inefficient, we combine the fuzzing tool named WFuzz and python scripts to automate the processing of the fuzzing process. Wfuzz simply replaces any reference to the FUZZ keyword by the value of a given payload and the payloads given are the source of data. In this way, WFuzz allows any kind of input data to be injected in any field of the HTTP request and different components including parameters, forms, and files, which perform various security attacks using the payload word list dictionary provided inside WFuzz.

The most important part of our project is the automation of web application fuzzing. Since WFuzz is developed based on Python and the WFuzz documents provide detailed explanations to perform fuzzing using the python WFuzz package installed by pip and python scripts. Therefore, we will develop an automated fuzzing tool as an extension of the semi-automatic tool WFuzz to perform fuzzing and analyze the vulnerabilities detected in the process. Specifically, our extension will be able to scrape URLs on a host website and perform SQL injections and XSS injections to detect related vulnerabilities, as well as weak password vulnerability for websites that need login operation.

Our modified WFuzz, AutoWFuzz, is both efficient and accurate. AutoWFuzz scans the whole website and grab all information for fuzzing in few seconds, without anything left out. With the collected data, the fuzzing can be performed with hundreds of generated commands sending to the target server per minute. The whole process is completed without human interference, hence avoiding human errors and carelessness.

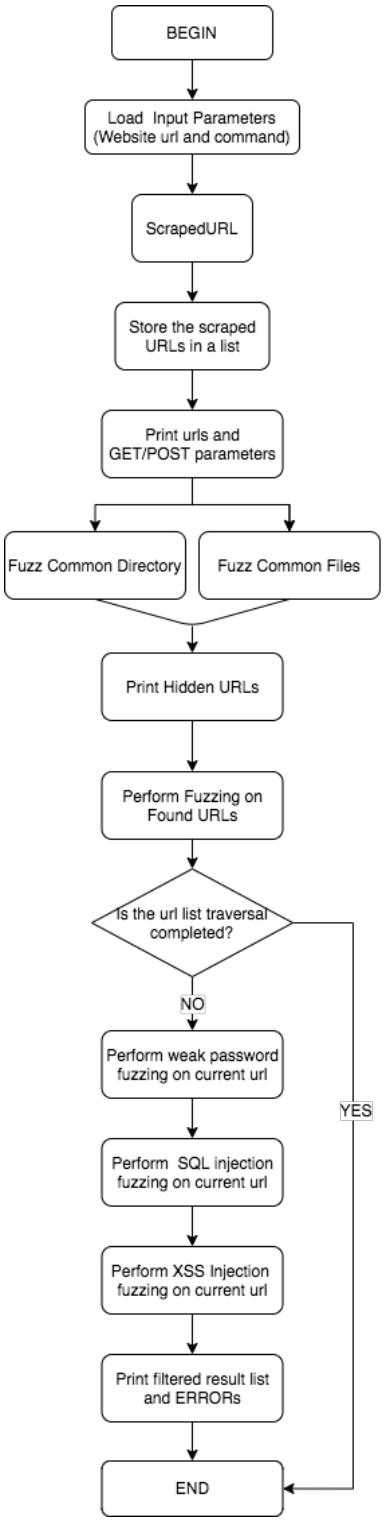


Figure 1: Flowchart of fuzzing algorithm.

Algorithm 1: AutoWFuzz

```
1: function MAIN(root_url)
2:    foundURLs, parameters ← WebsiteScraping(root_url)
3:    InformationLeakFuzzing(url, common_dir_payload, common_files_payload)
4:    for each url in foundURLs
5:       WeakPasswordFuzzing(url, parameters, weak_pass_payload)
6:       XSSInjectionFuzzing(url, parameters, xss_injection_payload)
7:       SQLInjectionFuzzing(url, parameters, sql_injection_payload)
8:    end for
9: end function

10: function WEBSITESCRAPING(url)
11:    found_URLs ← ScrapeLinks(url)
12:    GETparameters ← ScrapeGETMethod(url)
13:    POSTparameters ← ScrapePOSTMethod(url)
14:    return found_URLs,(GETparameters,POSTparameters)
15: end function

16: function XSSINJECTIONFUZZING(url, parameters, xss_injection_payload)
17:    output ← WFuzz(url, parameters, xss_injection_payload)
18:    output ← ClusterOuput(output)
19:    print output
20: end function

21: function SQLINJECTIONFUZZING(url, parameters, sql_injection_payload)
22:    output ← WFuzz(urls, parameters, sql_injection_payload)
23:    output ← ClusterOutput(Input)
24:    print output
25: end function

26: function INFORMATIONLEAKFUZZING(url, common_dir_payload, common_files_payload)
27:    found_common_urls ← WFuzz(urls, common_dir_payload, common_files_payload)
28:    hidden_urls ← find_hidden_urls(found_common_urls, urls)
29:    print hidden_urls
30: end function

31: function WEAKPASSWORDFUZZING(url, parameters, weak_passpayload)
32:    output ← WFuzz(url, parameters, weak_pass_payload)
33:    output ← ClusterOutput(output)
34:    print output
35: end function
```

2 Description of Method

In this section, the detailed automated web fuzzing process and the pseudo code for each part are presented. The fuzzing steps and a flow chart is presented in subsection 2.1. The fuzzing starts with website scraping to look for sub URLs, and URL parameters and POST parameters corresponding to each URL, which is discussed in subsection 2.2. Subsection 2.3 presents the web vulnerability detection methods, including potential leak of information, weak password, SQL injection and XSS injection. At last, subsection 2.4 introduces how the output of fuzzing is presented and how to extract useful information related to vulnerabilities from the output.

2.1 Fuzzing Steps

The pseudo code of the fuzzing algorithm is shown in Algorithm 1. Fig.1 presents the flowchart of the entire algorithm. The fuzzing process contains two main parts, web scraping and fuzzing for vulnerabilities. First, the home URL of the website to test and command options are loaded. Then, the web scraper is called to recursively look for subURLs, and the URL and POST parameters on each URL. After that, the InformationLeakFuzzing is performed by looking for directories and files with common names. With this, the hidden URLs, that are found by the web scraper, might be revealed. Next, the algorithm performs fuzzing on the list of URLs found in the scraping process. For each URL in the list, the algorithm checks if it is a login page and decide whether a weak password fuzzing is needed. Then, the fuzzing on both URL parameters and POST parameters are performed to detect SQL injections and XSS injections. The fuzzing results are analyzed and printed along the process.

2.2 Website Scraping

The XML codes embedded in one web page contains useful information, including links to other pages and sockets to the back end of the site. They can be extracted using python XML parsing package 'BeautifulSoup'. A python script is created to scrape all required data of the website. By recursively browsing all the internal URLs of the website, a sitemap containing all the open pages can be obtained. Meanwhile, all the GET and POST methods coded within the website are retrieved, granting the opportunity of automatically probing the back end implementation by sending customized URLs to the server. One limitation of website scraping is that the hidden files, which has never been pointed by any link, will not be detected, thus requiring random fuzzing on URL.

2.3 Vulnerability detection

2.3.1 Cross-Site Scripting (XSS) Injection Detection

Cross-site Scripting (XSS) injection refers to malicious codes injected to the website to either break into the host machine or attack unsuspected users.[3] It is a widespread issue across the Internet due to the fact that a big portion of data flowing to Web applications is through unsafe methods such as web requests. Malicious scripts included in dynamic content are not validated and then sent to web users, causing problems.

XSS attacks can generally be categorized into stored and reflected XSS attacks, where the latter one is the dominant type. For reflected XSS attacks, the website server is hijacked and malicious codes would be executed on user's end if certain links are clicked or specific forms are submitted.
Intentional XSS injections are performed on the target website to detect any possible vulnerabilities. First, the payload is generated based on the file wfuzz-master/wordlist/Injections/XSS.txt. Then WFuzz is applied on the scraped URLs of the host site using the accompanying list of URL and POST parameters. The responses are collected in FuzzResultCluster instances, and distinct outputs and errors are picked out.

2.3.2 SQL Injection Detection

For a web application, a SQL query is a way to talk to the back-end database. SQL injection means malicious queries accessing unauthorized part of the database or even perform harmful commands.
In our program, SQL injection tests are performed on target web pages with URL parameters or POST parameters. First, payloads based on wfuzz-master/wordlist/Injections/SQL.txt are created. They are then combined with the tested URL to form WFuzz commands. After fuzzing is finished, the responses are collected and clustered, leaving unique responses and error messages for analysis.

2.3.3 Weak Password Detection

The weak password has been a threat to both users and web applications. The term 'weak' does not only refers to the length and characters of the password, but also means guessability. With a weak password, the attacker can simply use brute force to fuzz the form parameters and obtain permission to the site.
AutoWFuzz generates usernames and passwords combinations using wfuzz-master/wordlist-/othres/common_pass.txt, which contains some common passwords or usernames that a user may use. They are then employed in fuzzing on the pages containing login interface to uncover weak password vulnerabilities.

2.3.4 Potential Leak of Information

The potential leak of information is also called File Inclusion. When file inclusion occurs, the attacker may be able to read, download hidden files and even upload files to the server. This will lead to exposure of sensitive information since the server might be misconfigured and running in overly high privileges.[5]
In our project, the detection of a potential leak of information is performed through common directory fuzzing and common file fuzzing. Specifically, to check whether some hidden files are included in the web application, we fuzz the scraped URLs by adding common file and directory names to the end of the URL. The exposed files and directories, which have not been detected by scraping, are highly likely to be sensitive information and exclusive for authorized personnel. It indicates the existence of file inclusion vulnerability on the tested website.

2.4 Output of fuzzing

As the fuzzing process is based on Wfuzz, the original output is in the format of the output of Wfuzz. Fig.2 shows a part of the output of Wfuzz when fuzzing SQL injections on the URL http://testphp.vulnweb.com/artists.php/artists.php?artist=FUZZ. The entry in the output corresponds to a payload, which is the string used to replace the word "FUZZ" in the URL. The output consists of the HTTP response code and the number of lines, words, and characters in the HTTP response message, and etc. There are many options to change the content of output. For example, one can choose to show the payload of each entry. From the sample output, we can see that many entries have the same HTTP response, as the code, lines, words and chars are the same. Here are two kinds of responses, which are labeled by the red rectangles. One can extract useful information by identifying different responses and manually visit a URL for each type of response. By comparing the content on the website or checking the actual response message using tools like Wireshark, one can find how different payload can invoke different responses and may be able to find vulnerabilities induced by certain payload.

2.4.1 Improvements on Wfuzz Output

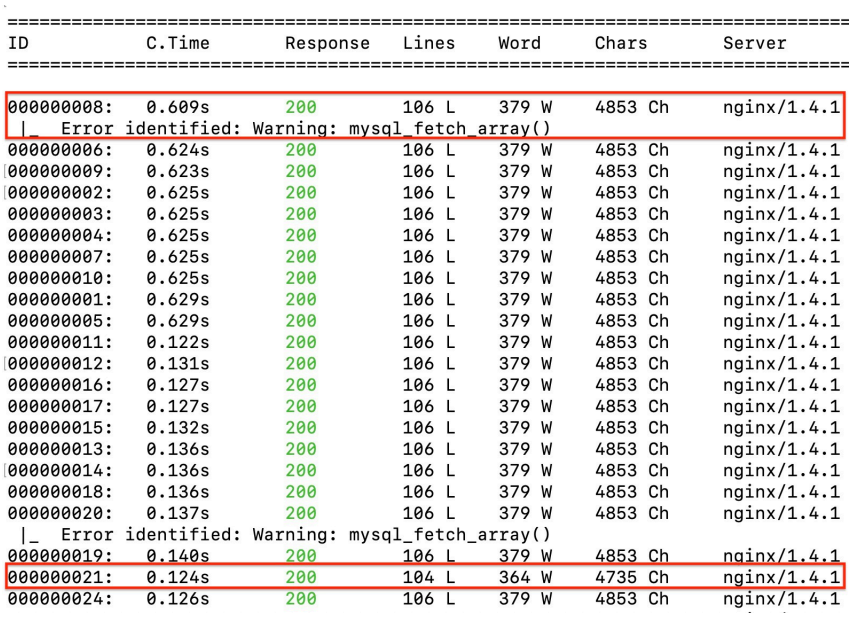
However, it is normally time-consuming to identify different types of responses from hundreds of entries. Thus, in our process, the fuzzing results are distributed into different clusters. Results in the same cluster has the same response code, lines, words, and chars. The output only shows one entry for each cluster. This clustering method greatly simplifies the output and reduce the manual involvement needed for locating vulnerabilities, thus increasing the automation of fuzzing.
The sample output in Fig.2 also shows two error messages related to MySQL errors. The error detection is enabled by utilizing a Wfuzz plugin, but it is found that the detection misses a great number of errors. For example, most entries in the sample output have the same response, but only two of them show there are errors. In our process, we scan the content part of the HTTP response message to look for common SQL error messages, such as "Warning: mysql_fetch_array()", "Error 1064", and "Error 1054". In the final output, all result entries with errors are listed, even though they might be in the same cluster.

3 Experiment Results

In this section, we discuss the experiment setup for testing our tool in subsection 3.1. The benchmark of the tool on websites with vulnerabilities is discussed in subsection 3.2. The comparison between AutoWFuzz and a customized random fuzzer is presented in subsection 3.3.

3.1 Experiment Setup

The AutoWfuzz is built on Wfuzz. To run it, one need to install Wfuzz and all its dependencies. The wordlist in the master folder of Wfuzz is also needed to generate fuzzing payloads. In addition, the BeautifulSoup package is required for web scraping. The tool is written in Python 3. In experiments, AutoWfuzz is run on a MacBook Pro, which has a 6-core Intel Core $i7$ processor, 16 GB memory and an SSD.



```
=====================================================================
ID           C.Time       Response   Lines    Word     Chars       Server
=====================================================================

000000008:   0.609s       200        106 L    379 W    4853 Ch     nginx/1.4.1
 |_  Error identified: Warning: mysql_fetch_array()
000000006:   0.624s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000009:   0.623s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000002:   0.625s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000003:   0.625s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000004:   0.625s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000007:   0.625s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000010:   0.625s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000001:   0.629s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000005:   0.629s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000011:   0.122s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000012:   0.131s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000016:   0.127s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000017:   0.127s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000015:   0.132s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000013:   0.136s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000014:   0.136s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000018:   0.136s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000020:   0.137s       200        106 L    379 W    4853 Ch     nginx/1.4.1
 |_  Error identified: Warning: mysql_fetch_array()
000000019:   0.140s       200        106 L    379 W    4853 Ch     nginx/1.4.1
000000021:   0.124s       200        104 L    364 W    4735 Ch     nginx/1.4.1
000000024:   0.126s       200        106 L    379 W    4853 Ch     nginx/1.4.1
```

Figure 2: Sample output of Wfuzz.

3.2 Benchmark

To benchmark the AutoWfuzz, we test it on several vulnerable websites that are designed for testing web vulnerability scanners. The first website is http://testphp.vulnweb.com, and the fuzzing results are shown in Table 1. We successfully find a list of hidden directories and files under the home page, part of which are shown in the table. Under the admin directory, we find a create.sql file, which contains source sql codes to create a database and a list of tables, which is a serious leak of back end data. A weak username and password pair (test, test) is detected. A large number of SQL injections are found on two webpages. The results of SQL injections includes generated MySQL error messages, unexpected extraction of data from database, and an unintentional login in the userinfo page. XSS injections are detected by verifying the inserted malicious scripts are successfully executed.

The second website for test is http://www.webscantest.com/. The website contains a list of subpages, each of which is specially designed to have a specific vulnerability. The fuzzing results are shown in Table 2. All subpages under the directory of datastore are designed for testing SQL injections, while the last URL in the table is designed for XSS injections. The AutoWfuzz has found vulnerabilities on all the pages except for one URL.

By testing AutoWfuzz on vulnerable websites, we find it has high coverage on web vulnerabilities. The running time for testing a website is within a few minutes. AutoWFuzz is shown to have both high accuracy and efficiency.

3.3 Compared to Random Fuzzing

To further benchmark the AutoWfuzz, we make a comparison between it and a customized random fuzzer. An original idea to perform random fuzzing is to add random strings to home URLs, but almost all the modified URLs are not found. Thus, to perform a meaningful comparison,

Vulnerability	URL	Sample Payload	Auto	Random
Leak of Information	../admin/create.sql ../index.bak ...	N/A	Y	N
Weak Password	../login.php	uname=test&pass=test	Y	N
SQL Injection	../artists.php/artists.php	?artist==%20– ?artist=0%20or%201=1	Y	Y
	../userinfo.php	uname=–'&pass=–'	Y	Y
XSS Injection	../hpp/	?pp=';alert... (a long string)	Y	N

Table 1: Fuzzing results on testphp.vulnweb.com. Auto and Random stands for AutoWfuzz and RandomFuzzer respectively. Y stands for vulnerability detected, while N is not detected.

URL	Sample Payload	Auto	Note	Random
../search_by_id.php	name='	Y	SQL Error	Y
	name=0 or 1=1	Y	data leak	
../search_by_name.php	name='%20	Y	SQL Error	Y
	name=' or 0=0	Y	data leak	
../search_get_by_id.php'	?id='	Y	SQL Error	Y
	?id=0%20or%201=1	Y	data leak	
../search_get_by_name.php	?name='	Y	SQL Error	Y
	?name='%20or%20'x'='x	Y	data leak	
../search_double_by_name.php	name='	Y	SQL Error	Y
	name=' or 0=0	Y	data leak	
../search_single_by_name.php		N		N
../search_by_statement_index.php	?id=\x27\x4F\x52 SELECT *	Y	data leak	N
../aboutyou2.php	<script>alert("WXSS")</ script>	Y	XSS Inject	N

Table 2: Fuzzing Results of SQL Injection on www.webscantest.com/datastore, except for the last entry, which is tested for XSS injection under www.webscantest.com/crosstraining. Y stands for vulnerability detected, while N is not detected.

we utilize the web scraper in AutoWfuzz to look for sub URLs and parameters. Then, a random input file is generated, which is composed of 40 random numbers, 40 random words, and 40 random strings that contain punctuation and symbols. Afterwards, Wfuzz is used to perform fuzzing on each found URL with the random input file as payloads.
Random fuzzing is perform on the same websites used for testing AutoWfuzz. The results are also shown in Table 1 and 2. It is found that random fuzzer can detect SQL injections by inducing SQL query errors, but it cannot leads to leak of data from databases. This is reasonable since SQL query errors might be induced by invalid inputs, which can be easily generated in the random input files, but causing data leak needs the input to match with specific SQL languages. AutoWfuzz uses different wordlists as payloads to detect different vulnerabilities, while random fuzzer only uses the random inputs. This also causes that the random fuzzer cannot detect vulnerabilities such as detection of hidden files or directories, weak password, and XSS injections.

4 Conclusions and future work

The AutoWFuzz is capable of performing multiple types of tests on the target website automatically. It supports a high variety of errors and recursively test all pages of the site in a time-efficient manner. Lastly, it filters out unique errors for fast diagnosis.
For future work, meaningful and readable error report to assist in fixing bugs and adding support for detecting other more vulnerabilities are good directions.

References

[1] WFuzz: The Web Fuzzer `https://wfuzz.readthedocs.io/en/latest/` pages

[2] Fuzzing Web Applications Hunting for XSS and SQLi effortlessly and automatically `https://medium.com/swlh/fuzzing-web-applications-e786ca4c4bb6` pages 2

[3] Cross Site Scripting `https://owasp.org/www-community/attacks/xss/` pages 5

[4] What is SQL injection? `https://www.veracode.com/security/sql-injection` pages

[5] File Inclusion Vulnerabilities `https://www.offensive-security.com/metasploit-unleashed/file-inclusion-vulnerabilities/` pages 6

[6] Weak Password Vulnerability `https://www.sciencedirect.com/topics/computer-science/weak-password` pages