FUZZ TESTING DATA GENERATION FOR NETWORK PROTOCOL USING CLASSIFICATION TREE

Rui Ma[1], Wendong Ji[1], Changzhen Hu[1], Chun Shan[1], Wu Peng[2]

[1]School of Software, Beijing Institute of Technology, Beijing, China
[2]China Academy of Electronics and Information Technology, Beijing, China
mary@bit.edu.cn, jiwendong09@163.com, {chzhoo, sherryshan}@bit.edu.cn, pw_bit@126.com

Keywords: Network Protocol Fuzzing, Test Data Generation, Classification Tree, Heuristic Operator.

Abstract

Aiming at the test data generation, which is one of the key issues in the network protocol fuzzing, this paper presents a new method on the basis of classification tree and heuristic operator. The method firstly builds up a protocol classification tree divided into 4 layers: target network protocol, protocol fields, attributes belonging to all fields, and attribute values. In order to reduce the scale of fuzz testing data, heuristic operators are defined to remove useless items from value sets of attributes. And then the test data for each protocol field was obtained by doing Cartesian product between value sets of attributes. The fuzz testing data for target network protocol is finally generated by replacing the corresponding field in the protocol with its fuzzing data one by one. Experimental results indicate that our method could successfully detect vulnerabilities, while dramatically reduce the number of test data and highly improve the quality of test data.

1 Introduction

With the rapid development of computer technology, software functions become more and more powerful. In order to ensure the quality and security of software, higher requirements of security testing are demanded. Fuzzing is an effective approach to detect security vulnerabilities, and has become one of research hotspots in the field of information security. Furthermore, network protocol fuzzing is the most attractive fuzzing type in that it could detect some higher-risk vulnerabilities[1].

Fuzzing is a method for discovering faults in software by providing unexpected input and monitoring exceptions[2]. It is a typical process that repeatedly supplies data to target software for processing. Therefore, test data generation is a very critical part of fuzzing and could directly affect the efficiency of fuzzing. High quality test data could detect more potential vulnerabilities in a shorter period of time[3].

There are 3 common approaches for fuzz testing data generation: pre-generated, mutated-based and generation-based[2]. In order to obtain pre-generated test cases, testers should analyse the protocol specification, and then add test cases manually. Its advantage is relatively easy to be implemented. However, it relies heavily on experiences and skills of testers, and test cases could not be intelligently changed for different targets[4]. Its typical representative is PROTOS. Mutated-based test cases are generated by mutating a portion of the normal input data. Now automatic generating methods, including random and brute force, have commonly replaced early manually methods. The random method[5] assigns a random value to the input data which is selected to be mutated. The brute force method[6] attempts a large amount of data as the input value. The advantage of mutated-based is relatively low labour cost. However, the number of its test cases will show "explosive" growth and mass redundant data might increase execution time of testing. Generation-based approach[7] should acquire the details of protocol specification. It needs construct profile or grammar to describe the protocol specification and then obtains test cases by dynamically parsing grammar or profile. It relies on the ability of testers for parsing protocol specifications. The key point focuses on the definition of grammar or profile.

For the problems of existing fuzz testing data generation, and considering the features of network protocols, this paper proposes a new test data generation method for network protocol fuzzing using classification tree[8,9] and heuristic operator[10]. First of all, it breaks the target protocol down into fields and uses them to make up a tuple. Secondly, it builds up a protocol classification tree after obtaining attribute sets for each field as well as value sets for each attribute. Next, it defines heuristic operators so as to remove useless items from value sets of attributes. Then, it obtains the fuzz testing data for each field in the tuple by doing Cartesian product between its reduced attribute value sets. At last, it obtains the fuzz testing data for target protocol by replacing the corresponding field in the tuple with its fuzzing data one by one.

In order to verify this method, we selected Peach acts as the fuzzing framework with FTP being the target protocol. And the custom mutator is used as well as heuristic operators are extracted with the help of IDA Pro. Experimental results indicate that the method could successfully detect vulnerabilities, while dramatically reduce the number of test data and highly improve the quality of test data.

The remainder of this paper is organized as follows. Section II introduces the details of test data generation using classification tree and heuristic operator, and Section III

presents our experimental processes and results. Finally, we conclude the paper in Section IV.

2 Test Data Generation using Classification Tree and Heuristic Operator

This paper proposes a new method to solve the problem of larger quantity and lower quality of test data in network protocol fuzzing. The contributions of this work are outlined as follows:

- Builds up a protocol classification tree divided into 4 layers: target network protocol, protocol fields, attributes belonging to all fields, and attribute values.
- Defines heuristic operators so as to remove useless items from sets of attribute value, so that the scale of fuzz testing data will be reduced during the testing.
- Obtains the fuzz testing data for each field separately by doing Cartesian product between sets of attribute value.
- Obtains the fuzz testing data for target network protocol through replacing the corresponding field in the protocol with its fuzzing data one by one.

2.1 Protocol Classification Tree

A protocol classification tree consists of four layers. The first layer is the target network protocol. Layer 2 is protocol fields. Layer 3 is attributes belonging to all fields, and the last layer is attribute values.

A network protocol classification tree can be represented by a quintuple $PT=<P, F, A, V, R>$, of which:

P represents the target network protocol.

F represents the protocol fields of target network protocol. A protocol field contains a separate part of content of the protocol. Assuming that the protocol P has n fields, which is denoted by n-tuple $<f_1, f_2, ..., f_n>$, thus $F=\{f_i \mid i=1,2,...,n\}$ represents the set of protocol fields.

A represents disjoint attributes of protocol fields. Each protocol field contains several attributes. Assuming that protocol field $f_i(i=1,2,...,n)$ has m_i attributes, which is denoted by tuple $<a_{i1}, a_{i2}, ..., a_{ij}, ..., a_{imi}>$, thus $A_i=\{a_{ij} \mid j=1,2,...,m_i\}$ represents the set of attributes for protocol field fi. Furthermore, $A=A_1 \cup A_2 \cup ... \cup A_i \cup ... \cup A_n=\{a_{ij} \mid i=1,2,...n,\ j=1,2,...,m_i\}$ represents the set of attributes for protocol P.

V represents the value of attributes. Assuming that $V_{ij}=\{v_1, v_2,\ldots\}$ is the value set of attribute a_{ij} ($i=1,2,...n$, $j=1,2,...,m_i$), thus $V_i=V_{i1} \cup V_{i2} \cup ... \cup V_{imi}$ represents the value set of attributes for protocol field f_i. Furthermore, $V=V_1 \cup V_2 \cup ... \cup V_n=V_{11} \cup V_{12} \cup ... \cup V_{1m1} \cup V_{21} \cup V_{22} \cup ... \cup V_{2m2} ... \cup V_{n1} \cup V_{n2} \cup ... \cup V_{nmn}=\{v_{ij} \mid i=1,2,...n, j=1,2,...,m_i\}$ represents the value set of attributes for protocol P.

R represents the relation between the parent-nodes and the child-nodes in the tree, which is denoted by $R=\{relation_1, relation_2, relation_3\}$. The $relation_1$ refers to the relation between the target protocol P and protocol fields F. The $relation_2$ and $relation_3$ are similar to the $relation_1$, and represents the relation between protocol fields F and attributes A as well as between attributes A and attribute values V, respectively.

Figure 1 shows the schematic diagram of network protocol classification tree.

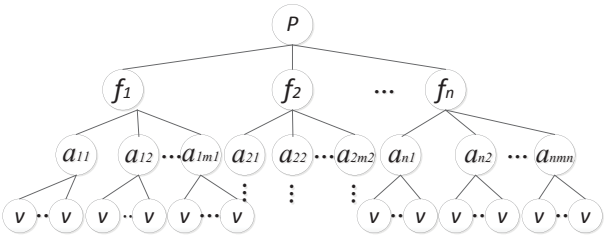

Figure 1. Example of protocol classification tree

2.2 Heuristic Operator

Without the heuristic operators, the value in attribute value set V_{ij} for protocol field f_i are usually generated by random method or experience. As a result, it needs enough values to get ideal coverage to meet test requirements. Therefore, in order to reduce the number of test data and further optimize the process of test data generation, it is necessary to take advantage of heuristic operators to obtain heuristic rules, which is used to remove useless items from the value sets of attributes.

Heuristic operator could be defined by a mapping function f_h: $V_{ij} \rightarrow V_{ij}^*$. Heuristic operator h is used to streamline the attribute value set V_{ij} and then obtain a new set V_{ij}^*. The number of elements for set V_{ij} and V_{ij}^* should satisfy the expression $|V_{ij}^*| < |V_{ij}|$. This means that the number of elements of set V_{ij} is smaller than that of set V_{ij}^*.

The heuristic operators could be extracted from the protocol specifications or by means of third-part tools.

The reducing of attribute values for protocol f_i by heuristic operators is shown as Figure 2.

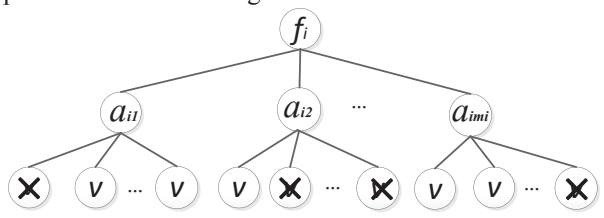

Figure 2. Example of reducing attribute values set

2.3 Generating Test Data

There are two steps to generate the fuzz testing data for target protocol.

Step 1: Generate the test data for each protocol field.

Mutual combination of attribute values could generate test data set for each protocol field. These values have been optimized by heuristic operators and belong to different attributes of each protocol field.

This combination means doing Cartesian product between value sets of attribute. For example, the test data set of protocol field f_i is denoted by $S_i=V_{i1}\times V_{i2}\times\ldots\times V_{imi}$ $(1\le i\le n)$. Elements in the set S_i is an ordered m_i-tuple $<mv_{i1}, mv_{i2}, \ldots, mv_{imi}>$ where mv_{ij} represents the value of attribute $a_{ij}(j=1,2,\ldots,m_i)$. Furthermore, all test data sets from different protocol fields can be referred as $S=\{S_1,S_2,S_3,\ldots,S_n\}$.

Step 2: Generate the fuzz testing data for network protocol.

Next, the fuzzing test data of network protocol could be obtained by mutating which is implemented inside each protocol field. To be specify, the mutation means selecting data from test data sets $S_1, S_2, S_3,\ldots, S_n$ for protocol field $f_1, f_2, f_3, \ldots, f_n$ respectively to replace the ordered n-tuple $<f_1, f_2, \ldots, f_n>$ in proper order.

The details of mutation are described in Figure 3.

$<f_1, f_2, f_3, \ldots, f_n>$
$\rightarrow<mv_{11}, f_2, f_3, \ldots, f_n>$ //replace f_1 with mv_{11}
$\rightarrow<mv_{12}, f_2, f_3, \ldots, f_n>$ //replace f_1 with mv_{12}
$\rightarrow<mv_{13}, f_2, f_3, \ldots, f_n>$ //replace f_1 with mv_{13}
$\rightarrow\ldots$
$\rightarrow<f_1, mv_{21}, f_3, \ldots, f_n>$ //replace f_2 with mv_{21}
$\rightarrow<f_1, mv_{22}, f_3, \ldots, f_n>$ //replace f_2 with mv_{22}
$\rightarrow<f_1, mv_{23}, f_3, \ldots, f_n>$ //replace f_2 with mv_{23}
$\rightarrow\ldots$
$\rightarrow<f_1, f_2, f_3, \ldots, mv_{n1}>$ //replace f_n with mv_{n1}
$\rightarrow<f_1, f_2, f_3, \ldots, mv_{n2}>$ //replace f_n with mv_{n2}
$\rightarrow<f_1, f_2, f_3, \ldots, mv_{n3}>$ //replace f_n with mv_{n3}
$\rightarrow\ldots$

Figure 3. Example of mutation

3 Experimental Evaluation

To evaluate the new method described in Section II, we selected FTP as target protocol and Peach as fuzzer. We modified Peach based on the new test data generation method. And then we applied Peach and modified Peach on test programs. In addition to the correctness of detecting vulnerabilities, we mainly evaluate the number of generated test data, the effectiveness of generated test data, and execution time of testing.

3.1 Experimental Environment

Firstly, we take Peach 2.3.8[11] as the fuzzer. Then in order to apply the new test data generation method, we also made some modifications on Peach, including adding a custom mutator and heuristic operator profiles. The implementation of mutated methods defined in custom mutator depends on heuristic operator profiles. And then Peach could use the custom mutator to do data mutation. In other words, the process of generating test data could be guided by the custom mutator with heuristic operator.

The target protocol focuses on FTP. And we selected Open-FTPD 1.2 and War FTP Daemon 1.82 RC11 server as the testing target, respectively.

3.2 Generating Test Data for FTP

1) Building Protocol Classification Tree

Each FTP[12] command starts with three or four uppercase NVT ASCII characters, and ends with CR (ASCII code for ENTER) or LF (ASCII code for newline). Its middle part is the optional parameters. According to the description of Section II, FTP command could be represented by a triple $<f_1, f_2, f_3>$, that is, the set of protocol fields is $F_{ftp}=\{f_1, f_2, f_3\}$.

To simplify the process, each protocol field could be described by two attributes: *content* and *length*. This means that protocol field f_1, f_2, and f_3 could be denoted by a tuple $<content, length>$, that is, $a_{11}=a_{21}=a_{31}=content$ and $a_{12}=a_{22}=a_{32}=length$. And thus the sets of attributes for three protocol fields are $A_1=A_2=A_3=\{content, length\}$. The range of value for attributes *content* and *length* is defined by the specification of protocol.

2) Defining Heuristic Operator with the help of IDA Pro

We use IDA pro[13] to extract heuristic operators.

The first step is to extract some critical instructions which could lead to vulnerability in the target program. This extraction could be achieved by analysing disassembly result that is obtained by IDA Pro. The critical instructions include CALL and CMP instructions[14]. The CALL instruction could reflect the usage of unsafe functions that is invoked by the target program, of which unsafe function refers to a function that might cause formatting vulnerability. Therefore, we could use it to extract the information about unsafe functions. While the CMP instruction could reflect decision branches existed in the code of the target program. And we also could use it to extract the information of comparison constants.

```
if heuristic_operator exists in funcList.conf
then construct token string with %s or %n
else break
```

Figure 4. Heuristic rule obtained from unsafe functions heuristic operator

```
if heuristic_operator .value == 0
then follow the special process
else if heuristic_operator .value∈[0, 127]
    then take heuristic_operator as ASCII
    else if heuristic_operator .value∈(127, 12700]
        then take heuristic_operator as length
        else take heuristic_operator as integer
```

Figure 5. Heuristic rule obtained from comparison constants heuristic operator

Next, we extract information of unsafe functions and comparison constants as heuristic operators when the fuzz testing data for FTP was generated. And then the information was written in the heuristic operator profiles *funcList.conf* and *constantList.conf*, respectively. Moreover, heuristic rules could be obtained on the basis of profiles. Figure 4 and Figure 5 show heuristic rules.

3) Generating Test Data for FTP

According to attributes *content* and *length*, also considering different fuzzing types, four kinds of data were choose as attribute values, such as formatting string, normal string, integer value, and different encoding string. As mentioned in Section II, then we could obtain test data for FTP.

3.3 Experimental Results

According to the vulnerability Information published by Venustech[15], Open-FTPD 1.2 might crash when USER or PASS command contains long strings. In this experiment, the USER command is to be tested. We compared the experimental results of the Peach and our method in Table 1.

Table 1. Results between Peach and our method based on Open-FTPD

	Vulnerabilities	Number of test data	Execution Time	Effectiveness	
				Data	Ratio
Peach	2	12966	10h16'28"	20	0.15%
the Method	2	2532	22'16"	34	1.34%

The second row of Table 1 means that all of results are obtained using Peach. And the third row means that all of results are obtained by our method which is implemented through doing some modifications on Peach.

According to the vulnerability information published by China National Vulnerability Database of Information Security[16], War FTP Daemon 1.82 RC11 may be appear denial of Service when CWD command contains long strings of which has "%s". In this experiment, we firstly use anonymous identity to login (username and password are all *anonymous*), and then test CWD command. We also compared the experimental results of the Peach and our method in Table 2. The meaning of each row is similar to Table 1.

Table 2. Results between Peach and our method based on War FTP Daemon

	Vulnerabilities	Number of test data	Execution Time	Effectiveness	
				Data	Ratio
Peach	0	12961	10h57'43"	0	0
the Method	1	8163	6h55'10"	1	0.01%

1) Correctness of Detection Vulnerabilities

The second column of the above tables means that the number of vulnerabilities which had been detected. Table 1 indicates that both Peach and our method could detect 2 vulnerabilities at the time of the OPEN-FTPD testing. These two vulnerabilities are "Tainted Data Controls Branch Selection" and "Tainted Data Passed to Function".

But Table 2 is different from Table 1. At the time of War FTP Daemon testing, Peach has not detected any vulnerability. However, our method detected the one vulnerability. This vulnerability results in the crash of War FTP Daemon program. This phenomenon is similar to a denial of service attack, which is consistent with the description in reference [16].

The results show that our method achieves a better ability to detect vulnerability.

2) Number of Test Data

The third column of the above tables means that the number of test data that are generated during testing. Table 1 and 2 indicate that our method generates test data which is 19.53% and 62.98% that of the Peach, respectively.

The results show that our method could effectively reduce the number of test data.

The reason is that Peach adopts the pre-generated approach for generating test data and could not generate specific test data for different testing targets. Meanwhile, there are a number of redundancy data. However, because the existence of heuristic operators, our method could generate different test data for different testing targets according to different heuristic rules. This makes our method more intelligent, and thus be able to reduce the number of redundant data.

3) Execution Time of Testing

The fourth column of the above tables means that the execution time of testing. Table 1 and 2 indicate that execution time of testing using our method is 3.61% and 63.12% that of the Peach, respectively.

The results show that our method could effectively reduce the execution time of testing. This further verifies that the quality of fuzz testing data directly affects the execution time of testing.

4) Effectiveness of Generated Test Data

The fifth column of the above tables means that the number of effective test data and the sixth column means that the ratio of effective test data and total test data. And the effective data means that it could trigger vulnerabilities.

For the OPEN-FTPD, if we use Peach to generate test data, there are 18 data triggering "Tainted Data Passed to Function" vulnerability and 2 data triggering "Tainted Data Controls Branch Selection" vulnerability. And the effec-

tive ratio is 0.15% approximately. While we use our method to generate test data, there are 22 data triggering "Tainted Data Passed to Function" vulnerability and 12 data triggering "Tainted Data Controls Branch Selection" vulnerability. And the effective ratio is 1.34% approximately. It proves that the decrease in total number of generated test data doesn't weaken its effectiveness.

For the War FTP Daemon, the Peach cannot catch any vulnerability after it has run out of all test data. So the effective test data is zero. However, the one vulnerability had been caught after using our method. And the effective test data is 1 as well as effective ratio is 0.01%.

The results highlight that our method could dramatically improve the effectiveness of generated test data.

In short, the experimental results indicate that our method could not only dramatically reduce quantity of generated test data, but also highly guarantee and improve the effectiveness of that. As a result, the execution time of testing is highly cut down. Especially it performs very well when it is used to fuzzing War FTP Daemon.

4 Conclusions

This paper proposes a new fuzz testing data generation method for network protocol based on classification tree and heuristic operator. The contribution has three aspects: (a) Build up a protocol classification tree. This provides the basic unit, such as attributes and protocol fields, for generating test data. (b) Introduce the heuristic operator. This removes useless items from the attribute value sets and further reduces the quantity of test data. (c) Obtain the test data for each field separately by doing Cartesian product between attribute value sets and further obtain the test data for protocol by mutating all protocol fields. Experimental results highlight that our method could not only dramatically reduce quantity of generated test data, but also highly guarantee and improve the effectiveness of that. Our findings could be applied in the network protocol fuzzing to detect vulnerability. Future work will be to optimize combination method instead of the simple Cartesian product to generate the test data for protocol fields, and preprocess the heuristic operators to remove redundancy values which might affect the generation of test data.

Acknowledgments

This paper is based upon works supported by the Key Project of National Defense Basic Research Program of China under Grant No. B1120132031.

We thank Qingxia Li, Hui Chang, Jingfeng Xue, Xiaolin Zhao, Yong Wang and Jingjing Hu for numerous discussions that helped us shape this paper. Also, thanks to the anonymous referees for providing useful comments and directing us to relevant references.

References

[1] Gorbunov S., and Rosenbloom A.: "Autofuzz: automated network protocol fuzzing framework", IJCSNS, 2010, 10, (8), pp. 239-245.

[2] Sutton M., Greene A., and Amini P.: "Fuzzing: brute force vulnerability discovery" (Pearson Education Press, 2007, 1st edn.) , pp. 21-36.

[3] Sui A. F., Tang W., Hu J. J., et al.: "An effective fuzz input generation method for protocol testing". Proc. Int. Conf. Communication Technology, Jinan, China, Sep. 2011, pp. 728-731.

[4] Hsu Y., Shu G., and Lee D.: "A model-based approach to security flaw detection of network protocol implementations". Proc. Int. Conf. Network Protocols, Florida, USA, Oct. 2008, pp. 114-123.

[5] Chen T. Y., Kuo F. C., Merkel R. G., et al.: "Adaptive random testing: the art of test case diversity". Journal of Systems and Software, 2010, 83, (1), pp. 60-66.

[6] Zhang Z., Wen Q. Y., and Tang W.: "An efficient mutation-based fuzz testing approach for detecting flaws of network protocol". Proc. Int. Conf. Computer Science and Service System, Nanjing, China, Jan. 2012, pp.814-817.

[7] Wang T., Wei T., Gu G., et al.: "TaintScope: a checksum-aware directed fuzzing tool for automatic software vulnerability detection". Proc. Symposium Security and Privacy, California, USA, May. 2010, pp. 497-512.

[8] Cain A., Chen T. Y., Grant D., et al.: "An automatic test data generation system based on the integrated classification-tree methodology", in "Software Engineering Research and Applications" (Springer Berlin Heidelberg Press, 2004, 1st edn.), pp. 225-238.

[9] Seo D., Lee H., and Nuwere E.: "SIPAD: SIP–VoIP anomaly detection using a stateful rule tree". Computer Communications, 2013, 36, (5), pp. 562-574.

[10] Malhotra R., and Khari M.: "Heuristic search–based approach for automated test data generation: a survey". International Journal of Bio-Inspired Computation, 2013, 5, (1), pp. 1-18.

[11] Peach. http://www.peachfuzzer.com,accessed March 2014.

[12] Stevens W. R.: "TCP/IP Illustrated Volume 1: the protocols" (China Machine Press, 2000, 1st edn.), pp. 316-331.

[13] The IDA Pro Disassembler and Debugger. http://www.hex-rays.com/idapro/, accessed March 2014.

[14] Seitz J.: "Gray Hat Python: Python Programming for Hackersand Reverse Engineers"(No StarchPress, 2009, 1st edn.), pp. 153-162.

[15] Venustech. Broadcast daily vulnerability. http://www.venustech.com.cn/NewsInfo/124/2313.Html, accessed March 2014.

[16] China National Vulnerability Database of Information Security. CNNVD-200611-119. http://www.cnnvd.org.cn/vulnerability/show/cv_id/2006110119, accessed March 2014.