Toward Fuzz Test Based on Protocol Reverse Engineering

Jun Cai(✉), Jian-Zhen Luo, Jianliang Ruan, and Yan Liu

School of Electronic and Information, Guangdong Polytechnic Normal University, Guangzhou 510665, China
{caijun,luojz}@mail.gpnu.edu.cn, 1891089@qq.com, liuyan_sysu@163.com

Abstract. Fuzz test is effective and efficient technique in discovering serious vulnerability in a network protocol by inserting unexpected data into the input message of the protocol and finding its bugs or errors. However, traditional fuzz test requires a large number of test cases to cover every test case, which is a time-consumed and inefficient process. In order to address this problem, we propose a novel method to reduce the number of test cases. The proposed method uses the technique of protocol reverse engineering to reconstruct the protocol's specification and create test cases by inserting fault fields into protocol input according to its format. The experimental results show that the proposed method can effectively identify the message fields of protocol and the total number of test cases is dramatically reduced.

Keywords: Vulnerability detection · Network security
Protocol reverse engineering

1 Introduction

Fuzz testing is a security test to discover the vulnerabilities of software systems by inserting random data or faults into the input of the software systems and detecting the software exceptions. Generally, there are two types of fuzz testing, i.e., Generation and data mutation [1]. The former type constructs test cases based on the complete specification of target protocol, while the latter type generates test cases by inserting faults into existing sample files.

The main problem of data mutation fuzz testing is that it needs too many fault-inserted files to cover all test cases, such as FileFuzz and SPIKEfile. The number of fault-inserted files come up to $2^{8\times\mathtt{FILESIZE}}$ if the size of sample file is FILESIZE. However, it is time consuming to handle so great sum of fault-inserted files when FILESIZE becomes large and many of which are not necessary for successful testing. Actually, a software system parses its input by considering the format of input and treats any file that do not follows the file format as invalid input. The system may throw an error and exit before it reaches the fault pieces of code.

Hence, it is a novel idea to use the generation type of fuzz testing and create test cases by taking the file format into account, such as PROTOS, a network protocol fuzzer. The advantage of generation type testing is that it decreases greatly the number of test cases but still maintains the maximum test cases coverage [2]. However, one has to completely analyze the target system and fully understand the protocol specification before he generates a series of effective but ad-hoc test cases. Since protocol reverse engineering [3] a the most promising technique to reconstruct the specification of private protocol, we apply the protocol reverse engineering technique to enhance the efficiency of network protocol fuzz test.

2 System Design

The protocol specification consists of both protocol message format and protocol state machine. The former is the protocol syntax rules which conduct the process of constructing different types of protocol messages, while the latter regulates the behaviors of protocol entities during the communication process, such as the order in which different types of messages should be sent or received.

In order to recover the protocol specification and perform a efficient fuzz test, we propose an architecture of protocol-reverse-engineering based fuzz test system. The core of the fuzz test is called QCD-PInfer, as shown in Fig. 1. The QCD-PInfer module includes the following four components: Data Pre-Processing, Multi-Change-Point Detection, Message Segmenting and Message Format Inference.

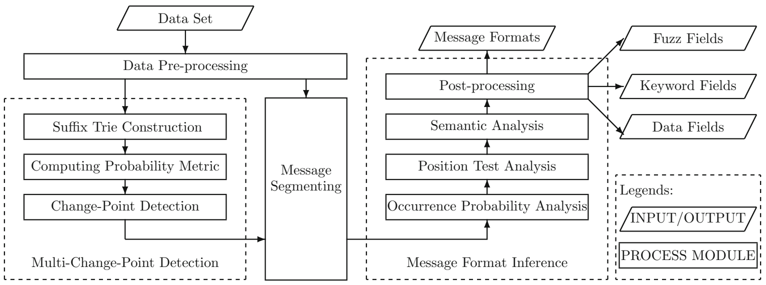

Fig. 1. The system architecture of QCD-PInfer.

We assume that a stochastic process is undergone by the observed messages and different type of fields have different statistical properties. So multiple change points would occur in the process and each of the change points indicates a change undergone by the statistical properties of the process. A change point means the ending of previous field and the beginning of a new field in a message. Under these assumption, our goal is transformed into the problem of multiple change-point detection and one can address this problem by using techniques of change-point detection [4,5].

Once the change-points are detected, message fields are identified. In order to determine the type of fields, a two-phase inference procedure, including occurrence probability analysis (OPA) and position test analysis (PTA), is present to classify the message fields into keyword fields, data fields and uncertain fields. In the OPA phase, fields with approximate zero-probability are marked as data fields. The other fields are passed to PTA to be further classified as keyword fields and uncertain fields. In the PTA phase, a benchmark position is selected for each field, and a binomial test is applied to test whether the field positions are equal to the benchmark position with probability 1 given a significance level α. The fields passed the statistic test are selected as keyword fields, while the rest fields are uncertain fields.

3 Results and Analysis

In this section, we perform experiments to evaluate the effectiveness of the proposed method. We also compare our results with those of Discoverer [6] and PI [7]. We implement the proposed approach on a system called QCD-PInfer in C/C++ and run all experiments on PCs with 2.93 GHz dual-core CPU, 4 GB RAM and operation system of Windows 7.

The recall and precision of inferred keyword fields are shown in Tables 1 and 2, respectively. It is important for us to note that, the true keywords are keywords occurred in the data set. Any keywords that do not appear in the data set will be omitted. We also note that the keyword quality of DNS and QQ would not be consider since the two protocols are pure binary protocol with no keyword defined in their protocol specifications.

As we seen, the recall rate of QCD-PInfer is higher than both Discoverer and PI. We also find that the recall rate of PI is too low: the recall rates of HTTP, FTP, SMTP and POP are less than 10%.

Table 1. The recall rate of protocol keyword.

System	HTTP	FTP	SMTP	POP	SSDP	BitTorrent
QCD-PInfer	87.0	92.9	85.7	84.0	74.1	100
Discoverer	78.3	60.7	64.3	40.0	33.3	100
PI	4.4	3.6	7.1	4.0	18.5	50.0

The precision of Discoverer is much lower than QCD-PInfer since Discoverer infers too many segments as keyword fields most of which are false positive. The precision of HTTP and FTP inferred by PI is 100%. However, recall rate of the two protocol by PI is 4.4% and 3.6%, respectively. The reason is that PI infers too few (less than 5) keyword fields, which leads to a low recall rate.

Table 2. The precision rate of protocol keyword.

System	HTTP	FTP	SMTP	POP	SSDP	BitTorrent
QCD-PInfer	66.7	97.0	35.0	95.8	66.0	66.7
Discoverer	7.2	23.3	19.2	22.8	33.9	5.3
PI	100	100	20.0	16.7	35.6	33.3

As shown in Fig. 2, the F-score of QCD-PInfer is higher than both Discoverer and PI for all the six protocols, which means that the quality of inferred keyword by QCD-PInfer is better than the other systems. Thus, QCD-PInfer outperforms Discoverer and PI in inferring keyword fields.

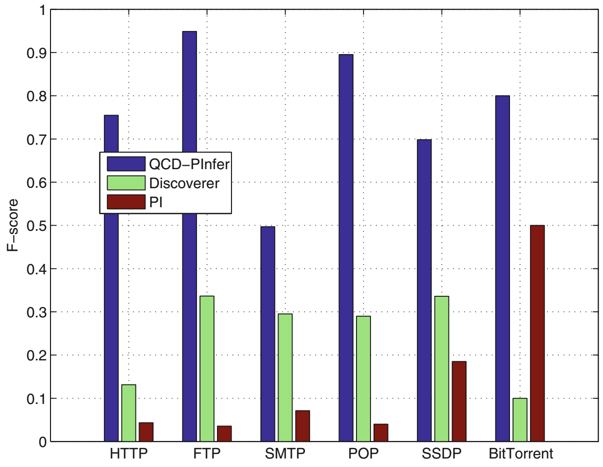

Fig. 2. The F-score value of keywords.

In this paper, we combine the proposed method with the idea of PROTOS to implement an automated fuzz testing tool (APREFuzz) for detecting the buffer overflow vulnerability of an information-centric network system in our test bed. The protocol used by the system could be considered as unknown protocol or private protocol since its protocol specification is not available to public. There are 5 message formats defined by the protocol specification, including “INTEREST” message, “DATA DISTRIBUTION” message, “DATA PUSHING” message, “RESPONSE WITH DATA” message and “RESPONSE WITHOUT DATA” message. APREFuzz is a proof-of-concept tool and only focuses on the buffer overflow vulnerability caused by the “DATA PUSHING” message.

Given a sample message, we firstly identify all protocol keyword fields and data fields in the message. Then generate test files by inserting fault data into these fields. In keyword fields, we insert fault data by replacing a particular keyword field with one of the inferred protocol keywords or one random string. In data fields, if the field contains only figures, we insert a boundary value of numbers into the field. Otherwise, we insert a random string into the field.

We compare our results with FileFuzz in Table 3. APREFuzz identifies 7 keyword fields and 7 data fields in the sample message. One of the data fields contains only figures. The number of inferred protocol keywords is 12. Thus, the total number of fault-inserted files is 248 ($= (12 + 11) \times 7 + 21 \times 1 + 11 \times 6$). However, FileFuzz generates $393,216$ ($= 1.5 \times 1024 \times 2^8$) fault-inserted files by replacing each byte with values from 0x00 to 0xFF. The results show that APREFuzz has detected one vulnerability while FileFuzz failed to detect the vulnerability.

Table 3. Fuzz testing to the information-centric network.

Fuzz system	Sample file	Fault-inserted files	Vulnerability
APREFuzz	1.5KB	248	1
FileFuzz	1.5KB	393,216	0

4 Conclusion

The key idea of the proposed method is to introduce the technique of protocol reverse engineering to enhance the performance of fuzz test. The proposed method considers the statistic properties of message fields and identifies the message fields by detecting the change point in these statistic properties and recover the message format by determining type and semantic of message fields. The technique of change point detection is an excellent solution to detect the change points. In order to deal with multiple change-points each of which corresponds to message field, a multi-change-point detection technique is proposed based on the traditional change point detection by restarting the detection procedure from an initial condition once a change point is detected. The message fields are further analyzed via occurrence probability test and position test, so as to identify the data fields, keyword fields and uncertain fields. The minimal description length criteria based position test analysis is proposed to identify those keyword fields which have multiple position in the message. The experiment results show that the protocol specification is useful for generating test cases for fuzz test.

Acknowledgment. This work was supported by the National Natural Science Foundation of China (61571141); Guangdong Natural Science Foundation (2014A030313637); The Excellent Young Teachers in Universities in Guangdong (YQ2015105); Guangdong Provincial Application-oriented Technical Research and Development Special fund project (2015B010131017); Science and Technology Program of Guangzhou (201604016108); Guangdong Future Network Engineering Technology Research Center (2016GCZX006); Special funds for public welfare research and capacity building in Guangdong Province (2014A010103032); Science and Technology Program of Guangdong (2016A010120010).

References

1. Munea, T.L., Lim, H., Shon, T.: Network protocol fuzz testing for information systems and applications: a survey and taxonomy. Multimed. Tools Appl. **75**(22), 14745–14757 (2016)
2. Kim, H.C., Choi, Y.H., Lee, D.H.: Efficient file fuzz testing using automated analysis of binary file format. J. Syst. Archit. **57**(3), 259–268 (2011). Special Issue on Security and Dependability Assurance of Software Architectures
3. Duchêne, J., Le Guernic, C., Alata, E., Nicomette, V., Kaâniche, M.: State of the art of network protocol reverse engineering tools. J. Comput. Virol. Hacking Tech., 1–16 (2017)
4. Zhao, Q., Ye, J.: Quickest detection in multiple on-off processes. IEEE Trans. Signal Process. **58**(12), 5994–6006 (2010)
5. Aminikhanghahi, S., Cook, D.J.: A survey of methods for time series change point detection. Knowl. Inf. Syst. **51**(2), 339–367 (2017)
6. Cui, W., Kannan, J., Wang, H.J.: Discoverer: automatic protocol reverse engineering from network traces. In: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, Berkeley, CA, USA, pp. 1–14. USENIX Association (2007)
7. Beddoe, M.A.: Network protocol analysis using bioinformatics algorithms (2004). http://www.baselineresearch.net/PI/