Tsinghua Science and Technology

Volume 21 | Issue 3 Article 11

2016

Test Data Generation for Stateful Network Protocol Fuzzing Using a Rule-Based State Machine

Rui Ma
the Beijing Key Laboratory of Software Security Engineering Technology, School of Software, Beijing Institute of Technology, Beijing 100081, China.

Daguang Wang
the Beijing Key Laboratory of Software Security Engineering Technology, School of Software, Beijing Institute of Technology, Beijing 100081, China.

Changzhen Hu
the Beijing Key Laboratory of Software Security Engineering Technology, School of Software, Beijing Institute of Technology, Beijing 100081, China.

Wendong Ji
the Beijing Key Laboratory of Software Security Engineering Technology, School of Software, Beijing Institute of Technology, Beijing 100081, China.

Jingfeng Xue
the Beijing Key Laboratory of Software Security Engineering Technology, School of Software, Beijing Institute of Technology, Beijing 100081, China.

Recommended Citation

Rui Ma, Daguang Wang, Changzhen Hu et al. Test Data Generation for Stateful Network Protocol Fuzzing Using a Rule-Based State Machine. Tsinghua Science and Technology 2016, 21(3): 352-360.

TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 11/11 pp352–360
Volume 21, Number 3, June 2016

Test Data Generation for Stateful Network Protocol Fuzzing Using a Rule-Based State Machine

Rui Ma*, Daguang Wang, Changzhen Hu, Wendong Ji, and Jingfeng Xue

Abstract: To improve the efficiency and coverage of stateful network protocol fuzzing, this paper proposes a new method, using a rule-based state machine and a stateful rule tree to guide the generation of fuzz testing data. The method first builds a rule-based state machine model as a formal description of the states of a network protocol. This removes safety paths, to cut down the scale of the state space. Then it uses a stateful rule tree to describe the relationship between states and messages, and then remove useless items from it. According to the message sequence obtained by the analysis of paths using the stateful rule tree and the protocol specification, an abstract data model of test case generation is defined. The fuzz testing data is produced by various generation algorithms through filling data in the fields of the data model. Using the rule-based state machine and the stateful rule tree, the quantity of test data can be reduced. Experimental results indicate that our method can discover the same vulnerabilities as traditional approaches, using less test data, while optimizing test data generation and improving test efficiency.

Key words: fuzzing; stateful network protocol; test data generation; rule-based state machine; stateful rule tree

1 Introduction

With the rapid development of network technology, vulnerability discovery in stateful network protocols has already become a research focus in the field of information security. At present, the main vulnerability discovery and analysis technology of stateful network protocols include manual testing, patch comparison, static analysis, dynamic analysis, fuzzing, etc.[1, 2]

Fuzzing is one of the most effective vulnerability discovery technologies. It is a method for discovering faults in software by providing unexpected input and monitoring exceptions[3]. Many of the vulnerabilities published by security organizations at home and abroad were discovered through fuzzing.

• Rui Ma, Daguang Wang, Changzhen Hu, Wendong Ji, and Jingfeng Xue are with the Beijing Key Laboratory of Software Security Engineering Technology, School of Software, Beijing Institute of Technology, Beijing 100081, China. E-mail: mary@bit.edu.cn.

∗ To whom correspondence should be addressed.

Manuscript received: 2016-01-22; accepted: 2016-03-24

Fuzzing plays an important role in the field of security testing of network protocols. But for stateful network protocols that have complex interactions and state transitions, test data generated by most general fuzzing methods do not achieve effective coverage of stateful trajectories. This seriously limits test efficiency and coverage.

For test data generation in stateful network protocol fuzzing, we propose a method based on a rule-based state machine and a stateful rule tree, so that the generated test data achieve high, targeted, and efficient coverage. Combining the ideas of traditional fuzzing methods with the characteristics of stateful network protocols, this method presents the concept of rule-based state machine and stateful rule tree to cut down the scale of state space. The rule-based state machine provides a formal description, and the stateful rule tree describes the relationship between states and messages for the stateful network protocol. According to a message sequence obtained by an analysis of paths, an abstract data model, which is a template for generating test data, is created. All instances of the abstract data

models constitute the final test set.

The remainder of this paper is organized as follows. Section 2 describes the characteristics and security testing of stateful network protocols, as well as the defects in current approaches. Section 3 introduces the details of test data generation using a rule-based state machine and a stateful rule tree, and Section 4 presents our experimental process and results in terms of a Session Initiation Protocol (SIP). We conclude the paper in Section 5.

2 Related Work

2.1 Characteristics of stateful network protocols

According to their context, network protocols can usually be divided into stateful and stateless. Most application-layer protocols are stateful.

In a stateful network protocol, multiple message interactions occur when a logical function needs to be performed, and while a context needs to be created according to its last state. Stateful network protocols therefore have certain characteristics:

(1) Complexity of communication. A complete interaction process usually includes a handshake, permissions validation, etc.

(2) Relevancy in context. In the process of message processing, not only the attributes of the current state but also the entire state trajectory before the current state are needed.

(3) Good transaction semantics. Each stage of complex requests in a stateful network protocol is divided into several sub-stages. The interactions of the sub-stages describe the process of a complete transaction.

(4) State transition. Depending on the message types being processed, the protocol can switch to different states while receiving or sending messages. State transition usually leads to a state space explosion during test data generation for stateful network protocol fuzzing.

2.2 Security testing of stateful network protocols

Research on security testing of stateful network protocol has achieved some results. Banks et al.[4] implemented a stateful network protocol fuzzer named *SNOOZE*, and described a method that can effectively identify vulnerabilities. This method allows the tester to describe a state operation of the protocol and the messages that need to be created in that state. Abdelnur et al.[5] proposed a fuzzer named *KIF* for stateful SIP, and discussed the usage of detecting vulnerabilities caused by software failures. Raniwala et al.[6] presented a testing method called *LRTP* for stateful transport protocols. Alrahem et al.[7] proposed *INTERSTATE*, which is a fuzzer for stateful SIP. Yu[8] used a protocol description language to describe the format of a network protocol so that the generated test data can be effective. Kitagawa et al.[9] presented a fuzzer called *AspFuzz* for application layer protocols. *AspFuzz* can automatically generate an attack message and then discover vulnerabilities aimed at some error messages that ignore protocol states and message sequences. Akbar and Faroop[10] proposed a security framework for Real-time Transport Protocol (RTP) fuzzing called *RTP*-miner. Gorbunov and Rosenbloom[11] designed a fuzzing framework based on an open-source framework, *AutoFuzz*. This framework constructs a finite-state machine for network protocols to extract protocol specifications, and guides test data generation. Li et al.[12] put forward a method that automatically identifies a variety of network protocols and generates a fuzzer for vulnerability discovery. Sui et al.[13] introduced a method that combines stochastic signal processing with regular expressions to guide test data generation, and then does fuzzing to test the protocol robustness. Tsankov et al.[14] implemented a lightweight fuzzer, *SECFUZZ*, which is designed for stateful encrypted network protocols and uses mutation rules to guide fuzz testing data generation. Seo et al.[15] presented a stateful rule-tree algorithm for stateful network protocol fuzzing, which generates test sequences by mapping each state and SIP grammar rules. Pan et al.[16] proposed a model-based testing method for network protocols. Using the protocol specification, the method builds up a formal model for input data, constructs the corresponding syntax tree for selected test nodes, and then uses these nodes to guide test data generation. Ma et al.[17] presented a fuzzing data generation method for network protocols using a classification tree and heuristic operators to reduce the quantity of test data.

2.3 Defects of stateful network protocol fuzzing

Although network protocol fuzzing has been widely used with different kinds of network protocols and related software products, and can effectively discover some vulnerabilities, it has some deficiencies.

First, it lacks support for state information. Owing to the characteristics of stateful network protocols, not

only each state, but also the entire state trajectory, should be considered in test data generation. Because the context information and all states of a message sequence are not included in traditional fuzzing, generated test data for each state are discrete, and cannot cover the full state trajectory. Thus, vulnerabilities in state transitions may not be detected.

Second, there are a lot of redundant test data. Test data are generated randomly without rules; as a result, most test data are neglected by the target protocol.

Finally, most methods have a lack of specificity and over-reliance on manual operations. In order to reduce the redundant test data, specifications defined by testers are required in some fuzzers. Although this method might decrease the blindness of test data generation, it is too dependent on testers, and may lead to low coverage and incomplete testing.

3 Test Data Generation on the Basis of Rule-Based State Machines

3.1 Overview

Focusing on the characteristics of stateful network protocols, as well as on defects in test data generation for traditional stateful network protocol fuzzing, this paper presents a new method for test data generation based on a rule-based state machine and a stateful rule tree. The main steps of the method are as follows[18]:

Step 1: Obtain the specification of a target network protocol, and then get protocol rules and a preliminary rule-based state machine through analyzing the protocol format.

Step 2: To generate a simplified state machine, remove the safety paths by using the preliminary rule-based state machine model and protocol rules.

Step 3: Generate the stateful rule tree, which combines state and rule information, according to the protocol rules and the rule-based state machine model. Then simplify the relationship between states and messages by using the stateful rule tree so as to remove the meaningless state–message combinations.

Step 4: Analyze all state–message paths according to the stateful rule tree, and obtain an abstract data model of test cases. Then generate initial test cases, including the state trajectory, messages for each state, and a series of state trajectories.

Step 5: Mutate initial test cases regularly with a data generation algorithm to generate a large number of test cases, which have the same state–message path but different values from the initial ones; then form the final set of test data.

The rule-based state machine model and the stateful rule tree are key points of the proposed method. The rule-based state machine model is for the storage of state information and the stateful rule tree is for the storage of message paths.

3.2 Rule-based state machine

The finite state machine is an appropriate formal description method for describing the state transitions of network protocols. Focusing on the characteristics of stateful network protocols, and combining them with requirements of fuzz testing data generation, this paper designs a rule-based state machine model for storing information of each state.

A rule-based state machine model can be represented by a 6-tuple $\mathrm{RM} = < S_0, S, I, O, F, V >$, where:

S_0 represents the initial state, which is the start of the entire state space.

S represents all states of the entire state space.

I represents the set of formats of input data. Each I_k of the set represents the format of the k-th input data unit, where k is greater than or equal to 0, but less than the total quantity of tuples.

O represents the set of formats of output data unit. Each O_k of the set represents the format of the k-th output data unit, where k is greater than or equal to 0, but less than the total quantity of tuples.

F is the state transition function that indicates the migration relations of states.

V represents the protocol state rules. It contains all attributes that indicate the characteristics of states, such as authentication ID, cookies, and others. A detailed description of V depends on the protocol specification.

The rule-based state machine is an effective theoretical formal description model for describing the stateful network protocol so as to store state and trajectory information of the target protocol. It not only describes the characteristics of states, but also can construct the state space using the rules, and optimize the state trajectory.

As shown in Fig. 1, there are two state trajectories in the state space: $S_0 \rightarrow S_1 \rightarrow S_3$ and $S_0 \rightarrow S_2 \rightarrow S_3$. Assume that there is a generation rule: if no exception occurs in the transition from one state to another, then this state trajectory is safe. The safety trajectory is meaningless for test data generation, so it can be ignored, to reduce the scale of the state space. Suppose

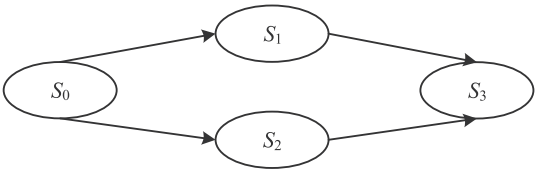

Fig. 1 Original state space.

that the path $S_0 \rightarrow S_2 \rightarrow S_3$ is a safety trajectory, the reduced state space is shown in Fig. 2.

3.3 Stateful rule tree

The stateful rule tree is a hierarchical tree based on the rule-based state machine and the protocol specification. It is represented by a 7-tuple $T = < S, M, \text{Sub}, H, F_1, F_2, F_3 >$ as the formal description, where:

S represents the set of states that indicates the state space of the stateful network protocol.

M represents the set of messages that contains all protocol messages specified in the stateful network protocol.

Sub represents the set of sub-messages that contains all protocol sub-messages specified in the stateful network protocol. In the stateful protocol specification, a message usually consists of several sub-messages.

H represents the set of message headers that includes all the message headers in the stateful network protocol.

$F_1 = S \times M$, which represents the mapping relationship between states and messages.

$F_2 = M \times \text{Sub}$, which represents the mapping relationship between messages and sub-messages.

$F_3 = \text{Sub} \times H$, which represents the mapping relationship between sub-messages and message headers.

The structure of the stateful rule tree is shown in Fig. 3. The first layer is the rule-based state machine of the protocol. The second layer is the message communication for each specific state, and indicates the received and sent message types under that state. The third layer is the specific messages that contain requests, confirmations, and so on. The fourth layer is message headers that contain some attribute fields. If necessary, you can also define sub-fields for the attribute fields in the fifth layer.

The stateful rule tree represents the correlation between states and messages. There is a path between S and M if and only if receiving or sending message

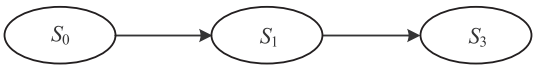

Fig. 2 State space after removing safety paths.

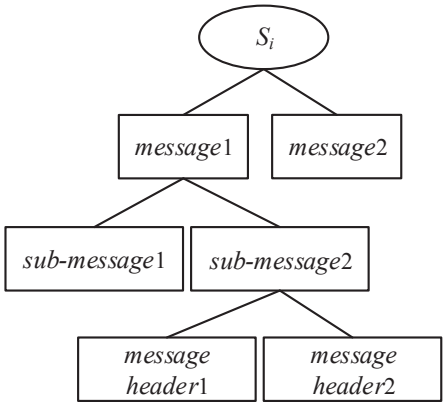

Fig. 3 Structure of stateful rule tree.

M is possible under state S, which means there is correlation between S and M. No path exists between S and M if and only if receiving or sending message M is impossible under state S, which means there is no correlation between S and M. For example, we could build a stateful rule tree as shown in Fig. 4 if sending *message*2 is impossible but receiving *message*1 is possible under the *state*.

The stateful rule tree shows the relationship between state combinations and messages. With this relationship description, we can remove meaningless test data quickly so as to generate more effective test data and furthermore prevent a state space explosion. Moreover, with an appropriate heuristic searching algorithm, we can also remove meaningless test data under a specific state to make the fuzzing more effective.

3.4 Generating abstract data models for test cases

The abstract data model for test case is the template of test data generation. It specifies the data format of test cases, as well as message sequence of testing.

The generation process for the abstract data model: Assume that there is a simplified rule-based state machine of a stateful network protocol (shown in Fig. 5). Three states of this state machine are *state*0, *state*1, and *state*2. The transition between states depends on received messages. Table 1 shows the state–message paths.

Table 1 shows three state–message paths from *state*0 to *state*2. The three paths correspond to three

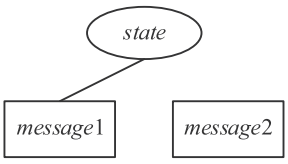

Fig. 4 Example of state rule tree.

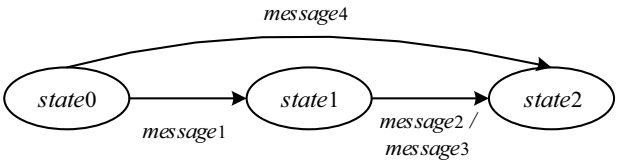

Fig. 5 Simplified state space.

Table 1 State-message paths.

No.	Path
1	*state*0→ *message*1→ *state*1→ *message*2→ *state*2
2	*state*0→ *message*1→ *state*1→ *message*3→ *state*2
3	*state*0→ *message*4 → *state*2

abstract data models of test cases, namely *message*1→ *message*2, *message*1→ *message*3, and *message*4.

According to the network protocol specification, we can determine the message format to obtain the test data model (Fig. 6) after the message sequence is determined. Figure 6 indicates that the message sequence consists of *message*1 and *message*2. Each message is made up of several fields, and each field represents a protocol attribute constrained with rules like data type, length, and so forth.

3.5 Generating test data

A test dataset is an instance of an abstract data model. An abstract data model can generate several test datasets. We can use different kinds of data-generation algorithms to generate the instance.

The collection of test data generated by all abstract data models is the final test dataset.

4 Experiment and Evaluation

In order to evaluate the method we proposed in Section 3, we selected the SIP as a target protocol and the Sulley[19] as a fuzzer, and we implemented the fuzz testing data generation method using the rule-based state machine and stateful rule tree for the stateful network protocol on the Sulley. Test data was generated by the method provided in the Sulley and our method, respectively. In addition to the correctness of discovering vulnerability, we evaluated the amount of generated test data and testing execution time.

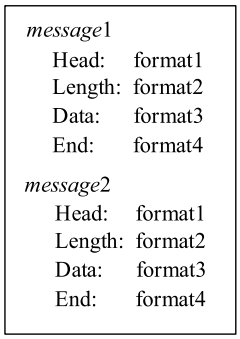

Fig. 6 Example of abstract data model for test case.

4.1 Experimental environment

Experiments were done under Windows 7. First, we take the Sulley as the fuzzer. Then we implemented the new test data generation method with rule-based state machines based on the Sulley. The target protocol focuses on the SIP, and we selected officeSIP server 3.1 as our testing target.

4.2 Generating test data for the SIP

The four main transaction states in the SIP are the INVITE transaction state of a client, non-INVITE transaction state of a client, INVITE transaction state of the server, and non-INVITE transaction state of the server. In this paper, our focus is mainly on analysis of the INVITE transaction state of the server.

4.2.1 Constructing a rule-based state machine for the SIP

According to RFC3261[20], the four states in the INVITE server of SIP are *Running*, *Completion*, *Confirmation*, and *Termination*. As shown in Fig. 7, there are some state transitions while sending and receiving messages.

For the states and characteristics of the SIP, we define the rule-based state machine model as RM $= < S_0, S, I, O, F, V >$, where:

$S_0 = \{Running\}$;

$S = \{Running, Completion, Confirmation, Termination\}$;

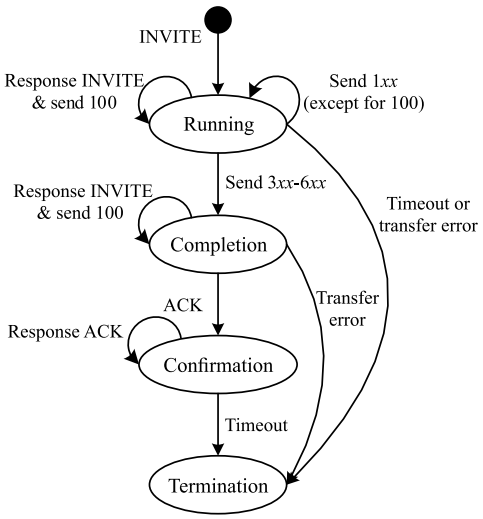

Fig. 7 Transaction state machine of INVITE server.

I ={INVITE, 1*xx*-6*xx*, ACK};
O ={ACK, *timeout*, *transfer error*};
F ={*Running*→*Running*,*Completion*→*Completion*,
Running→*Completion*,*Confirmation*→*Confirmation*,
Completion→*Confirmation*,*Confirmation*→*Termination*,
Running→*Termination*,*Completion*→*Termination*};
V represents the protocol state rules.

The rule-based state machine of the SIP is constructed as follows: the server enters the *Running* state when it receives an INVITE request, and transfers to the *Completion* state when it receives a 300–699 response. If the connection is established successfully during the *Completion* state, the server will receive an ACK message, and then move into the *Confirmation* state. Otherwise, the server will directly move into the *Termination* state. If the server receives a *timeout* during the *Confirmation* state or *Running* state, then it directly moves into the *Termination* state.

If there is no exception caused by unsafe data from the *Completion* state to the *Termination* state , or from the *Confirmation* state to the *Termination* state, we mark these two paths as safety paths that should be removed. The simplified rule-based state machine of SIP is shown in Fig. 8.

4.2.2 Building a stateful rule tree for SIP

Based on the formal definition of a stateful rule tree, we build up the stateful rule tree for SIP as shown in Fig. 9.

The stateful rule tree of SIP is described as a 7-tuple $T = < S, M, \text{Sub}, H, F_1, F_2, F_3 >$, where:

$S = \{Running, Completion, Confirmation, Termination\}$;
M = {INVITE, ACK, 1*xx*, …, 6*xx*, …};
Sub ={Call-ID, From, Accept, …};
H = {*callid*, *HCOLON*, *from-spec*,
accept-range, …};
F_1 = {*Running* → INVITE, *Running* → ACK,
Running → 1*xx*, …, *Running* → 6*xx*,
Completion→INVITE, *Completion*→ACK,
Completion → 1*xx*, …, *Completion* → 6*xx*,
Confirmation → INVITE,
Confirmation → ACK, …};
F_2 = {INVITE → Call-ID, INVITE → From,
ACK→ Accept, ACK → Call-ID,
ACK → From, …};
F_3 = {Call-ID→*callid*, Call-ID→HCOLON,
From→ *from-spec*, From → HCOLON,
Accept→ HCOLON,
Accept→*accept-range*, …}.

As shown in Fig. 9, since sending a response message like 1*xx*-6*xx* is impossible in the *Confirmation* state, the sub-tree on the right of *Confirmation* is meaningless, and should be removed. Because no messages will be received or sent in the *Termination* state, the tree has only the *Termination* state as its root node. According to the above rules, Fig. 10 shows the new pruned state rule tree, based on the one shown in Fig. 9.

The stateful rule tree combines protocol states and protocol rules to make pruning possible. After pruning the stateful rule tree, large numbers of meaningless combinations are removed, and thus prevented from generating too much invalid test data. Therefore, the stateful rule tree acts as an effective means for test data generation.

4.2.3 Generating an abstract data model for test case

With the stateful rule tree shown in Fig. 10, we obtain ten state–message paths, some of which are listed in Table 2. For example, *Path*1 represents the situation

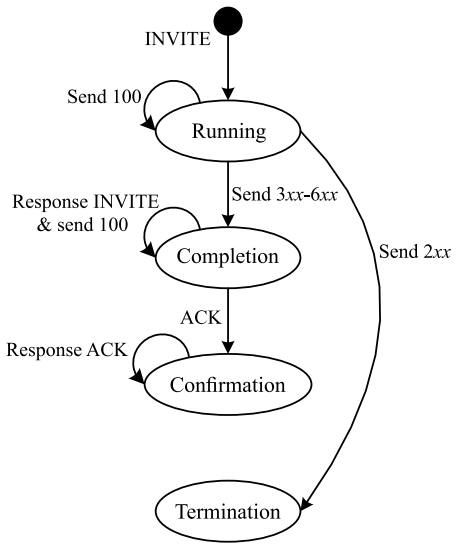

Fig. 8 Simplified rule-based state machine of SIP.

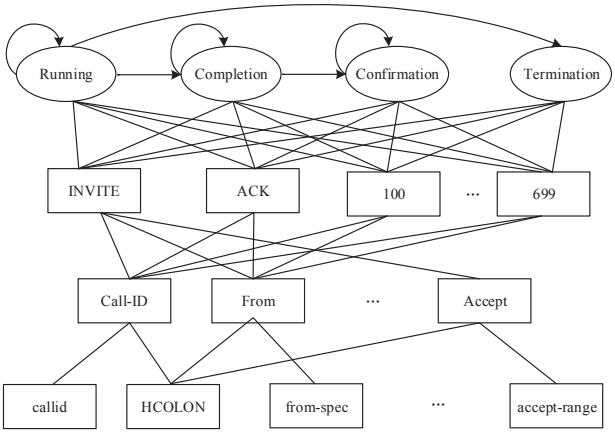

Fig. 9 Stateful rule tree of SIP before pruning.

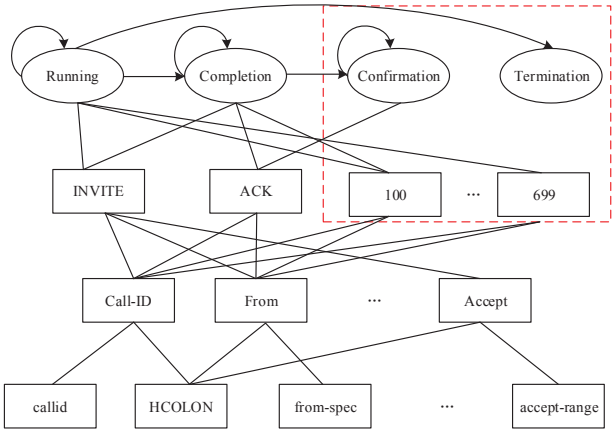

Fig. 10 Stateful rule tree of SIP after pruning.

Table 2 Some state–message paths of SIP.

Name	Path
*Path*1	INVITE → *Running* → 300–699 → *Completion* → ACK → *Confirmation*
*Path*2	INVITE → *Running* → 200–299 → *Termination*
*Path*3	INVITE → *Running* → 100–199 → *Running* → 200–299 → *Termination*
*Path*4	INVITE → *Running* → 100–199 → *Running* → 300–699 → *Completion* → ACK → *Confirmation*

in which the server enters the *Running* state when it receives an INVITE request. The server then enters the *Completion* state when it receives an ACK message. At this point the path ends. The state–message paths determine the sequence of messages. For instance, the message sequence INVITE → 300-699→ ACK is obtained according to *Path*1.

After determining the sequence of messages, and according to the message rules of SIP given in Ref. [15], we define the data format of each message. For example, the rule of the *callid* field in the INVITE message is "`callid: (# ASCII # {1,50} (|@(|w|*) {1,32}).?)`", where "`# ASCII # {1,50}`" indicates that the field uses ASCII encoding, as well as that its length is greater than 1 byte and less than 50 bytes.

The data format of messages determines the abstract data model for test case. Moreover, an instance of the abstract data model is a test data, and the collection of test data generated by all abstract data models is the final test dataset.

4.3 Experimental evaluation

In order to verify the test data generation of the stateful network protocol fuzzing using a rule-based state machine and a stateful rule tree, we compared the experimental results of the Sulley method and our method in terms of vulnerability discovery, the quantity of generated test data, and testing execution time.

4.3.1 Vulnerability discovery

The capability of discovering vulnerability is one of the important factors for evaluating the effectiveness of fuzz testing data generation approaches. Table 3 shows a comparison of vulnerability information before and after the improvement. The third column means that all results were obtained using the Sulley method, and the fourth column means that all results are obtained using our generation method, implemented through modification of the Sulley approach.

Table 3 indicates that both the Sulley and our method detected one vulnerability at the time of the officeSIP server 3.1 testing. This vulnerability was published by the China National Vulnerability Database of Information Security, as well as in the Common Vulnerabilities & Exposures. It is a kind of remote denial-of-service vulnerability, and is due to an error when handling "To" headers in SIP requests. With a specially crafted SIP INVITE request containing an incomplete recipient address in the "To" header, a remote attacker can cause the server to crash, resulting in a loss of availability[21].

The results show that the same number of vulnerabilities was detected before and after the improvement, and our method achieves the same capability of discovering vulnerability as the Sully.

4.3.2 Fuzzing efficiency

The efficiency of fuzzing focuses on the quantity of test data and the testing execution time. Table 4 shows a comparison before and after our improvement.

(1) Test data quantity

The quantity of generated test data illustrates the

Table 3 Effectiveness of test data.

Vulnerability name	Vulnerability No.	Discovering vulnerability	
		Sulley	Our method
OfficeSIP Server Input Validation Vulnerability	CNNVD-201202-145/ CVE-2012-1008	Yes	Yes

Table 4 Fuzzing efficiency.

	Test data quantity	Testing execution time
Sulley	14 247	15 h 31 min
Our method	11 641	12 h 52 min

effect of state-space pruning. The data indicates that our method generates test data that is 81.71% the quantity of the Sulley, this also demonstrates that our method can effectively reduce the state space. The reason is that the Sulley adopts a pre-generated approach for generating test data, and cannot generate specific test data for different testing targets. This results in redundant test data. Our method uses a rule-based state machine model and a stateful rule tree to remove safety paths to reduce the scale of the state space, as well as removing useless states and messages. This makes our method more intelligent, and thus able to reduce the quantity of redundant data.

(2) Testing execution time

The test efficiency is represented as the number of vulnerabilities divided by the testing execution time. Thus for a given number of vulnerabilities, if the testing execution time is shorter, the test efficiency is higher.

The data in Table 4 indicates that execution time was reduced 2 hours and 39 minutes using our method, when the Sulley and our method have the same vulnerability discovery capability.

In short, the experimental results highlight that our method could not only guarantee the capability of discovering vulnerabilities, but also reduce the quantity of generated test data and the testing execution time. This proves that our method can scale down the state space of a stateful network protocol as well as optimize the combinations of states and messages. With the optimization, the test dataset of stateful network protocol fuzzing can achieve more efficiency.

5 Conclusion

Network protocol fuzzing is one of the important focal points in the field of information security in recent years. Because of the characteristics of state transitions in stateful network protocols, they cannot be tested effectively by traditional fuzzing. Considering the characteristics of protocol states, this paper proposes a test data generation method based on a rule-based state machine and a stateful rule tree for stateful network fuzzing. We use the rule-based state machine to describe a stateful network protocol formally, and use rules to screen the generation process so as to obtain more effective and simplified test data. We also use the stateful rule tree to store message paths so as to generate test data to cover both each state and any state transition trajectory. Experiments have been done to verify the correctness of vulnerability discovery, the quantity of test data, and the testing execution time. The experimental results reveal that our method could not only guarantee vulnerability discovery, but also reduce the quantity of test data and decrease testing execution time.

Our research work still exhibits some defects. For example, the verification is not adequate. We need to select more protocols, especially privacy protocols whose specifications could not be made available. Future work will also be on how to extract rules from protocols.

Acknowledgment

This work was supported by the Key Project of National Defense Basic Research Program of China (No. B1120132031) and also supported by the Cultivation and Development Program for Technology Innovation Base of Beijing Municipal Science and Technology Commission (No. Z151100001615034).

References

[1] Z. Pan, C. Liu, S. Liu, and S. Guo, Vulnerability discovery technology and its applications, *Journal of Software*, vol. 8, no. 8, pp. 2000–2007, 2013.

[2] J. Chen, H. Wang, D. Towey, C. Mao, R. Huang, and Y. Zhan, Worst-input mutation approach to Web services vulnerability testing based on SOAP messages, *Tsinghua Science & Technology*, vol. 19, no. 5, pp. 429–441, 2014.

[3] M. Stutton, A. Greene, and P. Amini, *Fuzzing: Brute Force Vulnerability Discovery*. London, UK: Pearson Education, 2007.

[4] G. Banks, M. Cova, V. Felmetsger, K. Almeroth, R. Kemmerer, and G. Vigna, SNOOZE: Toward a stateful network protocol fuzzer, *Lecture Notes in Computer Science*, vol. 4176, pp. 343–358, 2006.

[5] H. J. Abdelnur, R. State, and O. Festor, KIF: A stateful SIP fuzzer, in *Proc. 1st Int. Principles, Systems & Applications of IP Telecommunications Conf.*, New York, NY, USA, 2007, pp. 47–56.

[6] A. Raniwala, S. Sharma, P. De, R. Krishnan, and T. C. Chiueh, Evaluation of a stateful tansport protocol for multi-channel wireless mesh networks, in *Proc. 15th IEEE Int. Quality of Service Workshop*, Evanston, IL, USA, 2007, pp. 74–82.

[7] T. Alrahem, A. Chen, N. DiGiussepe, J. Gee, S. Hsiao, and S. Mattox, INTERSTATE: A stateful protocol fuzzer for SIP, *Defcon*, no. 15, pp. 1–5, 2007.

[8] H. H. Yu, Research on vulnerbility discovering for network protocol based on fuzz testing, (in Chinese), MS thesis, Dept. Sci. & Tech., Huazhong University, Wuhan, China, 2008.

[9] T. Kitagawa, M. Hanaoka, and K. Kono, AspFuzz: A state aware protocol fuzzer based on application-layer protocols,

in *Proc. IEEE Computers & Communications Symposium*, Riccione, Italy, 2010, pp. 202–208.
[10] M. A. Akbar and M. Faroop, RTP-miner: A real-time security framework for RTP fuzzing attacks, in *Proc. 20th Int. Network & Operating Systems Support for Digital Audio & Video Workshop*, Amsterdam, Netherlands, 2010, pp. 87–92.
[11] S. Gorbunov and A. Rosenbloom, Autofuzz: Automated network protocol fuzzing framework, *International Journal of Computer Science & Network Security*, vol. 10, no. 8, pp. 239–245, 2010.
[12] M. W. Li, A. F. Zhang, J. C. Liu, and Z. T. Li, An automatic network protocol fuzz testing and vulnerability discovering method, (in Chinese), *Chinese Journal of Computer*, vol. 34, no. 2, pp. 242–255, 2011.
[13] A. F. Sui, W. Tang, J. J. Hu, and M. Z. Li, An effective fuzz input generation method for protocol testing, in *Proc. 13th IEEE Int. Communication Technology Conf.*, Jinan, China, 2011, pp. 728–731.
[14] P. Tsankov, M. T. Dashti, and D. Basin, SECFUZZ: Fuzztesting security protocols, in *Proc. 7th Int. Automation of Software Test Workshop*, Zurich, Switzerland, 2012, pp. 1–7.
[15] D. Seo, H. Lee, and E. Nuwere, SIPAD: SIP-VoIP anomaly detection using a stateful rule tree, *Computer Communications*, vol. 36, no. 5, pp. 562–574, 2013.
[16] F. Pan, Y. Hou, Z. Hong, L.Wu, and H. Lai, Efficient model based fuzz testing using higher-order attribute grammars, *Journal of Software*, vol. 8, no. 3, pp. 645–651, 2013.
[17] R. Ma, W. D. Ji, C. Z. Hu, C. Shan, and W. Peng, Fuzz testing data generation for network protocol using classification tree, in *Proc. Communication Security Conf.*, Beijing, China, 2014, pp. 97–101.
[18] C. Z. Hu, R. Ma, X. Han, C. Shan, and Y. Wang, A rule-based method of designing model for stateful network protocol, (in Chinese), China Patent CN201410333944.0, Nov. 12, 2014.
[19] Sulley, https://github.com/OpenRCE/sulley, 2015.
[20] RFC3261, http://www.ietf.org/rfc/rfc3261.txt, 2015.
[21] CVE, http://cve.scap.org.cn/CVE-2012-1008.html, 2015.

Rui Ma received the PhD degree from Beijing Institute of Technology. She is currently an associate professor with the School of Software, Beijing Institute of Technology. Her current research interests include software security and Internet of Things.

Daguang Wang is a master student in the School of Software at the Beijing Institute of Technology. She received the BEng degree from Inner Mongolia University in 2013. Her current research interests include software testing and network security.

Changzhen Hu received the PhD degree from Beijing Institute of Technology. He is currently a professor with the School of Software, Beijing Institute of Technology. His current research interest is information security.

Wendong Ji is a master student in the School of Software at the Beijing Institute of Technology. He received the BEng degree from Beijing Institute of Technology in 2013. His current research interest is software security.

Jingfeng Xue received the PhD degree from Beijing Institute of Technology in 2003. He is currently a professor with the School of Software, Beijing Institute of Technology. His current research interest is software security.