TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 02/10 pp458-468
Volume 22, Number 5, October 2017

Semi-valid Fuzz Testing Case Generation for Stateful Network Protocol

Rui Ma*, Shuaimin Ren, Ke Ma, Changzhen Hu, and Jingfeng Xue

Abstract: Network protocols are divided into stateless and stateful. Stateful network protocols have complex communication interactions and state transitions. However, the existing network protocol fuzzing does not support state transitions very well. This paper focuses on this issue and proposes the Semi-valid Fuzzing for the Stateful Network Protocol (SFSNP). The SFSNP analyzes protocol interactions and builds an extended finite state machine with a path marker for the network protocol; then it obtains test sequences of the extended finite state machine, and further performs the mutation operation using the semi-valid algorithm for each state transition in the test sequences; finally, it obtains fuzzing sequences. Moreover, because different test sequences may have the same state transitions, the SFSNP uses the state transition marking algorithm to reduce redundant test cases. By using the stateful rule tree of the protocol, the SFSNP extracts the constraints in the protocol specifications to construct semi-valid fuzz testing cases within the sub-protocol domain, and finally forms fuzzing sequences. Experimental results indicate that the SFSNP is reasonably effective at reducing the quantity of generated test cases and improving the quality of fuzz testing cases. The SFSNP can reduce redundancy and shorten testing time.

Key words: network protocol fuzzing; extended finite state machine; test sequence; state transition marking algorithm; semi-valid algorithm

1 Introduction

To ensure the quality and security of software, information security now has higher requirements for software security testing. Fuzzing is a very effective and widely used method. It works "by providing the unexpected inputs and monitoring malformed results to find software failures"[1]. Network protocol fuzzing has a high probability of discovering high-risk vulnerabilities. Therefore, it attracts increasingly more security researchers. Furthermore, since the network protocol is widely used in communication on the Internet, once a vulnerability is discovered, the effect will be very serious. Thus, network protocol fuzzing is quite significant.

Network protocols are divided into stateless and stateful. Stateful network protocols have complex communication interactions and state transitions. However, the existing network protocol fuzzing cannot support state transitions very well. For these issues, according to the characteristics of the stateful network protocol, this paper proposes the semi-valid stateful network protocol fuzzing based on the extended finite state machine, namely the Semi-valid Fuzzing for the Stateful Network Protocol (SFSNP).

First, the SFSNP builds an extended finite state machine with a path marker for the network protocol

• Rui Ma, Shuaimin Ren, Changzhen Hu, 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.
• Ke Ma is with the Internet Center, Institute of Communication Standard Research, China Academy of Information and Communication Technology, Beijing 100191, China.
∗ To whom correspondence should be addressed.
Manuscript received: 2016-09-30; accepted: 2016-10-21

by analyzing the protocol interaction. Then, it gets test sequences of the extended finite state machine. It further performs mutation operations by using the semi-valid algorithm for each state transition in the test sequence. Finally, it obtains fuzzing sequences. The same state transitions may belong to different test sequences. Thus, the SFSNP uses a state transition marking algorithm to reduce the redundant test cases caused by this problem. Using the protocol stateful rule tree and extracting constraints from the protocol specifications, the SFSNP constructs semi-valid fuzz testing cases within the sub-protocol domain to form final fuzzing sequences.

The remainder of this paper is organized as follows. Section 2 introduces the analysis of some issues in stateful network protocol fuzzing. Section 3 proposes the details of SFSNP. Section 4 presents our experimental processes and results. Finally, Section 5 concludes.

2 Related Work

The stateful network protocol fuzzing concerns on two aspects, namely formal description and test case generation.

Currently, the most common representation of the protocol is the network protocol specification consisting of natural language. Because natural language is pure text, it is not conducive to perform fuzzing. Meanwhile, selecting the type of formal description is especially important because of the characteristics of the stateful network protocol.

Test case generation for the protocol is the key step of fuzzing. Therefore, for stateful network protocol, we need to improve the efficiency of the fuzz testing case generation and guarantee vulnerability discovering ability.

2.1 Characteristics of stateful network protocols

If a network protocol needs multiple message interactions when performing logical functions and considers the preceding state when data was transmitted, the network protocol can be called a stateful network protocol.

The stateful network protocol has four characteristics:

(1) Complexity of communication: Complete interaction processes generally include hand shaking, permissions validation, and so on.

(2) State transition: The stateful network protocol can switch to different states according to the type of messages currently being processed. State transitions can lead to state space explosion in the process of fuzz testing case generation for the stateful network protocol.

(3) Relevancy of contextual information: During message processing, we need not only the attributes of the current state, but also the attributes of the previous state in the entire state trajectory until now.

(4) Good transaction semantics: The stateful network protocol divides complex function requests into several sub-stages. The interaction of multiple sub-stages describes the process of a complete transaction.

2.2 Formal description of stateful network protocols

A formal description of the network protocol is the premise of fuzzing[2]. An appropriate protocol formal description model can not only accurately describe the details of the protocol, but also provide credible foundation for fuzzing.

There are several common technologies currently used: Petri net, temporal logic, Finite State Machine (FSM), CCS, and Z Representation[3]. Nevertheless, in the aspect of protocol interaction, especially for the stateful protocol based on state transition and message-driven, the FSM is the optimal formal description model. The acquisition of test sequences based on the FSM has many well-developed theories. For example, focusing on the vulnerability discovering for the protocol, Shu et al.[4] proposed a model-based method of monitoring security vulnerability. This method uses the extended L^* algorithm and the thought of machine learning to construct the protocol specification FSM model based on a normalized model—Symbolic Parameterized Extended Finite State Machine (SP-EFSM). The proposed model guides the specific implementation of the testing.

2.3 Stateful network protocol fuzzing

Lately, research on stateful network protocol fuzzing has matured. Exceedingly more fuzzers can discover vulnerabilities of the stateful network protocols. Banks et al.[5] developed a Stateful NetwOrk prOtocol fuzZEr (*SNOOZE*), which allows users to describe a state operation of the protocols and the messages that need to be created in that state. Abdelnur et al.[6] proposed a fuzzer named *KiF* for stateful Session Initiation Protocol (SIP), and discussed the usage of detecting vulnerabilities caused by software

failures. Raniwala et al.[7] proposed the Linked-aware Reliable Transport Protocol (*LRTP*) method for stateful transport protocols. Alrahem et al.[8] proposed *INTERSTATE*, which is a fuzzer for stateful SIP. Chen et al.[9] used a protocol description language to describe the format of a network protocol so that the generated test cases can be effective. Kitagawa et al.[10] presented the *AspFuzz*, a state-aware protocol fuzzer for application. The *AspFuzz* can discover vulnerabilities caused by neglecting the order of the protocol states or message sequences. Akbar and Faroop[11] proposed a security framework for Real-time Transport Protocol (*RTP*) fuzzing, named *RTP*-miner. Gorbunov and Rosenbloom[12] designed an automated network protocol fuzzing framework based on an open-source framework, named *AutoFuzz*. This framework constructs an FSM for network protocols to extract protocol specifications, and guides test case generation. Li et al.[13] proposed a method that automatically identifies a variety of network protocols and generates a fuzzer for vulnerability discovery. Sui et al.[14] introduced a method that combines stochastic signal processing with regular expressions to guide test case generation, and then performs fuzzing to test the protocol robustness. Tsankov et al.[15] proposed a light-weight technique for fuzz-testing security stateful protocols and used a set of fuzz operations to performed mutation. Seo et al.[16] put forward a stateful rule tree algorithm for stateful network protocol fuzzing. To generate test sequences, it maps each state and SIP grammar rule. Pan et al.[17] proposed a model-based testing method for network protocols. Using the protocol specification, the method builds up a formal model for the input data, constructs the corresponding syntax tree for the selected test nodes, and then uses these nodes to guide the test case generation. Ma et al.[18] presented a fuzzing case generation method for network protocols using a classification tree and heuristic operators to reduce the number of test cases. Later, Ma et al.[19] proposed a new method, using a rule-based state machine and a stateful rule tree to guide the generation of fuzz testing cases.

2.4 Defects of traditional stateful network protocol fuzzing

Although network protocol fuzzing has been widely used and can discovery vulnerabilities effectively, there still exist some issues for the stateful network protocol:

(1) Lack of support for state information: Owing to the characteristics of stateful network protocols, test case generation should consider not only each state, but also the entire state trajectory. Because traditional fuzzing does not include contextual information and all states of a message sequence, test cases generated for each state are discrete and cannot cover the entire state trajectory. Thus, traditional fuzzing cannot discover vulnerabilities in the state transitions.

(2) Many redundant test cases: Traditional fuzzing generates test cases randomly, so there must be numerous redundant test cases neglected by the target protocol.

(3) Lack of specificity and reliance on manual operations: Testers define specifications to reduce the blindness of test case generation. However, significant reliance on manual operations may lead to lower coverage and incomplete testing.

For the above issues, this paper proposes the SFSNP. This method can construct the protocol FSM according to the protocol interaction. It can also decrease the size of generated fuzz testing case sets and improve the quality of fuzz testing cases using the state transition marking algorithm and the semi-valid algorithm.

3 Semi-valid Fuzz Testing Case Generation for Stateful Network Protocol

3.1 Overview

Based on the state transition marking algorithm and the semi-valid algorithm, this paper proposes the SFSNP method based on the extended FSM. The basic idea and main steps of the method are as follows.

First, analyze the target network protocol and mark each state transition path in the state machine. Then, we will obtain the extended FSM model with the marking variables. In this network protocol FSM, every state transition path is a tested object.

Second, obtain test sequences of the network protocol FSM using the test sequence generation method based on the FSM. Each obtained test sequence is composed of multiple state transitions.

Third, reduce the redundancy rate of test sequences and shrink the size of the test case sets using the marking algorithm.

Finally, divide the protocol message into several sub-protocol domains using the protocol stateful rule tree. Then, at the level of sub-protocol domain, create the "brute" semi-valid fuzz testing cases for the network protocol using the semi-valid algorithm. Thus, it could

improve the quality of the generated fuzz testing cases for the network protocol.

3.2 Constructing extended FSM abstract model

The network protocol model based on the FSM is a quintuple represented by $P_{\text{FSM}} = < s_0, S, M, f, L >$.

s_0 represents the initial state of the state set. It is the start of the entire state space. From this state, it can reach any other state after a series of state transitions.

$S = \{s_0, s_1, s_2, ..., s_{n-1}\}$ represents the state set of FSM.

$M = \{m_0, m_1, m_2, ..., m_{n-1}\}$ represents the test case set. It contains the protocol messages enabling the FSM to perform state transitions.

f: $S_i \times m_i \rightarrow S_j$ is the state transition function. It represents the transfer relation between the states.

$L = \{l_0, l_1, l_2, ..., l_{n-1}\}$ is the marking variable set of the state transition. It indicates whether the fuzzing has been completed. The default value of l_i is *false*.

A directed graph, as shown in Fig. 1, can represent a network protocol described by the FSM. In this graph, nodes represent states and directed edges represent state transitions. From Fig. 1, the network protocol FSM model perfectly reflects that the protocol states can transfer from some states to corresponding states under the message-driven protocol. It also reflects that the stateful network protocol is context-sensitive, and is related to data and its historical trajectory.

3.3 Obtaining test sequences for the network protocol

To obtain the test sequences of the network protocol FSM, we first provide some definitions as follows.

State transition: A state transition is defined as $\sigma = (s_i, m_i, s_j)$, where $f(s_i, m_i) = s_j$, $m_i \in M$. Here s_i represents the state before state transition σ; s_j represents the state after state transition σ; m_i is the protocol message, which enables the FSM to transfer from state s_i to state s_j.

Test sequence: A test sequence includes multiple state transitions. Hence, for the network protocol FSM, a whole test sequence can be formally expressed as $t = < \sigma_1, \sigma_2, ..., \sigma_{n+1} >$, where $\sigma_i = (s_i, m_i, s_j)$. The test sequences contained in Fig. 1 are $t_1 = < (s_0, m_0, s_1), (s_1, m_1, s_2), (s_2, m_3, s_4) >$ and $t_2 = < (s_0, m_0, s_1), (s_1, m_2, s_3), (s_3, m_4, s_4) >$.

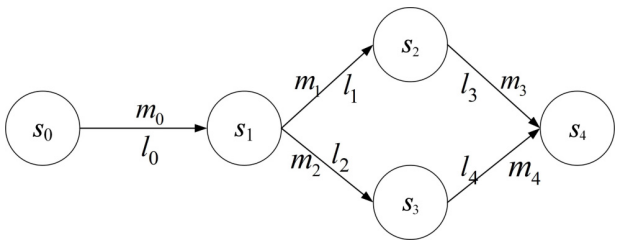

Fig. 1 State transition of network protocol FSM.

A single state transition represents a protocol FSM transfer from one state to the next state using the message driven protocol. In network protocol fuzzing, the state transition is the basic unit of the obtained test sequences. A fuzzing sequence includes multiple state transitions, which could follow and embody the interactive process of the protocol.

3.4 Marking state transitions

There may be the same state transitions between different test sequences. These identical state transitions can directly lead to the generation of redundant fuzz testing cases. It will also result in excessive generated test cases, prolong the testing time, and reduce the efficiency of the test case sets.

Therefore, we provide some principles to determine whether to mark the state transition:

(1) Determine whether the depth testing has been completed for the current state transition. If not, generate the fuzzing sequences for this state transition using the semi-valid algorithm. Otherwise, move to principle (2) for further verification.

(2) Evaluate whether the preceding state of the current state transition is obtained by more than one state transitions. If so, conduct secondary depth testing; otherwise, it is unnecessary to perform the secondary deep testing. Just stop the mutation operation and directly assign the valid request information.

From Fig. 1, there is an identical state transition (s_0, m_0, s_1) in the test sequences t_1 and t_2. When generating the fuzz testing cases, the secondary depth testing is supposed to be performed for the current state transition. However, according to the proposed principles, the secondary depth testing can be omitted in some circumstances. Thus, it is necessary to mark the state transitions of the network protocol state machine to reduce the number of generated fuzz testing cases based on the marking principles.

Algorithm 1 shows the pseudo code of the state transition marking algorithm. This algorithm has two input variables, *Edges* and *TS*. The *Edges* variable represents the state transition set of the protocol state machine. The *TS* variable represents the initial test sequence obtained according to the protocol state

Algorithm 1 State transition marking algorithm

Edges = {state transition set of network protocol finite state machine},
TS = {obtained initial test sequence set for network protocol finite state machine},
Results = {generated fuzz testing case sets for network protocol}

Input: *Edges*, *TS*
Output: *Results*

```
1: while TS is not empty do
2:     SELECT and REMOVE t from TS;
3:     while edge is in t do
4:         SEARCH the match edge in Edges;
5:         if edge's flag is not TRUE then
6:             SET the edge's flag to be TRUE;
7:             CALL the semi-valid model to GENERATE a
   test case t' for t;
8:             Results ← Results +t';
9:         else if edge's preceding states are multiple then
10:            CALL the semi-valid model to GENERATE a
   test case t' for t;
11:            Results ← Results + t';
12:        end if
13:    end while
14: end while
```

machines. The output is *Results*, which is a collection-type variable to store the semi-valid fuzz testing cases.

3.5 Generating semi-valid fuzz testing cases

3.5.1 Concept of the semi-valid test case

A test case, which only violates one constraint in the protocol specification and satisfies all the remaining constraints, is called the semi-valid test case. Therefore, the legitimacy of the semi-valid data may have high probability to bypass the verification. However, at the same time, it still has illegitimacy capability to trigger the vulnerability. Compared with fuzz testing cases for the network protocol created randomly, semi-valid test cases have higher quality and are not easy to be discarded in the early stage of fuzzing. Therefore, the efficiency of the semi-valid test case sets is higher. In addition, the protocol specification has constraints; when constructing malformed data, only the single protocol specification will be damaged. Therefore, the number of generated fuzz testing cases will be reduced accordingly.

Relevant definitions of the semi-valid fuzz testing cases are as follows.

Input data set: The infinite set I represents all input data sets. The input data contained in the set I includes all data types, such as string, message sequence, integer, and so on.

Constraint of input data: c is the constraint of the input data. It is the subset of I, namely $c \subseteq I$. The constraint c defines the range of valid values or the input data formats.

Input data: Symbol i represents the input data, where $i \in c$ means that input data i satisfies the constraint c; otherwise, input data i violates the constraint c.

Valid input data set: The finite nonempty set C includes all constraints of the testing object. Thus, the valid input data set I_{valid} satisfies: $I_{\text{valid}} = \bigcap_{c \in C} c$, which means that valid input data must totally satisfy the constraints.

Semi-valid input data: If input data i is not in the set I_{valid}, that is, $i \notin I_{\text{valid}}$, it is called invalid input data. If the invalid input data only violates one constraint, it is called semi-valid input data. Assume τ_c is a semi-valid input data set. It should satisfy the following formula:

$$\tau_c = \{i \in I \,|\, i \notin c \wedge (\forall c' \in C - \{c\},\ i \in c')\}.$$

Moreover, the semi-valid input data set $I_{\text{semi-valid}}$ is the union set of all sets τ_c.

Figure 2 shows an example of semi-valid input data, where there are three constraints c_1, c_2, and c_3. The intersection of the constraints is a valid input data set, which is represented by $I_{\text{valid}} = c_1 \cap c_2 \cap c_3$; and τ_{c2} is the semi-valid input data. It satisfies two constraints, namely c_1 and c_2. At the same time, it only violates one constraint c_3.

3.5.2 Fuzz testing case generation

To improve the quality of generated protocol fuzz testing cases and more completely describe the network protocol model, one needs to construct the semi-valid protocol fuzz testing cases with finer granularity. Thus, this paper references the idea of the protocol stateful rule tree[20, 21]. According to the construction rules of the protocol stateful rule tree, the protocol comprises four layers, from top to bottom: state

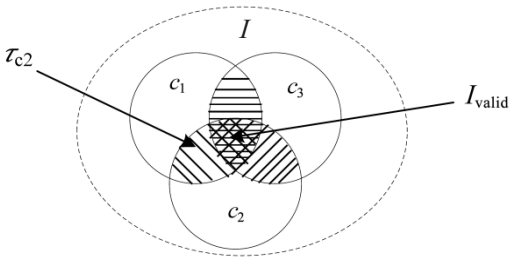

Fig. 2 An example of semi-valid input data.

representation layer, message-type representation layer, message representation layer, and sub-protocol domain presentation layer. Figure 3 shows the schematic diagram of the stateful rule tree. The first layer is the state transition from state s_i to state s_j. In the second layer, the protocol message is divided into the request message and the response message. The third layer is the specific request or response message. The fourth layer is the sub-protocol domains obtained by dividing the specific protocol messages according to the format in the protocol specifications. Because of the fine granularity of the sub-protocol domain, different protocol messages can share the same sub-protocol domain. When constructing malformed packets, the semi-valid algorithm will act on the sub-protocol domains. Because the protocol message is composed of several sub-protocol domains, the malformed protocol messages mainly embody constraint violation of the sub-protocol domain.

The construction process of the semi-valid protocol message is:

(1) Select the protocol message m_i on the current state transition.

(2) Divide the protocol message m_i into sub-protocol domains according to the protocol specification. Then, we obtain the set of sub-protocol domains $m_i = \{field_1, field_2, ..., field_n\}$, which can be presented by the ordered n-tuples $< field_1, field_2, ..., field_n >$.

(3) $I_{\text{valid}} = \{I_1, I_2, I_3, ..., I_n\}$ is a set. Every element in I_{valid} corresponds to a sub-protocol domain and indicates the range of values satisfying constraints. The constraints can be derived from the protocol specification of the targeted network protocol or third-party software.

(4) Set the invalid value set of the sub-protocol domains as the complementary set of the valid value set I_{valid}, that is, $I_{\text{invalid}} = \{\overline{I_1}, \overline{I_2}, \overline{I_3}, ..., \overline{I_n}\}$. According to the definition of the semi-valid fuzz testing cases, we perform the following operations for whole sets of the sub-protocol domains in the protocol message m: every time only one sub-protocol domain in the set violates a protocol constraint. It can obtain the complementary set of valid values in the corresponding domain, as shown in Fig. 4.

(5) After Step 4, the semi-valid protocol messages, such as $m_{i1} = <\sim \boldsymbol{field_1}, field_2, ..., field_n >$, $m_{i2} = < field_1, \sim \boldsymbol{field_2}, ..., field_n >$, ... , $m_{in} = < field_1, field_2, ..., \sim \boldsymbol{field_n} >$, can be obtained in turn. The set $M_i = \{m_{i1}, m_{i2}, ..., m_{in}\}$ is the constructed semi-valid protocol messages.

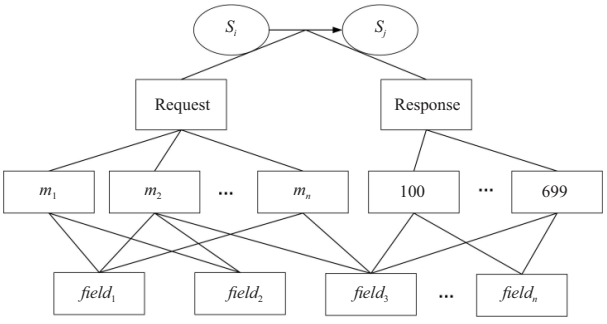

Fig. 3 Network protocol state-rule tree.

4 Experimental and Evaluation

4.1 Experimental environment

In this paper, the verification platform is the open-source fuzzer, namely Sulley. The target protocol is the Simple Mail Transfer Protocol (SMTP). The experimental operating system environment is Windows 7.

4.2 Generating fuzz testing cases for the SMTP

4.2.1 Constructing extended FSM for the SMTP

The SMTP needs to orderly send HELO, MAIL, RCPT, DATA, and QUIT in the transmission of e-mail. To better execute the fuzzing for SMTP and conveniently obtain fuzzing sequences, we obtain the extended FSM of SMTP by analyzing the SMTP specification and the state characteristics of SMTP. Figure 5 shows the SMTP extended FSM.

The extended FSM model of SMTP is $P_{\text{SMTP}} =< s_0, S, M, f, L >$, where:

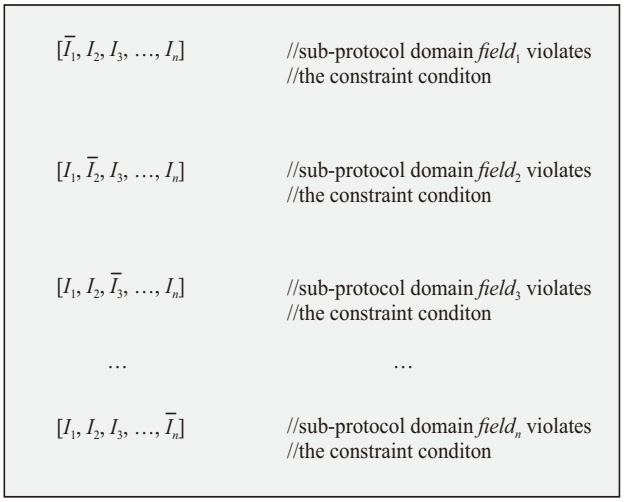

Fig. 4 Semi-valid sub-protocol domain.

(1) s_0 is the initial state of the whole FSM for SMTP.

(2) S is the state set of the SMTP.

(3) M is the protocol message in the FSM for SMTP. It is the fundamental of driving the protocol state machine to perform the state transition. In Fig. 5, set M of the protocol extended FSM as $M = \{$*helo, ehlo, mail from, soml, saml, rcpt to, data, rset, quit*$\}$.

(4) f is the state transition function. It represents the transfer relation between the states.

(5) L is the marking variable set of the state transition. It represents whether fuzzing has been performed. The default value of each element in L is *false*.

4.2.2 Obtaining test sequences for the SMTP

An SMTP command consists of stages: establishing the connection, sending the e-mails, and terminating the connection. Because some vulnerabilities of the protocols can be triggered only when sending specific fuzz testing cases for the protocol under specific state paths, the obtained test sequence sets need to closely cover all state transitions. According to the description of test sequences in Section 3.3, partial original test sequences are obtained, as shown in Table 1.

4.2.3 Marking state transition for the SMTP

From the original test sequences obtained in Section 4.2.2, we observe that there are multiple identical state transitions between them. If the secondary depth testing is performed for each state transition in each test sequence, it will waste time and increase the number of redundant test cases. However, with the use of the state transition marking algorithm and the marking principles, it is unnecessary to perform the secondary depth testing in some circumstances. Meanwhile, the comprehensiveness of the protocol state machine fuzzing is also not affected.

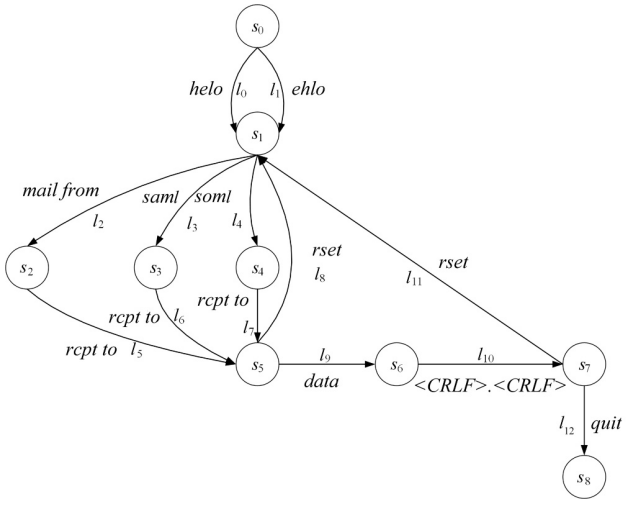

Fig. 5 SMTP extended finite state machine.

Table 1 Partial original test sequences of the SMTP extended FSM.

Name	Original test sequence
Path1	$< (s_0, helo, s_1), (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$
Path2	$< (s_0, ehlo, s_1), (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$
Path3	$< (s_0, helo, s_1), (s_1, saml, s_3), (s_3, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$
Path4	$< (s_0, ehlo, s_1), (s_1, saml, s_3), (s_3, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$
Path5	$< (s_0, helo, s_1), (s_1, soml, s_4), (s_4, rcpt\ to, s_5), (s_5, rset, s_1), (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$
Path6	$< (s_0, ehlo, s_1), (s_1, soml, s_4), (s_4, rcpt\ to, s_5), (s_5, rset, s_1), (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$
Path7	$< (s_0, helo, s_1), (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, rset, s_1), (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$
Path8	$< (s_0, ehlo, s_1), (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, rset, s_1), (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$

Take *Path1* and *Path2* as examples. These two test sequences both include six state transitions. In addition to the first state transition, the remaining five state transitions are all identical. Because the preceding state of the state transition $(s_1, mail\ from, s_2)$ is obtained by two state transitions, it is necessary to conduct the secondary depth testing for this state transition according to the marking principles. But the remaining four state transitions are obtained by only one state transition, so it is unnecessary to perform the secondary depth testing. As a result, fuzzing sequence sets of *Path1* and *Path2* are shown in Tables 2 and 3, respectively. The symbol " $\sim$ " represents that we will

Table 2 Fuzzing sequence sets of *Path1*.

Name	Fuzzing sequence
Path11	$< \sim (s_0, helo, s_1), (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$
Path12	$< (s_0, ehlo, s_1), \sim (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$
Path13	$< (s_0, helo, s_1), (s_1, mail\ from, s_2), \sim (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$
Path14	$< (s_0, helo, s_1), (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), \sim (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$
Path15	$< (s_0, helo, s_1), (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), \sim (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$
Path16	$< (s_0, helo, s_1), (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), \sim (s_7, quit, s_8) >$

Table 3 Fuzzing sequence sets of *Path2*.

Name	Fuzzing sequence
Path21	$< \sim (s_0, ehlo, s_1), (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$
Path22	$< (s_0, helo, s_1), \sim (s_1, mail\ from, s_2), (s_2, rcpt\ to, s_5), (s_5, data, s_6), (s_6, < CRLF > . < CRLF >, s_7), (s_7, quit, s_8) >$

perform the semi-valid operation for the state transition.

Assume the number of generated semi-valid fuzz testing cases is n. According to fuzzing sequences in Tables 2 and 3, there will be $12n$ test cases before marking state transitions, and $8n$ test cases after the marking. There is a reduced number of test cases, about 33% ($(12n-8n)/12n \approx 33\%$). The above calculation is just an example. It also can be inferred that if the state transition marking operation is performed for all test sequences, the number of generated fuzzing sequences definitely decreases.

4.2.4 Constructing semi-valid fuzz testing cases for the SMTP

Semi-valid fuzz testing cases for the network protocol violate only one protocol specification in the construction of the test cases. Therefore, during the construction of the fuzz testing cases based on the protocol stateful rule tree, the necessary work is to extract the protocol constraints. In this paper, the extraction of the SMTP constraints mainly derives from the protocol specification. The library of protocol constraints can be formed by information extraction of the SMTP protocol specification RFC2821[21]. When constructing the semi-valid fuzz testing cases for each protocol message, a negation operation can be performed for each sub-protocol domain of the message based on the library.

Using the command HELO of the SMTP as an example, we introduce the generation of the test cases. Based on the obtained protocol stateful rule tree model, the set of sub-protocol domains is $\{command, space, forward\text{-}path, CRLF\}$. The pairs of sub-protocol domains and corresponding valid values sets are $\{command \rightarrow HELO\}$, $\{space \rightarrow \text{“ ”}\}$, $\{forward\text{-}path \rightarrow \text{“ } < \text{ ”} < mailbox > \text{“ } > \text{ ”}\}$, as shown in Table 4. Next, we can generate semi-valid fuzz testing cases according to the length and content of the command parameters. In Table 4, as constraint 1 described, “the maximum total length of a user name or other local-part is 64 characters”, here 64 is the valid value range of the sub-protocol domain; and in constraint 5, “not including space or special characters” is mentioned when constructing the mailbox strings. Therefore, strings that violate the protocol specification are also the key point when generating test cases.

4.3 Experimental evaluation

The experiment selects MailEnable Professional 4.25, a type of implementation for the SMTP protocol, as the testing target. To verify the SFSNP, we compared

Table 4 Partial SMTP constraints.

1. *The maximum total length of a user name or other* local-part is 64 characters.
2. *The maximum total length of a domain name or* number is 255 characters.
3. *The maximum total length of a reverse-path or forward-path* is 256 characters (including the punctuation and element separators).
4. *The maximum total length of a command line including the* command word and the $< CRLF >$ is 512 characters.
5. $< forward - path > ::= < path >, < path > ::= \text{“ } < \text{ ”} < mailbox > \text{“ } > \text{ ”}, < mailbox > ::= < local - part > \text{“@”} < domain >, < local - part > ::= < dot - string > | < quoted - string >, < dot - string > ::= < string > | < string > \text{“.”} < dot - string >, < string > ::= < character > | < character > < string >, < character > ::= < c > | \text{“/”} < x >,$ $< c >$::= 128 ASCII characters, not including space or special characters, $< x >$::= all 128 ASCII characters

the experimental results of the Sulley and the SFSNP fuzzers in terms of vulnerability discovery, the quantity of generated test cases, and testing execution time.

4.3.1 Vulnerability discovery

The capability of discovering vulnerability is one of the important factors for evaluating the efficiency of fuzz testing case generation approaches.

Table 5 compares the vulnerability information both before and after the proposed SFSNP. The third column represents the results obtained by the Sulley, and the fourth column represents the results obtained by the SFSNP.

Table 5 indicates that both the Sulley and the SFSNP detected one vulnerability when testing the MailEnable Professional 4.25. This vulnerability was published by the China National Vulnerability Database of Information Security, as well as in the Common Vulnerabilities & Exposures. It is a type of remote denial-of-service vulnerability. When MailEnable receives the command "*mail from*" with a very long e-mail address or the command "*rcpt to*" with a very long domain name, its built-in service (MESMTPC.exe) will crash[22].

Moreover, experimental results address that a new valid test case was found after the SFSNP was introduced. The new test case is a long string composed of spaces. In addition, the structure of this string breaks the rule "128 ASCII characters, not including space or special characters". The rule comes from the extraction of the constraints of the protocol specification.

The results show that the same number of vulnerabilities were detected before and after the SFSNP, and the SFSNP achieves the same capability of vulnerability discovery as the Sulley.

4.3.2 Fuzzing efficiency

The efficiency of fuzzing focuses on the quantity of test cases and the testing execution time. Table 6 compares the Sulley with the SFSNP.

Table 5 Effectiveness of test case.

Vulnerability name	Vulnerability no.	Discovering vulnerability	
		Sulley	SFSNP
MailEnable 'MESMTRPC.exe' SMTP server Remote denial-of-service Vulnerability	CVE-2010-2580/ CNNVD-201009-129	Yes	Yes

Table 6 Fuzzing efficiency.

	Test data quantity	Testing execution time
Sulley	11 112	3 h12 min
SFSNP	7884	2 h38 min

(1) Test case quantity

Table 6 indicates that the SFSNP generates test cases that are 71% that of the Sulley, demonstrating that the SFSNP can effectively reduce the number of test cases.

The reason is that the SFSNP adopts the state transition marking algorithm. For some specific commands, the secondary depth testing is not necessary to be conducted. It further avoids repeatedly generating fuzz testing cases. Thus, the SFSNP reduces the redundancy rate and decreases the quantity of generated test cases.

(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 will be higher.

When the Sulley and the SFSNP have the same vulnerability discovering capability, the results in Table 6 indicate that the execution time was reduced by 34 min using the SFSNP, which is nearly 82% of the Sulley.

With the number of test cases reduced, the testing execution time will inevitably be reduced. This data also demonstrates the effectiveness of the SFSNP in reducing the size of test case sets and the testing execution time.

In conclusion, the experimental results highlight that the SFSNP could not only guarantee the capability of discovering vulnerabilities, but also reduce the number of generated test cases and the testing execution time.

5 Conclusion

Recent years, vulnerability discovering for stateful network protocol fuzzing is one of the important focal points in the field of information security. However, the existing network protocol fuzzing fails to consider the state transition of the network protocol. Therefore, traditional network protocol fuzzing is not applicable to stateful network protocol fuzzing. Based on the state transition of the stateful network protocol, this paper proposes the SFSNP. The SFSNP first establishes the extended FSM model with a path marker according

to the protocol interaction. It further obtains the test sequences according to the extended FSM. Then, it performs the semi-valid mutation operations for each state transition in the test sequences. Finally, it obtains the protocol fuzzing sequences. Moreover, different test sequences have the same state transitions when generating protocol fuzzing sequences. That may cause redundant test cases. Therefore, the SFSNP uses the state transition marking algorithm to solve this problem. Experiments were conducted to verify the correctness of the vulnerability discovery, the quantity of test cases, and the testing execution time. The experimental results reveal that the SFSNP could not only guarantee vulnerability discovery, but also reduce the quantity of test cases and decrease the testing execution time.

Our research work still exhibits some defects. For example, the verification is not adequate. Existing theoretical analysis and experimental verification both apply to public protocols. It is still not perfect for the proprietary protocols with an unknown protocol format and an unknown interactive process. Future work is required to determine how to apply the SFSNP to other protocols.

Acknowledgment

This work was supported by the National Key R&D Program of China (No. 2016YFB0800700).

References

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

[2] W. Wang, H. Ding, and Q. Zeng, Research and implementation of test case generation based on formal description, (in Chinese), *Journal of Computer Applications*, vol. 28, no. 4, pp. 1018–1022, 2008.

[3] Z. Zhu, Y. Xu, and M. Zhou, Generation method survey of network protocol testing, (in Chinese), *Computer Engineering and Applications*, vol. 41, no. 15, pp. 172–175, 2005.

[4] G. Shu, Y. Hsu, and D. Lee, Detecting communication protocol security flaws by formal fuzz testing and machine learning, *Lecture Notes in Computer Science*, vol. 5048, pp. 299–304, 2008.

[5] 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.

[6] 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.

[7] 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.

[8] T. Alrahem, A. Chen, N. DiGiussepe, J. Gee, S. Hsiao, and S. Mattox, INTERSTATE: A stateful protocol fuzzer for SIP, presented at DEFCON 15, Las Vegas, NV, USA, 2007.

[9] T. Y. Chen, F. C. Kuo, R. G. Merkel, and T. H. Tse, Adaptive random testing: The art of test case diversity, *Journal of Systems and Software,* vol. 83, no. 1, pp. 60–66, 2010.

[10] 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.

[11] 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, the Netherlands, 2010, pp. 87–92.

[12] 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.

[13] 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 Computers*, vol. 34, no. 2, pp. 242–255, 2011.

[14] 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.*, Ji'nan, China, 2011, pp. 728–731.

[15] P. Tsankov, M. T. Dashti, and D. Basin, SECFUZZ: Fuzz-testing security protocols, in *Proc. 7th Int. Automation of Software Test Workshop*, Zurich, Switzerland, 2012, pp. 1–7.

[16] 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.

[17] 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.

[18] R. Ma, W. D. Ji, C. Z. Hu, C. Shan, and W. Peng, Fuzz testing data generation for network protocol using classificaiton tree, in *Proc. Communication Security Conf.*, Beijing, China, 2014, pp. 97–101.

[19] R. Ma, D. G. Wang, C. Z. Hu, W. D. Ji, and J. F. Xue, Test data generation for stateful network protocol fuzzing using a rule-based state machine, *Tsinghua Science and Technology*, vol. 21, no. 3, pp. 352–360, 2016.

[20] 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, July 14, 2014.

[21] RFC2821, https://www.ietf.org/rfc/rfc2821.txt, April, 2001.
[22] Venustech, Everyday vulnerability weekly newspaper, http://202.85.219.10/NewsInfo/124/8109.Html, Sep. 14, 2010.

Rui Ma received a PhD degree from Beijing Institute of Technology in 2004. She is an associate professor with the School of Software, Beijing Institute of Technology. Her current research interests include software security and Internet of things.

Shuaimin Ren is a master student in the School of Software at the Beijing Institute of Technology. She received the BEng degree from Beijing Institute of Technology in 2016. Her current research interests include software testing and network security.

Ke Ma received the master degree from Beijing Institute of Technology in 2004. He is currently a senior engineer with Internet Center, Institute of Communication Standard Research, China Academy of Information and Communication Technology. His current research interests include information security and IP carrier.

Changzhen Hu received the PhD degree from Beijing Institute of Technology in 1996. He is currently a professor with the School of Software, Beijing Institute of Technology. His current research interest is information 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.