SIP Robustness Testing Based on TTCN-3*

Yang Xiang¹,², Zhiliang Wang¹,³, Xia Yin¹,²
¹Tsinghua National Laboratory for Information Science and Technology
 Beijing, P. R. China, 100084
²Department of Computer Science & Technology, Tsinghua University
 Beijing, P. R. China, 100084
³Network Research Center of Tsinghua University
 Beijing, P. R. China, 100084
{xiangy08, wzl, yxia}@c
snet1.cs.tsinghua.edu.cn

Abstract

Providing flexibility and extensibility, SIP has recently gained significant attention in many areas. Critical requirements on reliability, fault tolerance and security highlight the necessity of SIP robustness testing. There are only a few researches on SIP robustness testing. The biggest challenge is the anomalous message generation. This paper proposes a method and architecture for SIP robustness testing based on TTCN-3, and presents a method for auto-generating large numbers of anomalous message. Based on these, we adopt compound test case to simplify the test suite, and use normal verification sequence to enhance the verdict mechanism. Our method and architecture is extensible for other text based protocols. The test practice on four SIP entities indicate that, our method is efficient to SIP robustness testing, and the test results can provide helpful reference to the upgrade of SIP products.

1. Introduction

Protocol specifications usually contain optional requirements or unsure words, which provide certain flexibility to protocol implementations. The growth of Internet scale and complexity leads to more and more disturbance and vicious attacks. The critical requirements on reliability, fault tolerance and security highlight the necessity of protocol robustness testing.

Protocol robustness testing is to verify whether IUT (Implementation Under Test) can function correctly under invalid inputs or stressful environments [1]. PROTOS [2] is a pioneering project of protocol robustness testing. The tester describes the structure of PDU in BNF (Backus-Naur Form). Lechun Wang et al. [3] studied the robustness testing of inter-domain routing protocol BGP-4. But most of their test cases are used to test semantics of protocol specification. Chuanming Jing et al. [4][5][6] build a novel Nondeterministic Parameterized Extended Finite State Machine model for mutation testing, and test OSPFv2 sufficiently based on extended TTCN-3. The main shortage is that their hex-based mutation strategy is difficult to be applied in text based application-layer protocol.

The Session Initiation Protocol (SIP) [7] is an application-layer protocol. Providing the flexibility and extensibility, SIP has recently gained significant attention in many areas. Therefore, there is an increasing demand for the SIP robustness testing. SIP conformance testing has been studied in previous work. ETSI has developed an abstract SIP conformance test suite [9] which is written in TTCN-3. Tian Li et al. [10] tested the conformance of SIP IUT use their IPv6 tester system.

But the test method and architecture of SIP conformance testing is not applicable for robustness testing. There are only a few researches on SIP robustness testing. PROTOS [2] had test SIP for practice, but the test suite is programmed by C or JAVA, which reduce the maintainability and extensibility. Tian Li et al. [11] introduce a simple test method for SIP robustness testing, but their anomalous messages are all manually generated and the method is hard to extend for other protocols because of the huge workload.

TTCN-3 (Testing and Test Control Notation version 3) [8] is a standard test specification language developed by ETSI. It can be applied in almost all kinds of black-box testing of reactive and distributed
We have developed a test system called PITSv3 [12] based on TTCN-3. Through the analysis of existing method’s defects and the characteristics of text based protocol, this paper presents a method and architecture for SIP robustness testing. The contribution of this paper includes (1) Adopts TTCN-3 to describe the test suite, implements the Coding and Decoding module and IUT Adapter of PITSv3 test system. That provides more flexibility and extensibility, and makes our method and architecture applicable for different SIP IUTs and extensible for other text based protocols. (2) Introduces a method for auto-generating large numbers of anomalous messages. That greatly reduces the workload of programming. (3) Adopts compound test case. That makes the injection of anomalous message more efficiently and reduces the number of test case. (4) Uses the normal verification sequence to enhance the verdict mechanism. (5) Performs the test practice on four open source SIP entities, figure out a bug of twinkle and give some advices to fixing.

This paper is organized as follows. Section 2 briefly gives an introduction to SIP. Section 3 describes our method and architecture of SIP robustness testing. In section 4 we test four open source SIP entities, including two user agents and two registrars. Section 5 gives conclusion and future work.

2. SIP Overview

The Session Initiation Protocol (SIP) [7] is an application-layer signaling protocol for creating, modifying and terminating multimedia sessions with one or more participants. SIP is a text based protocol with simplicity, flexibility and extensibility. It adopts a flexible distributed architecture as shown in Fig. 1. User Agent (UA, such as soft-phones and hard-phones) can be divided into User Agent Client (UAC, creates a new request) and User Agent Server (UAS, generates a response). The proxy server mainly forwards the SIP requests from a UAC to the destination UAS. The redirect server responds to a request issuer with the destination address. The registrar is responsible for the registering of the UA to provide mobility.

Request and response are two types of SIP message. They all consist of a start-line, one or more header fields, an empty line indicating the end of the header fields, and an optional message-body. Fig. 2 shows an example of INVITE message. The method in the start-line of request messages determines the message type, including INVITE, ACK, REGISTER, OPTIONS, CANCEL and BYE. Similarly, according to the status code in the start-line, response messages can be divided into six distinct classes.

3. SIP Robustness Testing

A transaction of SIP consists of a request and at least one response. Fig. 3 shows an example of SIP session flow. User A sends an INVITE request to user B through proxies. If agreeing, User B responses to user A with a 200 OK. After ACK is received by user B, the session initiation is completed. Any participant can send a BYE request when he wants to close the session. After that the other one should send a 200 OK response to confirm the close.
3.1. PITSv3

PITSv3 (Protocol Integrated Test System version 3) [12] is our test system based on TTCN-3. Fig. 4 shows the Architecture of PITSv3. Main function modules include: User Interface; Compiler module; Test Management module; Timer module; Coding and Decoding module; Distributed Running Environment; IUT Adapter. Compiler module can translate test suite described by TTCN-3 to Java code and generate ETS (Executable Test Suite).

3.2. Overview

The goal of robustness testing is to verify whether IUT can function correctly in the presence of invalid inputs or stressful environments. A biggest difference between robustness testing and conformance testing is that the former one needs to inject large numbers of anomalous messages to the IUT. Main defects of existing robustness testing method include:

1. Use programming language such as C or JAVA. That reduces the maintainability and readability. We use TTCN-3 to program the test suite. As a standard test specification language, it provides more readability and maintainability.

2. Generation of anomalous message is not efficient. TTCN-3 doesn’t support auto-generating large numbers of anomalous messages. Manually generating these messages will be a huge task, and makes more difficult to the maintenance and extension. Chuanming Jing et al. [4][5][6] extend TTCN-3 for auto-generation anomalous messages. Its hex-based mutation strategy is useful to low-level protocols such as BGP and OSPF, but difficult to be applied in text based application-layer protocol. Besides, text dealing and numerical computation in TTCN-3 is a little complex. So we use JAVA to implement the mutation process in the Coding module, which can automatically generate large numbers of anomalous message. Then through adopting the compound test case, several anomalous messages can be sent out in one test case. That largely reduces the number of test case and makes the injection of anomalous message more efficiently.

3. Verdict mechanism is too simple. Existing work usually make verdicts through observing whether IUT is crashed or CPU usage is too
high. We adopts the normal verification sequence to enhance the verdict mechanism. After anomalous injection, a normal request will be sent to IUT, verify whether IUT can correctly response.

(4) Test system is usually not generic to other protocols.

We use PITSv3 as our test system, which is based on TTCN-3 and generic to other protocols. Besides, according to the characteristics of SIP, we design some mutation rules for generating anomalous message. They don’t rely on a specific protocol. So only a few alterations need to be done when testing another protocol. That makes our method extendable for different text based protocols.

3.3. Method and Architecture

We focus on the robustness testing of user agent and registrar. They are both end systems, so we can adopt the remote test method [13] and the single-port test structure as shown in Fig. 5. Tester connects to SIP IUT through IP network.

![Fig. 5. SIP Robustness Testing Architecture](image)

We mainly need to implement three modules (bold in Fig. 4 and Fig. 5). ATS (Abstract Test Suite) is a set of test cases described using TTCN-3. IUT Adapter is in charge of transferring messages between tester and IUT. Due to the flexibility of SIP and the requirement of injecting large numbers of message, Coding and Decoding becomes an important module for interpreting messages received from IUT and generating messages send to IUT.

3.4. Test Suite

Protocol robustness testing aims to vulnerabilities of IUT, such as malformed message parsing, buffer overflow and huge resource consumption when lots of anomalous messages are injected. SIP is a text based protocol, so we mainly consider four types of mutations when generating anomalous message:

1. Long field value or reduplicate delimiters.

   SIP message usually use some delimiters to separate fields (i.e., , ; @ = < > SPACE, CRLF). We want to check whether IUT allocated large enough buffer to deal with long field value or long reduplicate delimiters.

2. Special characters and invalid bytes.

   Special characters include control and escape symbols such as “%d” and “0”. Because SIP adopts UTF-8 as its encoded mode, we also want to know whether IUT can correctly deal with invalid UTF-8 bytes. Invalid bytes can be produced according to Table 1.

3. Invalid numerical value.

   There are some numerical fields in SIP message, such as Max-Forwards, CSeq and Content-Length. As a type of mutation, numerical fields can contain an invalid value (i.e., negative value, huge integer or non-numeric string).

4. Wrong position of SPACE or CRLF.

   Field separator SPACE and line separator CRLF are two important separators. Now they may occur in a wrong position.

We design four test cases (corresponding to the above four mutation rules) for every kind of SIP message. All of these test cases formed our test suite of SIP robustness testing.

<table>
<thead>
<tr>
<th>Unicode Character</th>
<th>Corresponding UTF-8 Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000–0000007F</td>
<td>0xxxxxxx</td>
</tr>
<tr>
<td>00000080–0000007FF</td>
<td>110xxxxx 10xxxxxx</td>
</tr>
<tr>
<td>00000800–0000FFFF</td>
<td>1110xxxx 10xxxxxx 10xxxxxx</td>
</tr>
<tr>
<td>00010000–001FFFFF</td>
<td>1111xxxx 10xxxxxx 10xxxxxx</td>
</tr>
<tr>
<td>00200000–03FFFFFF</td>
<td>111110xx 10xxxxxx 10xxxxxx 10xxxxxx</td>
</tr>
<tr>
<td>04000000–7FFFFFFF</td>
<td>1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx</td>
</tr>
</tbody>
</table>

1. `testcase REGISTER_BAD_INT()` runs on MyTestComponentAsync
2. `system SystemComponent {` 
3. `var RES BRANCH bSuf := "1234567890aBeDeFgHi";`
Fig. 6. TTCN-3 Source Code of Test Case ‘REGISTER_BAD_INT’

Fig. 6 is the source code of test case ‘REGISTER_BAD_INT’, which tests the robustness of registrar when there is invalid numerical value in the REGISTER request. It’s worth noting that, the test case does not generate actual anomalous messages. It just appends a tag (the mutation type) in the head of a normal message. TTCN-3 has an excellent ability of controlling test flow. But on the other hand, it does not have sufficient capability to deal with text and computation. Therefore, it is difficult to do some operations in TTCN-3 test cases. These operations include random number generation, data encryption and decryption which are often required in SIP sessions. In order to simplify the programming of test case, the mutation work will be done by the Coding module.

Line 3 to line 7 is variable declarations and port connection. Line 9 to line 15 is the process of anomalous messages injection. As mentioned before, in order to make the injection of anomalous message more efficiently and reduce the number of test cases, we adopt the compound test case. That is to say, several invalid fields of one PDU and several invalid values of one field (with the same mutation type) are injected in one test case. In line 9, a message is tagged with a mutation type and sent to IUT Adapter, then passed to the Coding module. The Coding module will generate numerous anomalous messages according to the mutation type. The ‘badTimer’ in line 10 and line 14 is used to help to consume responses when anomalous messages being injected.

Line 17 to line 28 is the normal verification sequence. In order to verify the status of IUT, a normal REGISTER request is sent to IUT (line 17). After that the tester will attempt to receive the correct response (from line 21 to line 25). If it can’t receive the corresponding response in a given time interval (goodTime indicates), MTC will make a ‘fail’ verdict, otherwise a ‘pass’ will be given.

3.5. Coding and Decoding module and IUT Adapter

Fig. 7 shows the structure of the Coding and Decoding module and IUT Adapter. In order to simplify the programming of test case, and auto-generate large numbers of anomalous messages, we implements the mutation process and regularization function in Coding and Decoding module. Coding module generates a large number of anomalous messages according to the mutation type given by MTC. Decoding module interprets messages received from IUT.
According to the mutation type given by MTC, the Decoding module can generate thousands of anomalous messages in a short time. For example, as mutation type (2), it replaces the text between every two adjacent delimiters with a random long string, which consist of special characters and invalid bytes.

Because of the flexibility, messages (even they are the same type) sent by (different) IUT usually are not the same (i.e., different ranking of headers and different length of field). In order to simplify the programming of test case, we need to normalize the received messages. Decoding module can reconstruct the received message to a regular type, before sending it to MTC. This normalize process also makes our test case applicable for different SIP IUT.

In the PITSv3 test system, IUT Adapter is an important component which is closely related to IUT. Its main function implement in two threads:

(1) Main thread in charge of managing network connection and sending message to IUT. In section 3.4 we mentioned that test case doesn’t generate actual anomalous message. When a message with mutation type arrived, IUT Adapter will call the Coding module to generate anomalous messages, and then send them to IUT one by one.

(2) Receiving thread receives message from IUT. It launches a thread listening in a given port. When a message arrived, it will be sent to the MTC (Main Test Component) after been decoded to an identifiable and regular format. The decoding work is done by the Decoding module.

4. Test Practice

As a practice, we test the robustness of four open source SIP entities. Twinkle [14] and Linphone [15] are both user agent and supporting a multitude of features. They can be used for direct IP phone to IP phone communication. Siproxd [16] and OpenSER [17] are both SIP daemon server. It handles registrations of SIP clients and acts as a proxy/redirect server in the network.

As mentioned before, our method is general for different SIP messages. In the following practice, we take INVITE (for user agent) and REGISTER (for registrar) as two examples. Test results are show in Table 2 and Table 3.

<table>
<thead>
<tr>
<th>Mutation Type</th>
<th>Twinkle</th>
<th>Linphone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>405</td>
<td>Fail</td>
</tr>
<tr>
<td>2</td>
<td>876</td>
<td>Pass</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>Pass</td>
</tr>
<tr>
<td>4</td>
<td>280</td>
<td>Pass</td>
</tr>
</tbody>
</table>

Table 3. Test Result of REGISTER request

<table>
<thead>
<tr>
<th>Mutation Type</th>
<th>Siproxd</th>
<th>OpenSER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>330</td>
<td>Pass</td>
</tr>
<tr>
<td>2</td>
<td>720</td>
<td>Pass</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>Pass</td>
</tr>
<tr>
<td>4</td>
<td>172</td>
<td>Pass</td>
</tr>
</tbody>
</table>

There is only one failed test case in the test practice. One reason is that, after various of version evolution, these product are very mature.

Twinkle failed to pass the first test. After analyze the source code of Twinkle we observed that, if user rejects an INVITE whose ‘method’ field of ‘CSeq’ doesn’t equal to ‘INVITE’, Twinkle will crash because of this following assertion: assert (resp->hdr_cseq.method == INVITE).

In this case, the fixing becomes easy. If an INVITE message’s method field of CSeq doesn’t equal to INVITE, instead of assertion, one can simply discard this request or append it to the failed call history.

5. Conclusion and Future Work

Due to the flexibility and extensibility of text based protocol, SIP robustness testing has its own characteristics and difficulties. This paper introduces a method and architecture of SIP robustness test using TTCN-3 and PITSv3. We separate partial functions to Coding and Decoding module and IUT Adapter, which
provides more flexibility and extensibility. It makes our method applicable for different SIP IUT and extensible for other text based protocols. Anomalous messages are all auto-generated and workload of programming is greatly reduced. By using the compound test case, injection of anomalous message becomes more efficient. In addition, the using of normal verification sequence largely enhances the verdict mechanism.

Through the test practice, we find that the proposed test method and architecture is efficient to SIP robustness testing, and the test result can provide helpful reference to the upgrade of SIP products. Future work will focus on the SIP robustness testing of proxy and redirect server.

6. References