A Robustness Testing Method for Network Security

Yulong Fu and Ousmane Kone

University of Pau and Academy of Bordeaux
yulong.fu@etud.univ-pau.fr, Ousmane.kone@univ-pau.fr

Abstract. In this paper one type of the security problem of DoS (Denial of Service) is studied and transformed to check the robustness of a multiple components system. The network components like attackers, normal clients and the network devices are modeled as implementations of the testing system. And by evaluating the system's robustness, the potential design defects can be detected. The methods on robustness testing of multiple components are studied, and a new model of Glued-IOLTS (Labelled Transition System) is given for defining this kind of multiple and networked system. Then a new approach and algorithm are given for generating the robustness test cases automatically.

1 Introduction

The Denial of Service attack is a normal way of network attacking aiming to crash the network service or make the network resource unavailable. If the system has some potential design defects, it risks to be attacked. As the networking becomes more and more complex and are consisted of different network components, checking defaults are more and more difficult. At the same time, the robustness testing methods consider the problem from the whole system view, and aiming to detect all the possible defects. If we take the system as a multiple components, we can transform this DoS defects checking problem to the problem of robustness testing for a networked and concurrent components.

The concurrent and networked components represent the components which are connected through some kinds of mediums, and communicate with each other to be a concurrent system to achieve some specific functions. While those networked components are generated by different manufactures and implement several network protocols or specifications which are defined by the organisations for standardization like IEEE, ISO...etc. However, different manufactures need their products uniquely and specially, and they will add or extend some functions to their implementations. And because of those expansions and augmentations, the manufactures strongly need to test their products conformance, robustness, and interoperability before they sell them to the market [9]. Related works- In [5] and [1], the authors considered the interoperability testing of two concurrent components, and proposed their C-Methods to describe the concurrent system, then derive the pathes between two components to generate the

H. Jahankhani et al. (Eds.): ICGS3/e-Democracy 2011, LNICST 99, pp. 38–45, 2012.

interoperability test cases. In [8], the authors do well on the robustness testing for the software components, they consider the robustness problems for the closed components by experience. They give a definition of the addition set LSE (language specific error) for "dangerous" input values, and use this "error" set in their "Path generation" to generate their robustness test cases. In [6], the author presents a method to get the extended specification which includes the considered "errors" to present the specification with Robustness. Our work is based on those forward works and go ahead to achieve the problem of robustness testing for the concurrent and networked components.

Our contribution is to extend the IOLTS (Input/Output Labelled Transition System) to give a definition of multiple concurrent and networked components, and then give an approach and algorithm for generating the robustness test cases. We consider black box testing, that is we do note care how the security mechanisms are implemented like in [2], but just whether they are correctly enforced by the network components implementations. This method can be used to design or examine the network protocols including multiple components and different network protocol layer. The method can also be used in the software testing domain.

The following sections are organized as follows: In Section two, we introduce the general testing theories and our testing architecture. In Section three, the formalism based on Labeled Transition System (LTS) is introduced, and our assumptions and approaches are given. In Section four, one case study of concurrent components using RADIUS protocol is given. And the Section five draws our conclusion and future works.

2 Robustness Testing of Concurrent Components

Formal testing methods take the implementations as block-boxes [10], which can be only observed with inputs and outputs. In a specification based testing, a test case is a pair of input and output, which are derived from the specifications, and are executed through the implementation by the tester. The specifications are described as graphs by some kinds of modeling languages like FSM, LTS, etc. The test cases can be taken as traces of the graphs, and the testing methods are trying to see whether the traces of the specifications also exist in the implementations [7]. When running the test case through an implementation, the input sequence of the test case will cause the corresponding output sequence. If the outputs are similar to the pre-defined outputs in the test cases, we say this test case running is successful, and we note "**Pass**" to this test case. Otherwise, the test running is failed, and the test case is marked as "**Fail**" [4].

The concurrent components refer to the networked system which has many local or remote components to work together to finish some functions. Those components are connected through some materials or mediums, and exchange messages and data through them. We considered this concurrent and networked components testing from a simple instance with only two communicated components. The testing architecture is presented in Fig.1. In a concurrent components

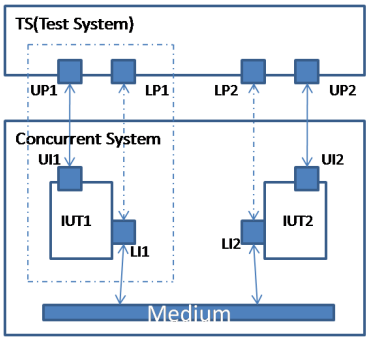

Fig. 1. Test Architecture

testing, each of the IUTs (implementation under test) has two kinds of interfaces. The lower interfaces LI_i are the interfaces used for the interaction of the two IUTs. These interfaces are only observable but not controllable, which means a lower tester(LT_i) connected to such interfaces can only observe the events but not send stimuli to these interfaces. The upper interfaces UI_i are the interfaces through which the IUT communicates with its environment. They are observable and also controllable by the upper tester(UT_i).

Robustness is the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions [3]. Robustness testing concerns the appearance of a behavior which possibly jeopardizes the rest of the system, especially under a wrong input. A system with robustness means it can be executed without crashing, even when it is used inappropriately [6]. We considered the specifications using IOLTS, which emphasize the input and output labels, and then we expand the IOLTS by adding the medium states and transitions into the definition to suit for the requirement of concurrent components.

Labeled Transition System

Labeled transition system is specification formalism studied in the realm of testing, it is used for modeling the behavior of processes, and it serves as a semantic model for various formal specification languages [12].

Definition 1: A labeled transition system is a 4-tuple array $\langle S, L, T, s_0 \rangle$ where

- **S is a countable, non-empty set of states;**
- **L is a countable set of labels;**
- **T is the transition relation, which $T \subseteq S \times (L \cup \{\tau\}) \times S$**
- **s_0 is the initial state.**

The labels in L represent the observable actions which occur inside the system; the states of S are changing just cause of the actions in L. The sign τ denotes the internal and unobservable actions. The definition of T reveals the relations between states in S, for example: $(s_0, a, s_1) \in T$. A trace is a finite sequence of

observable actions. The set of all traces over L is denoted by L^* , and ε denotes the empty sequence. If $\sigma_1, \sigma_2 \in L^*$, then $\sigma_1 * \sigma_2$ is the concatenation of σ_1 and σ_2. $|\sigma|$ denotes the length of trace of σ.

Definition 2: An input-output transition system p is a labeled transition system in which the set of actions L is partitioned into input actions L_I and output action L_U($L_I \cap L_U = \emptyset$, $L_I \cup L_U = L$), and for which all input actions are always enabled in any state. If q ∈ S, then $Out(q)$ denotes all the output labels from q, $In(q)$ denotes all the input labels to q, and $Out(S, \sigma)$ denotes the output of S after σ. $ref(q)$ represents the input actions which are not accepted by state q.

3 Our Approach

As we described in Section 2, the concurrent components communicate each other through a common medium using their lower interfaces, and receive the messages from the environments through their upper interfaces(see Fig.1). We separated the states of each component which are directly connected to the common medium into higher_level states, and we use the low_level states to define the common medium.

Definition 3: The states of the concurrent and networked components system have two levels:

- **higher_level state s_i_u connects to the environment or other states of the same component.**
- **lower_level state s_i_l connects to the states of other components**

A common medium is a subset of the lower_level interfaces of the states, which stimulate the messages to other components. We make S_M to denote all the states in the medium, s_i denote some state in $IOLTS_i$, s_j denote some state in $IOLTS_j$ then

$$\{\forall s \in S_M \mid \exists s_i,\ \exists s_j,\ s = s_i\_l, and\ Out(s_i\_l) \cap In(s_j) \neq \emptyset\}$$

With the help of a common medium, we can glue the components together. We connect the medium states and the stimulated component's initial states with the same label as the medium state received(denoted as L_M). Then the different components are glued.

Definition 4
A Glued IOLTS represents a set of IOLTS $\langle S_i, L_i, T_i, s_i\_0\rangle$ (i=1,n) and a medium M, which is a 4-tuple:
$IOLTS_{glu} = \langle S_{glu}, L_{glu}, T_{glu}, s_{glu}\_0\rangle$, whith

- $S_{glu} = \langle S_1 \times S_2 \times ... \times S_n \times S_M\rangle$,
- $L_{glu} = \langle L_1 \cup L_2 \cup ... \cup L_n\rangle$,
- $s_{glu}\_0 = \langle s_1\_0, s_2\_0, ..., s_n\_0\rangle$ is the initial state,
- $T_{glu} \subset S_{glu} \times L_{glu} \times S_{glu}$
  $T_{glu} = \{(s_1, s_2, ...s_i, ...s_m) \xrightarrow{\alpha} (s_1, s_2, ...s'_i, ...s_m) | (s_i, \alpha, s'_i) \in T_i \cup T_M\}$,
  $T_M = \{(s_i\_l, \mu, s_j\_l) | i \neq j, \mu \in Out(s_i\_l) \cap In(s_j\_l)\}$

One example of Glued-IOLTS is presented in Fig.2 of the next Section.

Robustness testing needs to take into account both normal and threatening inputs. In order to obtain all the possible traces in the concurrent and networked components, first we need to extend the specification to include all the possible inputs actions. We use the so called "Meta-graph" [6] to describe the processes of invalid inputs, and use the "Refusal Graph" [12] to describe the inopportune inputs, then join them to one extended Glued IOLTS: S^+_{glu} to describe all the possible pathes.

Here for a better understanding, we use GIB (Graph Invalid inputs Block) to describe the process of dealing with invalid inputs. By adding the elements of invalid and inopportune input actions, the S^+_{glu} includes all possible actions. We say the implementation of concurrent and networked components is robust if it follows the following conditions:

Definition 5
The implementations of a concurrent and networked components system are denoted as $IUTs$, S_{uni} represents the specification of the those implementations, then:

$$IUTs\ Robust\ S_{uni} \equiv_{def} \forall \sigma \in traces(S^+_{glu}) \Rightarrow Out(IUTs, \sigma) \subseteq Out(S^+_{glu}, \sigma)$$

According to this Definition, to check the robustness of the system, we need to see whether any traces in S^+_{glu} can also be found in its implementations.

So the robustness test case can be generated through the following approach:

- Analyze the compositions' specifications to figure out the concurrent system described using Glued IOLTS S_{glu}.
- Calculate the S^+_{glu}
- Calculate all the possible pathes of the S^+_{glu} to generate the test cases.
- Test Cases run on the implementation. If the implementation can pass all the test cases, the implementation is robust. If not, the implementation fail the robustness testing.

We give an algorithm in Listing 1 to calculate the testing cases automatically. We assume the "initial" states are reachable, and we define the "end" states as the states which after them, the system goes back to the "initial" state or stop. The inputs of this algorithm is the Extended Glued_Specification. The pair $\langle stimulate, reponse \rangle$ denotes the actions between different systems, and the function opt() in the algorithm is to calculate the corresponding actions in this pair. The algorithm uses two recursions to trace back the specifications from the "end" states to the "initial" states. The algorithm uses an arraylist "Trace" to record all the passed labels. When the algorithm reach the "initial" state, it uses the function $Check\_glue()$ to detect the actions inputs from the common medium. If it find that the passed traces need the inputs from the medium, then it adds the corresponding medium labels, and continue to trace back to another system. If it can not find the requirements from the passed traces, the algorithm stops this traceback, and continue to the next trace.

Listing 1.1. Algorithm

```
Inputs: the states of Glued_Specification S,
        the labels of Glued_Specification L;
Outputs: possible trace arraylists trace[m];
int k,m,n=0;
Arraylist trace[m], L_sti[k];
//trace[m] records the passed actions, and m represents different traces.
//L_sti[k] records the actions in one trace which will stimulate another systems.
//k represents different traces.
public main(){
     ArrayList<state> s_end;
     For (int i=0;i<S.size();i++){
          if(S.get(i).getStatus().equals("end"));
               s_end.add(S.get(i));}
     For (int i=0;i<s_end.size();i++){
          Traceback(s_end[i]);
          For(int j=0;j<n;j++){
               Check_glue(trace[j]);}
          For(int j=0;j<n;j++){
               print trace[j];}}}
public trace Traceback(state s){
     ArrayList L= In(s);//arraylist L records all the input actions to state s
     If (s is initial state){
          return trace[m];}
     For(int i=0; i<L.size(); i++){
          trace[m+i].add(trace[m]);
          m=m+i;
          n=m;//count arraylist trace}
     For(int i=0;i<L.size();i++){
          trace[m].add(L.get(i));
          s=L.get(i).pre_state;
          Traceback(s);
          m=m-1;}}
public void Check_glue(arraylist trace){
     For(int i=0;i<trace.size;i++){
          If (trace.get(i) in L_stiulate){
               L_sti.add(trace.get(i));}}
     If L_sti.size()=0{
          return trace;}
     else{
          For(int i=0;i<L_sti.size();i++){
               trace.add(opt(L_sti.get(i)));
               Traceback(opt(L_stiulate.get(i)).pre_state);}
          For(i=0;i<m;i++){
               Check_glue(trace[i]);}}}
```

4 Case Study - RADIUS Protocol

RADIUS protocol is a network protocol between three basic components: client, NAS (network access server), and RADIUS server. The three components connected and worked together, to finish the handshaking and AAA (authentication, authorization, and accounting) security processes. This RADIUS system is a concurrent and networked components system, and we need to use our approach to check the robustness of the implementations of the RADIUS protocol.

Analyze the Specification and Construct the Glued_IOLTS

We take the client as one part of the environment, and in the RADIUS protocol, there are two components: NAS and RADIUS server are considered. Fig.2 presents the Glued Specification of the "Authentication" processes between NAS and RADIUS server according to the standard RFC 2865 [11]. The interactions τ of RADIUS server part represent the processes of security checking.

Calculate the S^+_{glu} and the Possible Traces

By adding the GIB and the refusal graphs at each side of Fig.2 to represent the invalid inputs and the self cycles to represent the inopportune inputs, the S^+_{glu} can be obtained and presented in Fig.3. In this case study, with the help of the algorithm, we got 17 traces by considering the invalid inputs.

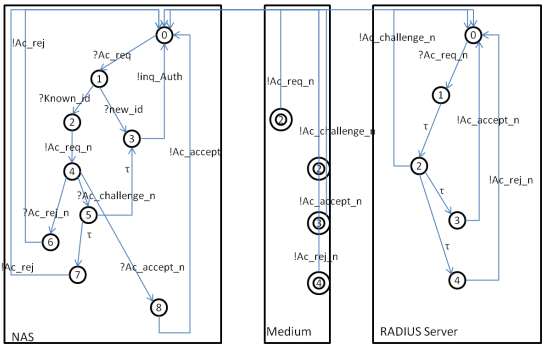

Fig. 2. RADIUS-NAS-Glued-Auth

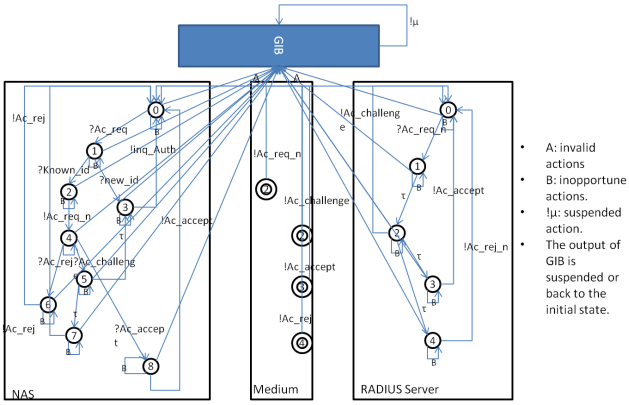

Fig. 3. RADIUS-NAS-Glued-Auth-Plus

Assess the Robustness of the Networked Implementations

After the generations of the test cases, we need to use those test cases to test the implementations. The implementations are tested by checking the outputs with the outputs of the test suites. If the outputs of the implementations are the same as the outputs of the test suites, the implementations are robust. We take the 17th trace of the results: {invalidinput, τ, τ, ?Ac_req_n, !Ac_req_n, ?Known_id, ?Ac_req} as an example. This trace of the RADIUS protocol implies the client(or hacker) sends an access request message(?Ac_req) to the NAS Server to ask for accessing to the protected network, then the NAS Server checks this user's id, and find it is an already known id(Known_id), and send a message {Ac_req_n} with the client's security information to the Radius Server. Then the Radius Server checks the client's encryption method, and find it is supported by the server(the first τ), then it checks the authentication of the client(the second τ), after this, the RADIUS Server receives a undefined message(invalidinput

which is maybe an attack), and the system should terminate this session. If after we input this test case to the system, and find it does not terminate the session, then we know there is a potential risk which exists in the system.

5 Conclusion

In this article, we use a formal method to describe the network components and we extend the definition of Labelled Transition System to model the concurrent component systems. Then we give a definition of robustness of concurrent components system, and give our approach to the robustness test design. We also do an experiment to generate the test cases for the RADIUS protocol. We believe by modeling the network system, and checking its robustness, the potential security defects can be detected and then be fixed.

In this work we did not consider extra functional requirements like real-time constraints. For future work, we also plan to investigate the time conditions required in the network behaviour.

References

1. Ansay, T.: Compositional testing of communication systems-tools and case studies. Master's thesis, Concordia University (2008)
2. Boulares, O., Kone, O.: A security control architecture for soap based services. In: International Conference on Emerging Security Information, Systems and Technologies, SECURWARE (2010)
3. Castanet, R., Kone, O., and Zarkouna. Test de robustesse. In: Proc. of SETIT 2003 (Mars 2003)
4. Desmoulin, A., Viho, C.: Interoperability test generation: Formal definitions and algorithm. In: ARIMA-Numero special CARI 2006, pp. 49–63 (2006)
5. Gotzhein, R., and Khendek, F. Compositional testing of communication systems. IFIP International Federation for Information Processing (2006)
6. Khorchef, S. Un Cardre Formel pour le Test de Robustesse des Protocols de Communication. PhD thesis, University of Bordeaux 1 (2007)
7. Lai, R. A survey of communication protocol testing. Systems and Software 62 (2002)
8. Lei, B., Li, X., and Liu, Z. Robustness testing for software components. Science of Computer Programming, 879–897 (2010)
9. Malek, M., and Dibuz, S. Pragmatic method for interoperability test suite derivation. In: The 24th Euromicro Conference, vol. 2, pp. 838–844. IEEE
10. Offutt, J., Liu, S., and Abdurazik, A. Genearting test data from state-based specification. The Journal of Software Testing,Verification and Reliability, 25–53 (2003)
11. Rigney, C., Willens, S., and Rubens, A. Remote authentication dial in user service (radius). Tech. rep., The Internet Society (2000)
12. Tretmans, J. Conformance testing with labelled transition system: Implementation relations and test generation. Computer Networks and ISDN Systems, 49–76 (1996)