Cadre formel pour le test de robustesse

Application au protocole SSL

Fares SAAD KHORCHEF — Ismail BERRADA* — Antoine ROLLET — Richard CASTANET

LaBRI (CNRS - UMR 5800)
351, cours de la Libération
33405 Talence Cedex - France
{saad-kho,rollet,castanet}@labri.fr

** L3I, Université de La Rochelle*
PST, 17042, La Rochelle Cedex - FRANCE
ismail.berrada@univ-lr.fr

RÉSUMÉ. *Dans le domaine des télécommunications, il est indispensable de valider rigoureusement les protocoles avant de les mettre en service. Ainsi, il faut non seulement tester la conformité d'un protocole, mais il s'avère aussi nécessaire de tester sa robustesse face à des événements imprévus. Dans ce document, nous décrivons une méthode pour tester formellement la robustesse d'un système. Nous expliquons comment augmenter la spécification nominale pour prendre en compte les aléas, puis nous formalisons, à travers une relation binaire, la notion de robustesse entre une implémentation sous test et la spécification augmentée. Ensuite, nous expliquons comment générer des séquences de test à partir de cette spécification augmentée. Enfin, nous proposons une étude de cas sur le protocole SSL en utilisant l'outil TGSE pour la génération de séquences de test.*

ABSTRACT. *In the telecommunications field, protocols have to be seriously validated before their startup. Thus, it is necessary to test the conformance of a protocol, but it is also important to test its robustness in presence of unexpected events. In this document, we give a formal method to test the robustness of a system. Firstly, we explain how to increase the nominal specification in order to take into account the hazards. Secondly, we show how to generate test sequences from this increased specification. Finally, we propose a case study on the SSL protocol, using the TGSE tool for sequence generation.*

MOTS-CLÉS : *Validation de protocole, test de robustesse, test formel, IOLTS, protocole SSL, outil TGSE*

KEYWORDS: *Protocol validation, robustness testing, formal testing, IOLTS, SSL protocol, TGSE tool*

1. Introduction

Le développement des systèmes a tendance à devenir de plus en plus complexe, notamment dans le cas des systèmes critiques. Afin d'éviter des conséquences désastreuses, ces systèmes doivent être validés formellement. Dans le domaine des protocoles, il est notamment nécessaire de tester le système réel. Cette étape consiste à soumettre l'implémentation sous test (ou IUT pour "Implementation Under Test") à diverses épreuves pour s'assurer que son comportement est bien celui attendu.

Depuis quelques décennies, de nombreux travaux ont été effectués dans le domaine du test de conformité. Cependant, il est apparu récemment un besoin au niveau de la robustesse des systèmes : comment le système réagit-il face à des événements imprévus ? Cette réponse ne peut pas être apportées par le test de conformité qui ne tient pas compte, en général, du caractère hostile l'environnement. Pour cette raison, il est aussi nécessaire de tester la robustesse.

La littérature concernant le test de robustesse est beaucoup moins conséquente que le test de conformité. [CAS 03] définit la robustesse comme : "la capacité d'un système à adopter un comportement acceptable en présence d'entrées invalides ou d'environnement stressant". Notre approche se fonde sur l'analyse du comportement du système face à des *aléas*. On considérera comme aléa tout événement non prévu amenant le système à une impossibilité temporaire ou définitive d'exécuter une action.

Les contributions de ce papier se situent principalement à deux niveaux :

1) Proposition d'un cadre formel comportant aussi bien une définition de la notion de robustesse, qu'une méthode pour tester la robustesse d'un système modélisé sous forme d'IOLTS (voir définition plus loin). Notre approche intègre les aléas représentables dans la spécification nominale, pour obtenir une *spécification augmentée*. Cette dernière servira de base pour la génération de séquences de test de robustesse.

2) Mise en pratique du cadre introduit précédemment à travers le protocole SSL.

Cet article est organisé de la façon suivante. La section 2 introduit les modèles, définitions et notations utilisés par la suite. La section 3 présente une définition et une classification des aléas. La section 4 propose une formalisation et une méthode de génération pour tester la robustesse. La section 5 présente une étude de cas sur le protocole SSL. L'état de l'art est présenté dans la section 6. La section 7 est réservée aux conclusions et perspectives.

2. Concepts de base

Cette section est consacrée aux modèles et notations utilisés dans cet article.

2.1. *Système de Transitions Étiquetées à Entrées/orties*

Définition 1 ([TRE 96]) *Un système de transition étiqueté ($IOLTS$) est un quadruplet $S = (Q, A, \rightarrow, q_0)$ tel que : Q est un ensemble fini d'états, q_0 est l'état initial, A est l'alphabet d'actions et $\rightarrow \subseteq Q \times A \times Q$ est une relation de transition.*

L'alphabet $A = A_O \cup A_I \cup I$ est partitionné en trois sous-ensembles : A_O est l'alphabet de sortie (une sortie est notée par $!a$), A_I est l'alphabet d'entrées (une entrée est notée par $?a$) et I est l'alphabet d'actions internes (une action interne est notée par τ).

Soit $S = (Q, A, \rightarrow, q_0)$ un $IOLTS$, $a_i \in A \backslash I$ une action observable, τ_i une action interne, $\sigma \in (A \backslash I)$ une séquence d'action observables et $q, q' \in Q$. Les notations standards des $IOLTS$ sont :

- $q \stackrel{\varepsilon}{\Rightarrow} q' \quad \equiv_{def} \quad q = q'$ ou $q \stackrel{\tau_1 \ldots \tau_n}{\rightarrow} q'$.
- $q \stackrel{a}{\Rightarrow} q' \quad \equiv_{def} \quad \exists\, q_1, q_2 \,|\, q \stackrel{\varepsilon}{\Rightarrow} q_1 \stackrel{a}{\rightarrow} q_2 \stackrel{\varepsilon}{\Rightarrow} q'$.
- $q \stackrel{a_1 \ldots a_n}{\Rightarrow} q' \quad \equiv_{def} \quad \exists\, q_0 \ldots q_n \,|\, q = q_0 \stackrel{a_1}{\Rightarrow} q_1 \stackrel{a_2}{\Rightarrow} \ldots \stackrel{a_n}{\Rightarrow} q_n = q'$.
- $q \stackrel{\sigma}{\Rightarrow} \quad \equiv_{def} \quad \exists\, q' \,|\, q \stackrel{\sigma}{\Rightarrow} q'$.
- $q\, after\, \sigma \quad \equiv_{def} \quad \{q' \in Q \,|\, q \stackrel{\sigma}{\Rightarrow} q'\}$, l'ensemble d'états atteignables à partir de q par σ. Par extension, S after $\sigma \quad \equiv_{def} \quad q_0$ after σ.
- $Traces(S) \equiv_{def} \quad \{\sigma \in A^* \,|\, q_0 \stackrel{\sigma}{\Rightarrow}\}$, les séquences observables de S.
- $Out(q) \quad \equiv_{def} \quad \{a \in A_O \,|\, q \stackrel{a}{\Rightarrow}\}$, l'ensemble des sorties possibles de q.
- $Out(S, \sigma) \equiv_{def} \quad Out(S \text{ after } \sigma)$.
- $ref(q) \equiv_{def} \quad \{a \in A_I | q \stackrel{a}{\not\rightarrow}\}$, l'ensemble des entrées non applicables dans q.

S est dit *déterministe* si chaque état admet un successeur unique par la même action observable. Formellement, $\forall a \in A \backslash I, \forall q \in Q$, si $q \stackrel{a}{\Rightarrow} q_1$ et $q \stackrel{a}{\Rightarrow} q_2$ alors $q_1 = q_2$. Il est dit *observable* si aucune transition n'est étiquetée par τ. S est dit *input-complet* s'il accepte toute entrée a dans chaque état ($\forall a \in A_I, \forall q \in Q$, alors $q \stackrel{a}{\Rightarrow}$).

2.2. *Automate suspendu [TRE 96]*

Trois types de blocage (ou absence d'actions) peuvent être détectés dans un état q d'un $IOLTS$: blocage (*deadlock*) si $\forall a \in A, q \stackrel{a}{\not\rightarrow}$, blocage de sortie (*outputlock*) si $\forall a \in A_O, q \stackrel{a}{\not\rightarrow}$, ou blocage vivant (*livelock*) si $q \stackrel{\varepsilon}{\Rightarrow} q$.

Définition 2 *L'automate suspendu associé à $S = (Q^S, A^S, \rightarrow_S, q_0^S)$ est un $IOLTS$ $S^\delta = (Q^S, A^{S^\delta}, \rightarrow_{S^\delta}, q_0^s)$ tel que : $A^{S^\delta} = A^S \cup \{\delta\}$ avec $\delta \in A_O^{S^\delta}$ (δ est considéré comme une sortie). $\rightarrow_{S^\delta}$ est obtenu à partir de $\rightarrow_S$ par l'addition d'une boucle $q \stackrel{\delta}{\rightarrow} q$ pour chaque état de blocage.*

2.3. *Meta-graphe*

Le meta-graphe sera employé pour modéliser les comportements du système en présence d'aléas. En fait, chaque état dans ce graphe est associé à un ensemble d'états ayant le même comportement en présence des mêmes aléas.

Définition 3 *Soit $S = (Q^S, A^S, \rightarrow_S, q_0^S)$ un IOLTS. Un meta-graphe associé à S est le triplet $G = (V, E, L)$ défini par :*

– $V = V_d \cup V_m$ un ensemble d'états. $V_m \subseteq 2^{Q^S}$ est appelé ensemble des meta-états et V_d est appelé ensemble des états dégradés tel que $V_d \cap Q^S = \emptyset$.

– L *un alphabet d'actions,*

– $E \subseteq V \times L \times V$ *un ensemble de transitions.*

Définition 4 (Composition $IOLTS \oplus G$)
Soit $S = (Q^S, A^S, \rightarrow_S, q_0^S)$ un $IOLTS$ et $G = (V, E, L)$ un meta-graphe associé à S. La composition de S et G, notée $S \oplus G$, est l'$IOLTS$ $(Q^{S\oplus G}, A^{S\oplus G}, \rightarrow_{S\oplus G}, q_0^{S\oplus G})$ défini par : $Q^{S\oplus G} = Q^S \cup V_d$, $q_0^{S\oplus G} = q_0$ et

1) $q \xrightarrow{a}_S q' \implies q \xrightarrow{a}_{S\oplus G} q'$

2) $(v, a, v') \in E$ *et* $v, v' \in V_d \implies v \xrightarrow{a}_{S\oplus G} v'$.

3) $(v, a, v') \in E$, $v \in V_m$ *et* $v' \in V_d \implies q \xrightarrow{a}_{S\oplus G} v'$ *pour chaque* $q \in v$.

4) $(v, a, v') \in E$, $v \in V_d$ *et* $v' \in V_m \implies v \xrightarrow{a}_{S\oplus G} q$ *pour chaque* $q \in v'$.

5) $(v, a, v') \in E$ *et* $v, v' \in V_m \implies q \xrightarrow{a}_{S\oplus G} q'$ *pour chaque* $q \in v$ *et* $q' \in v'$

Cette composition consiste à ajouter à S l'ensemble des transitions et des états du meta-graphe G. En effet, pour un état q de S faisant partie d'un meta-état (ensemble d'états) v de G, on ajoute à S l'ensemble des transitions et/ou états de G partant de v. Nous utiliserons cette composition pour intégrer des aléas exprimés sous forme de meta-graphes dans la spécification nominale. La figure 3(c) illustre cette composition.

3. Aléas

Du point de vue du test de robustesse, le terme *aléa* désigne tout imprévu pouvant provoquer l'impossibilité, d'un système ou d'un composant, à exécuter une action.

Dans cette section, nous proposons une extension de la classification des aléas donnée dans [CAS 03]. Cette classification considère la situation vis-à-vis des frontières du système et du testeur.

Situation vis-à-vis des frontières du système. On distingue des *aléas internes* (e.g. débordement de mémoire, défaillance de processeurs, etc), des *aléas externes* (e.g. tentatives d'utilisation, conditions de stress, etc) et des *aléas au-delà des frontière* du système.

Situation vis-à-vis du testeur. Du point de vue du test, les aléas peuvent être regroupés selon leur observabilité, leur représentabilité ou leur contrôlabilité par le testeur. Les différentes combinaisons sont données ci-dessous (Cf. figure 1) :

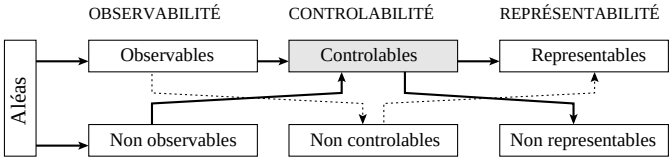

Figure 1. *Classification des aléas.*

1) Aléas observables, contrôlables et représentables. Ils regroupent les *entrées invalides* (e.g. entrées erronées, vides, modifiées etc) et les *entrées inopportunes* (e.g. entrées en retard ou en avance).

2) Aléas observables, contrôlables et non représentables. Ils regroupent les conditions de stress et les pannes contrôlables par le testeur (i.e, le testeur peut manipuler leur exécutions) et qui ne sont pas représentables à travers des entrées ou des sorties.

3) Aléas inobservables, contrôlables et représentables. Ils regroupent les défaillances internes dont l'influence est modélisables par des entrées ou des sorties.

4) Aléas inobservables, contrôlables et non représentables. Ils regroupent certaines défaillances internes commandables par le testeur mais qui ne sont pas modélisables par des entrées dans un modèle formel.

Les autres situations sont éliminées car on ne peut pas tester ce qui n'est pas contrôlable.

Le caractère inconnu des environnements d'exécution pose un problème de prévision d'aléas. En conséquence, il est nécessaire de se concentrer sur le choix d'un ensemble fini d'aléas susceptibles de pouvoir influencer l'exécution de système. Dans ce but, nous proposons de diviser les aléas contrôlables et représentables en trois sous-ensembles :

Entrées invalides. Les entrées invalides regroupent les entrées obtenues par des transformations aléatoires sur les entrées valides (par exemple fautes d'affectation, erreurs d'initialisation, défauts de temporisation).

Entrées inopportunes. Une entrée inopportune désigne toute entrée correcte déjà spécifiée dans le modèle du comportement initial (la spécification nominale) du système mais sa réception n'est pas attendue dans un état donné q (données par $ref(q)$).

Sorties acceptables. Tenir compte des aléas peut mener, dans certains cas, à accepter quelques sorties additionnelles émises par le système. Par exemple, le redémarrage d'un système, la réinitialisation ou la fermeture d'une connexion peuvent être considérés comme des comportements acceptables. Pour éviter des problèmes d'oracle dans ce cas, nous devons ajouter toutes les sorties acceptables dans la spécification initiale.

4. Approche proposée

Dans cette section, nous proposons une approche formelle pour générer les tests de robustesse. D'abord, nous montrons comment augmenter la spécification pour tenir compte des aléas. Ensuite, nous formalisons, à travers une relation binaire, la notion de robustesse d'une implémentation vis-à-vis de la spécification augmentée. Ensuite, nous définissons une technique pour générer les cas de test de robustesse. Enfin, nous appliquons notre approche sur le protocole Handshake (sous-protocole de SSL/TSL). Le plan général est donné dans la figure 2.

4.1. *Augmentation de la spécification*

Le but de la spécification augmentée est de décrire formellement les comportements acceptables en présence des aléas contrôlables et représentables. Notre ap-

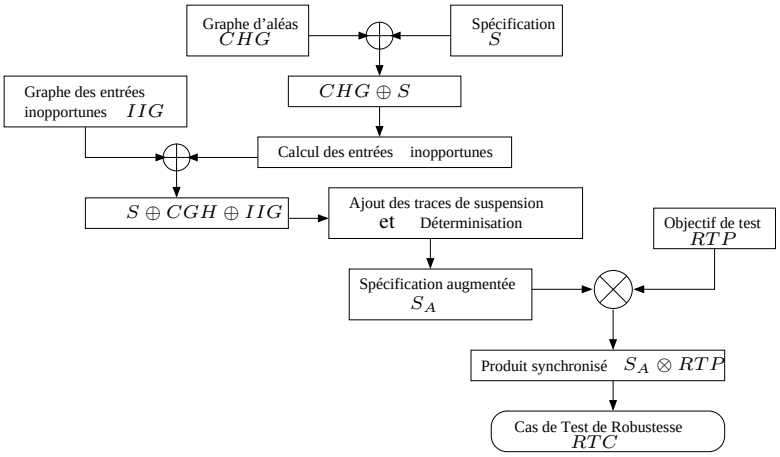

Figure 2. *Plan de l'approche proposée*

proche consiste à exprimer le comportement du système en présence d'un (ou plusieurs) aléa sous forme de meta-graphe(s). Les figures 3 (a) et (b) illustrent ce concept. En supposant que le système S (figure 3 (a)) se trouve à l'état 1, s'il reçoit l'aléa $?a'$, il doit se déplacer, selon le meta-graphe de la figure 3 (b), vers l'état dégradé $d2$ (état de blocage) ; en revanche, s'il émet la sortie acceptable $!x'$, il doit se déplacer vers l'état dégradé $d1$ qui permet au système de retourner au comportement nominal (ici, l'état initial) en cas de réception de $?a$.

Par la suite, nous supposons que les aléas (entrées invalides et sorties acceptables) sont représentés par des meta-graphes CHG (*Controllable Hazard's Graph*). Pour les entrées inopportunes, elles peuvent être calculées automatiquement et elles sont représentées par des meta-graphes IIG (*Inopportune Input Graph*). La spécification augmentée (voir figure 3 (f)) est obtenue comme suit : étant donnés une spécification nominale S et un ensemble d'aléas exprimé par un meta-graphe CHG, la composition $CGH \oplus S$ est effectuée (figure 3 (c)). Ensuite, le calcul des entrées inopportunes de $CGH \oplus S$ est réalisé donnant lieu au meta-graphe IIG (figure 3(d)). Finalement, après la construction de $CGH \oplus S \oplus IIG$ (figure 3 (e)), on procède à l'ajout des traces de suspension et à la déterminisation de $CGH \oplus S \oplus IIG$ (figure 3(f)) pour obtenir la spécification augmentée.

4.2. *Relation de robustesse*

Dans cette section, nous formalisons, à travers une relation binaire, la notion de la robustesse entre une implémentation sous test et la spécification augmentée. Pour ce faire, les hypothèses suivantes sont nécessaires :

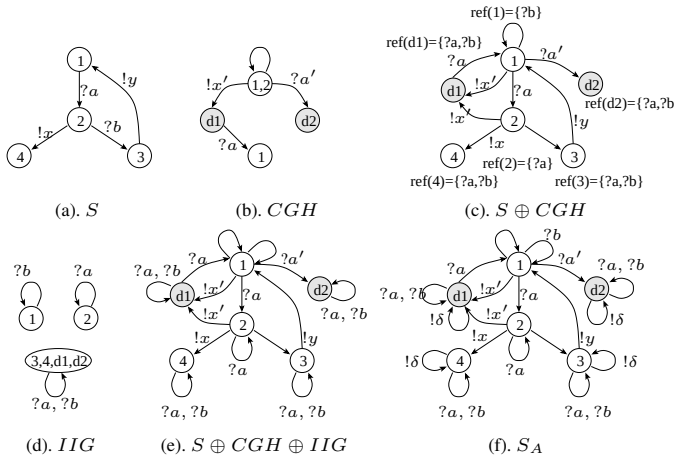

Figure 3. *Augmentation de la spécification*

Modèle de la spécification. Nous supposons que la spécification initiale est modélisable par un $IOLTS$, $S = (Q^S, A^S, \rightarrow_S, q_0^S)$. La *spécification augmentée* est un $IOLTS$ déterministe, observable, et input-complet dénoté par $S_A = (Q^{S_A}, A^{S_A}, \rightarrow_{S_A}, q_0^{S_A})$. Rappelons que $Q^S \subseteq Q^{S_A}$, $A^S \subseteq A^{S_A}$,$\rightarrow_S \subseteq \rightarrow_{S_A}$ et $q_0^S = q_0^{S_A}$.

Modèle de l'implémentation. Dans l'approche du test boîte noire, l'implémentation réelle sous test (IUT) est inconnue. Nous considérons les hypothèses du test suivantes : 1). Les comportements possibles de l'IUT sont modélisables par un $IOLTS$ noté $IUT = (Q^{IUT}, A^{IUT}, \rightarrow_{IUT}, q_0^{IUT})$ tel que $A_I^{S_A} \subseteq A_I^{IUT}$ et $A_O^{S_A} \subseteq A_O^{IUT}$. 2). L'implémentation est supposée input-complète sur son alphabet d'entrée augmentée.

Relation de robustesse. Soit IUT une implémentation, S une spécification et S_A la spécification augmentée de S. La relation de robustesse ***Robust*** est définie par :

$$IUT \ \textbf{Robust} \ S_A \equiv_{def} \forall \sigma \in Traces(S_A) \backslash Traces(S^\delta) \Rightarrow Out(IUT^\delta, \sigma) \subseteq Out(S_A, \sigma). \quad (1)$$

Seuls les comportements augmentés sont visés par le test de robustesse car les comportements nominaux sont déjà vérifiés par le test de conformité (par exemple, selon la relation ioco) .

Exemple 1 *Considérons la figure 4.*

– $IUT1$ ***Robust*** S_A *car toute trace de* $IUT1$ *est incluse dans* S_A

– $\neg$*(*$IUT2$ ***Robust*** S_A*) car* $IUT2$ $after$ $?a'$ *émet* $!y$ *mais* S_A $after$ $?a'$ *émet* $!x'$.

– $\neg$*(*$IUT3$ ***Robust*** S_A*) parce que* $IUT3$ $after$ $?a'$ *émet* $!y$ *contrairement à* S_A.

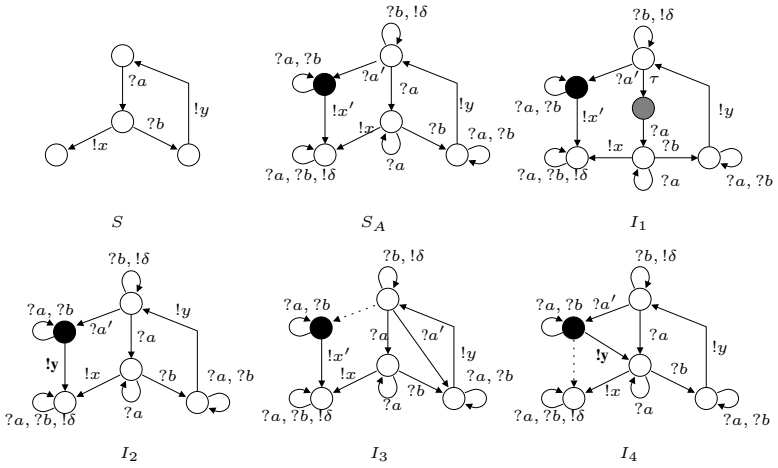

Figure 4. *Relation de robustesse*

– ¬($IUT4$ ***Robust*** S_A*) parce que* $IUT4$ $after$ $?a'$ *émet* $!y$ *(faute de sortie) et atteint un état différent de celui dans* S_A*.*

4.3. *Génération des tests de robustesse*

Après l'intégration des aléas (entrées invalides, entrées inopportune et les sorties additionnelles), la taille de la spécification augmentée devient de plus en plus grande. Pour réduire les coûts du test, nous proposons de sélectionner les tests en utilisant des *objectifs de test*. Cette méthode a été utilisée dans certains travaux sur le test de conformité [BER 05, JER 03]. Le principe de cette méthode est basé sur la synchronisation de la spécification avec un objectif de test.
Le processus de la génération des tests de robustesse peut s'organiser selon les étapes suivantes : 1) Choix d'un objectif de test de robustesse. 2) Synchronisation de cet objectif avec la spécification augmentée afin d'en déduire les comportements qui satisfont cet objectif. 3) Inversion du produit synchronisé, par image miroir. 4) Extraction des cas de test de robustesse. La définition d'un objectif de test de robustesse (RTP pour *Robustness Test Purpose*) doit tenir compte des aspects de fonctionnement[1] ou des aspects relatifs à la réalisation d'une propriété de robustesse. Dans la section 5, nous utilisons l'outil TGSE pour générer les cas de test pour SSL.

5. Etude de cas SSL/TLS

Le protocole SSL (*Secure Socket Layer*) a été développé par Netscape. SSL/TLS permet de l'authentification mutuelle du serveur et du client, le chiffrement et la vérifi-

1. e.g. la sécurité, la confidentialité, l'authentification, etc.

cation de l'intégrité des connexions. Les spécifications standards du protocole (version 1.1) sont données dans la RFC 2246 [HIC 95, DIE 99]. Plusieurs implémentations du protocoles SSL/TLS sont mises en œuvre (Open SSL, SSLeay, BSAFF 3.0, SSL Plus, SSL Ref 3.0, etc).

Le protocole SSL/TLS est constitué de quatre sous-protocoles : SSL *Handshake*, SSL *Changes Cipher Spec*, SSL *Alert* et SSL *Record*. Le protocole *Record* a pour but de chiffrer les connexions avec un algorithme symétrique, et de vérifier leur intégrité à l'aide d'une fonction de hachage de type $HMAC$. Le protocole *Handshake* est composé de deux phases. La première consiste à sélectionner l'algorithme de chiffrement (*cipher*), les échanges des clés de session (*master-key*) et l'authentification du serveur. La deuxième consiste à authentifier le client.
Dans cet article nous nous concentrons sur protocole *Handshake*. Les standards précédents décrivent essentiellement trois modes pour négocier une connexion sécurisée entre le client et le serveur (voir tableau ci-dessous) :

Cas	**Séquences**
Session non identifiée et Client non authentifié	!Client-Hello, ?Server-Hello, ?Client-Master-Key, ?Client-Finished, !Server-Verify, !Server-Finished
Session identifiée et Client non authentifié	!Client-Hello, ?Server-Hello, ?Client-Finished, !Server-Verify, !Server-Finished
Session identifiée et Client authentifié	!Client-Hello, ?Server-Hello, ?Client-Master-Key, ?Client-Finished, !Server-Verify, ?Request-Certificate, !Client-Certificate !Server-Finished

Par ailleurs, quatres modèles de fautes sont traités dans ces spécifications. L'absence du certificat (*No-Certificate-Error*), mauvais certificat (*Bad-Certificate-Error*), certificat non supporté (*Unsupported-Certificate-Type-Error*) et absence d'algorithme du chiffrement (*No-Cipher-Error*). En outre, il y a deux autres messages d'erreur qui ne sont pas signalés dans ces standards [BRA 95]. Il s'agit de *Unsupported-Authentification-Type-Error* et de *Unexpected-Message-Error*.

Afin de générer des cas de test pour vérifier la robustesse de ce protocole, nous allons intégrer les aléas précédents, les entrées inopportunes et les traces de suspension dans la spécification initiale. Pour ce faire, on fait les deux hypothèses suivantes :

1) Le comportement robuste vis-à-vis des entrées inopportunes est représenté par une boucle à chaque état de réception, i.e, le protocole néglige la réception des entrées inopportunes.

2) En présence des aléas *Unsupported-Authentification-Type-Error* ou *Unexpected-Message-Error*, le protocole se comporte comme dans les autres cas d'erreur, i.e, il ferme la connexion.

Les deux hypothèses précédentes peuvent être modélisées par des meta-graphes. La spécification augmentée obtenue est composé de 20 états et 176 transitions. Pour des raisons de visibilité, les entrées inopportunes sont représentées par une boucle "*ref(q)*" pour un état q.

5.1. *Génération de test de robustesse avec TGSE*

Afin de tester la robustesse du protocole Handshake en tenant compte des différentes aléas introduits dans la figure **??**, nous avons défini un ensemble d'objectifs (propriétés) visant à tester le comportement de l'implémentation en présence des aléas au niveau du certificat (No-Certificate-Error, Bad-Certificate-Error, Unsupported-Certificate-Type-Error), au niveau de l'algorithme de chiffrement (No-Cipher-Error) et au niveau des aléas Unexpected-Message-Error et Unsupported-Authentification-Type-Error :

– **RTP1** : Le maintien de la connexion entre le serveur et le client après la détection d'un certificat non approprié.

– **RTP2** : Fermeture de la connexion après la détection d'un problème au niveau de l'algorithme de chiffrement.

– **RTP3** : Le maintien de la connexion entre le serveur lors d'un message d'erreur non attendu.

Signalons que l'ajout des aléas inopportuns et le blocage dans une spécification est géré automatiquement par l'outil TGSE [BER 05]. Le tableau 1 résume les différents résultats obtenus avec TGSE sur une station Linux Fedora 3 (Intel Pentium 4 CPU 1.80GHz, mémoire 128Mo). "Taille RTC" représente la taille moyenne d'un cas de test vérifiant une propriété et "Temps CPU" le temps CPU moyen pour générer un cas de test.

Propriété	Taille RTC	Temps CPU (ms)
RTP1	53	0.9618
RTP2	10	0.3599
RTP3	20	0.15797

Tableau 1. *Expérimentation avec TGSE*

6. État de l'art

Beaucoup de travaux de recherche ont été effectués dans le domaine du test de protocoles et beaucoup de techniques ont été proposées. Ces travaux concernent en grande majorité le test de conformité. Dans cette section, nous nous intéressons plus particulièrement aux travaux sur le test de robustesse d'un système.

[CAS 03] propose un cadre précis pour définir le test de robustesse, une classification des différents aléas, ainsi qu'une étude sur les différentes approches envisageables. Les auteurs s'accordent pour définir la robustesse comme "la capacité d'un système à opérer correctement en présence d'entrées invalides ou d'environnement stressant". Il est possible de séparer le test de robustesse en deux principales catégories : les approches fondées sur l'injection de fautes sur l'implémentation sous test (IUT) et les approches à partir d'une spécification formelle du système.

En ce qui concerne l'approche par injection de fautes, [KAR 95] propose une étude comparative sur trois formes d'injections physiques de fautes : radiations avec des ions lourds, perturbations au niveau des connexions des composants, perturbations électromagnétiques. Ce genre d'approches dites destructives sont en général coûteuses. Pour éviter cela, d'autres techniques sont fondées sur l'injection logicielle de fautes, comme les outils FIAT ([BAR 90]), qui modifie l'image binaire d'un processus en mémoire, FTAPE ([TSA 96]), [REG 05] qui propose d'appliquer aléatoirement des interruptions sur l'IUT, ou encore l'outil BALLISTA ([KRO 98], [DEV 99]) qui utilise une approche fondée sur les types de données en utilisant uniquement l'interface pour générer des tests. [MAR 02] propose une caractérisation des différentes modes de pannes et sur la façon de les détecter sur l'IUT.

L'autre approche consiste à utiliser une ou des spécifications formelles pour générer des séquences de test. La principale difficulté de ce genre d'approches réside dans la prise en compte au niveau formel des aléas. Cette approche a été choisie dans différents travaux : [JER 03, SAA 05, FER 05, ROL 03]. Plus spécifiquement, [ROL 03] propose une approche à base de deux spécifications, une nominale et une dite dégradée qui définit les actions prioritaires du système. Dans ce cas, les aléas sont insérés directement dans les séquences de test obtenues après génération. Dans [FER 05], les auteurs essaient de modéliser toutes les fautes possibles dans une spécification dite "mutante" qui sert ensuite de support à la génération de séquences de test.

7. Conclusion

Dans cet article, nous avons tout d'abord présenté une formalisation et une technique de génération de séquences pour tester la robustesse d'un système à base de spécification formelle sous forme d'IOLTS. Nous avons proposé une méthode qui intègre à la spécification les aléas représentables après génération du graphe de suspension et déterminisation. Ensuite, nous avons utilisé cette spécification augmentée pour générer des cas de test de robustesse en utilisant un objectif de test. Dans un deuxième temps, nous avons proposé une étude de cas sur le protocole SSL, en décrivant comment modéliser les aléas pour SSL, puis en générant des séquences de test à l'aide de l'outil TGSE. Nous avons pu ainsi mettre en évidence la pertinence du test de robustesse, et montrer ainsi l'aspect complémentaire entre le test de conformité et le test de robustesse.

Actuellement, nous travaillons sur une façon de simplifier la prise en compte des aléas pour le concepteur du système. D'autre part, nous travaillons sur la prise en compte des contraintes temporelles dans notre cadre de test de robustesse.

8. Bibliographie

[BAR 90] BARTON J. H., CZECK E. W., SEGALL Z. Z., SIEWIOREK D. P., « Fault Injection Experiments Using FIAT », *IEEE Trans. Comput.*, vol. 39, n° 4, 1990, p. 575–582, IEEE Computer Society.

[BER 05] BERRADA I., FELIX P., « TGSE : Un outil générique pour le test », *CFIP'2005 : Ingénérie des Protocoles*, 29 Mars 2005, p. 67-84.

[BRA 95] BRADLEY J., DAVIES N., « Analysis of the SSL Protocol », rapport n° CSTR-95-021, June 1995, Department of Computer Science, University of Bristol.

[CAS 03] CASTANET R., WAESELYNK H., « Techniques avancées de test de systèmes complexes : Test de robustesse », rapport, 11 2003, Action spécifique 23 du CNRS.

[DEV 99] DEVALE, KOOPMAN J., P., D. G., « The Ballista Software Robustness Testing Service », *Testing Computer Software Conference (TCSC99)*, June 1999.

[DIE 99] DIERKS T., ALLEN. C., « The TLS Protocol Version 1.0, RFC 2246 », rapport, January 1999, Internet Engineering Task Force (IETF).

[FER 05] FERNANDEZ J.-C., MOUNIER L., PACHON C., « A Model-Based Approach for Robustness Testing », LNCS, Ed., *Testing of Communication Systems*, vol. 3502, ifip, may/june 2005, p. 333-348.

[HIC 95] HICKMAN K., « The SSL Protocol », rapport, Feb 9 1995, Netscape Communications Corp.

[JER 03] JERON T., « Génération de tests pour les systèmes réactifs. Un survol des théories et techniques », IRIT, Ed., *ETR2003. Systèmes, Réseaux et Applications*, IRIT, Septembre 2003, p. 105-122.

[KAR 95] KARLSSON J., FOLKESSON P., ARLAT J., CROUZET Y., LEBER G., « Integration and Comparison of Three Physical Fault Injection Techniques », *Predictably Dependable Computing Systems, chapter V : Fault Injection*, p. 309–329, Springer Verlag, 1995.

[KRO 98] KROPP N. P., JR. P. J. K., SIEWIOREK D. P., « Automated Robustness Testing of Off-the-Shelf Software Components », *Symposium on Fault-Tolerant Computing*, 1998, p. 230-239.

[MAR 02] MARSDEN E., FABRE J.-C., ARLAT J., « Dependability of CORBA systems : service characterization by fault injection », *Int. Symposium on Reliable Distributed Systems (SRDS-2002)(Osaka, Japan)*, , 2002, p. 276-285.

[REG 05] REGEHR J., « Random testing of interrupt-driven software », *EMSOFT '05 : Proceedings of the 5th ACM international conference on Embedded software*, New York, NY, USA, 2005, ACM Press, p. 290–298.

[ROL 03] ROLLET A., « Testing robustness of real-time embedded systems », *In Proceedings of Workshop On Testing Real-Time and Embedded Systems (WTRTES), Satellite Workshop of Formal Methods (FM) 2003 Symposium, Pisa, Italy*, , 2003.

[SAA 05] SAAD-KHORCHEF F., DELORD X., « Une méthode pour le test de robustesse adaptée aux protocoles de communication », 29 mars 2005.

[TRE 96] TRETMANS J., « Test Generation with Inputs, Outputs, and Quiescence », MARGARIA T., STEFFEN B., Eds., *Second Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, vol. 1055 de *Lecture Notes in Computer Science*, Springer-Verlag, 1996, p. 127–146.

[TSA 96] TSAI T. K., IYER R. K., JEWITT D., « An Approach towards Benchmarking of Fault-Tolerant Commercial Systems », *Symposium on Fault-Tolerant Computing*, 1996, p. 314-323.