# Génération automatique de politiques de sécurité pour SecSIP

Abdelkader Lahmadi, Olivier Festor

# Génération automatique de politiques de sécurité pour SecSIP[1]

**Abdelkader Lahmadi*, Olivier Festor****

* *LORIA - ENSEM - Nancy Université*
** *INRIA Nancy Grand Est*
*Campus Scientifique*
*BP. 20529, Vandoeuvre-lès-Nancy CEDEX, France*

*{Abdelkader.Lahmadi, Olivier.Festor}@loria.fr*

RÉSUMÉ. *Nous présentons une méthode pour la génération automatique de mesures de protection contre l'exploitation des vulnérabilités connues dans le protocole SIP. Ces contres-mesures sont décrites sous forme de spécifications dans un langage dédié, nommé VeTo. Notre méthode s'appuie sur des algorithmes génétiques pour générer ces spécifications à partir d'un ensemble de messages d'exploits. Ce type d'algorithme, nous a permis de générer de manière automatique des expressions régulières qui capturent au mieux une malformation dans un message d'exploit ou une séquence malveillante de messages. Ces expressions régulières sont ensuite traduites en spécifications VeTo pour alimenter le pare-feu SecSIP dédié à la protection des environnements basés sur le protocole SIP.*

ABSTRACT. *We present a method to automatically generate counter-measures preventing the exploit of known vulnerabilities in the SIP protocol. These counter-measures are built using VeTo language specifications. Our method relies on genetic algorithms to generate these specifications from a set of exploit messages. This type of algorithm allows us to automatically generate regular expressions which match malformed patterns within an exploit message or a malicious sequence of messages. These regular expressions are then coded into VeTo specification to be feed up into the SecSIP firewall dedicated to SIP based environments.*

MOTS-CLÉS : *VoIP, SIP, VeTo, contres-mesures, algorithme génétique, génération, automatique.*

KEY WORDS: *VoIP, SIP, VeTo, specifications, genetic algorithm, automatic, generation.*

1. Ce travail a été partiellement financé dans le cadre du projet ANR VAMPIRE.

## 1. Introduction

Les services liés à la Voix sur IP, se sont largement déployés ces dernières années dans l'Internet et représentent aujourd'hui une part importante des communications aussi bien dans les entreprises que chez les particuliers. Cette technologie repose essentiellement sur le protocole SIP [ROS 02] pour la gestion des sessions entre les différentes parties impliquées dans un échange de données, de voix ou de vidéo. Plusieurs travaux de recherche, des contributions à l'IETF, ainsi que des bulletins de sécurité émis ces dernières années ont montré que le protocole SIP ainsi que la plupart des implantations de celui-ci sont exposés à de nombreux types d'attaques. À titre d'exemple, le monde de la voix sur IP connaît depuis peu des attaques à grande échelle issues de fermes de serveurs de type EC2. Ces attaques exploitables à des fins malveillantes d'ordre social et/ou économique sont essentiellement possible grâce à des erreurs de configuration et/ou la présence de vulnérabilités d'implantation non corrigées et/ou à des faiblesses dans les spécifications des standards de communication y compris dans le protocole SIP utilisé dans de nombreux services. En raison de son déploiement extrêmement fort et de l'augmentation d'attaques à grande échelle, il est devenu indispensable de définir des méthodes de protection pour ce protocole intégrant sa sémantique et ses états pour déceler ces attaques et mettre en place des contres-mesures adéquates. Une contre-mesure efficace contre l'exploitation d'une vulnérabilité requiert la connaissance des équipements et celle des vulnérabilités qui y sont associées. Une vulnérabilité SIP est généralement relative à un ou plusieurs équipements particuliers Cette caractéristique n'est pas supportée pas les pare-feux généralistes qui opèrent souvent au niveau IP.

Dans [LAH 09, LAH 10], nous avons défini un environnement de prévention dédié aux services basés sur le protocole SIP. Cet environnement s'appuie principalement sur deux techniques : (i) la spécification sous forme de règles des vulnérabilités et de leur contexte dans un langage dédié nommé VeTo ; (ii) un moteur d'inspection qui exécute cet ensemble de règles sur les messages SIP qui traversent le réseau. Ce moteur identifie les messages ou séquences de messages qui exploitent une vulnérabilité et applique les contre-mesures associées. Ces techniques permettent d'aboutir à un environnement flexible dans lequel il est facile d'intégrer des nouvelles protections et les associer aux équipements qui les concernent. L'une des problématique que nous avons rencontrée au cours de ces travaux est l'édition de contres-mesures. En effet, lorsque le nombre de vulnérabilités découvertes est faible une édition manuelle est possible. En revanche, dès que le nombre devient important, cette étape d'édition des contres-mesures pour alimenter le pare-feux SecSIP avec leurs spécifications devient lente, fastidieuse et engendre des erreurs.

Dans cet article, nous présentons une méthode basée sur les algorithmes génétiques pour générer automatiquement les spécifications VeTo. Cette méthode s'appuie sur l'analyse syntaxique et les machines d'états associées aux messages d'exploit de la vulnérabilité. Cette analyse a pour but de construire un ensemble d'expressions régulières capables de caractériser une malformation au niveau d'un seul message d'exploit ou une séquence malveillante au niveau d'un dialogue SIP. Ce processus de génération optimise ces expressions régulières pour minimiser les faux positifs et sélectionner les plus pertinentes en couvrant le maximum de messages exploit.

Afin de présenter notre approche, l'article est organisé de la façon suivante : en section 2 nous présenterons le protocole SIP et ses vulnérabilités. Nous présentons aussi les éléments syntaxiques majeurs du langage VeTo. Dans la section 3, nous détaillons notre approche de génération auto-

matique des spécifications VeTo pour des vulnérabilités exploitables par un seul message. Dans la section 4, nous présentons l'extension de notre méthode pour spécifier les contres-mesures des vulnérabilités exploitables par une séquence de messages. Nous concluons et nous présentons les perspectives dans la section 5.

## 2. Contexte et travaux relatifs

### 2.1. *Le protocole SIP et les vulnérabilités associées*

Le protocole SIP [ROS 02] est un protocole à base de transactions utilisé pour la gestion des sessions multimédias. Les sessions de voix représentent un des principaux services gérés par le protocole SIP. Ce protocole repose sur deux entités principales qui sont l'agent utilisateur et le serveur SIP. L'agent se place à différents niveaux de l'architecture et assure l'initiation des transactions. Le second se place soit au niveau du réseau d'accès soit au niveau d'un terminal et joue le rôle de répondeur à des demandes de transactions. Différentes entités implémentant les fonctions à la fois d'agent client et agent serveur sont déployées sur une infrastructure de services SIP : les terminaux, les proxys qui assurent différentes fonctions d'enregistrement, d'authentification, de facturation, ... S'ajoutent à cette faune, différentes passerelles vers d'autres réseaux et services (par exemple des passerelles vers le réseau téléphonique commuté). Les services utilisant ce protocole sont depuis peu la cible d'un nombre croissant d'attaques. SIP subit, comme nombre d'autres protocoles de l'Internet, des attaques classiques (dénis de service, inondations, brute-force,...). Les plus dangereuses et les plus efficaces sont cependant celles qui exploitent directement des faiblesses présentes dans ses implantations ou dans sa spécification. Nous avons identifié trois éléments qui favorisent les attaques au niveau de la spécification du protocole SIP. Le premier élément est le format textuel de ses messages les rendant particulièrement faciles à manipuler. Cette caractéristique qui n'est bien sûr pas une faille en soi est aussi présente dans d'autres protocoles comme HTTP ou SMTP. Le deuxième élément est l'absence d'authentification et d'intégrité fortes imposées. Dans sa spécification, les concepteurs du protocole SIP ne proposent pas, ni n'imposent, de mécanismes particuliers pour assurer un service d'authentification des différents éléments intervenants dans un échange. La spécification ne fait que recommander certains mécanismes comme l'authentification à base de HTTP digest. De même, aucun mécanisme d'intégrité ne fait partie des fonctions imposées par le standard. La troisième faiblesse se situe au niveau du mécanisme de routage des messages SIP. SIP possède son propre routage applicatif entre les différents relais (proxy) lors de l'acheminement de ses messages. Ces relais possèdent la capacité de modifier certains champs de ces messages pour des besoins de routage ou en raison de contraintes particulières du service cible. Cette capacité peut être exploitée par une entité malveillante intermédiaire afin de modifier ces champs et introduire des exploits de types dénis ou vol de service. Au niveau des implantations du protocole SIP, aussi bien les agents utilisateurs que les serveurs possèdent un nombre important de vulnérabilités. Celles-ci sont découvertes par différentes méthodes dont notamment des techniques de Fuzzing [ABD 07]. Ces vulnérabilités sont essentiellement dues à des erreurs de programmation introduites dans les phases de développement des couches logicielles des équipements ou à des erreurs de configuration.

Ces vulnérabilités, que ce soit au niveau des spécifications, des configurations ou au niveau des implantations [KER 09], laissent les réseaux SIP exposés aux attaquants. Pour remédier à ce problème, l'application de *patchs* aux implantations et la révision des spécifications est nécessaire.

Cependant, les délais de production et d'application de ces *patchs* sont généralement importants [RES 03]. Souvent, certains utilisateurs et administrateurs sont même réticents à leur application pour des raisons de conflit des mises à jours ou des délais de planification. De ce fait, des systèmes de prévention comme SecSIP [LAH 09] sont indispensables afin de protéger les réseaux et leurs équipements SIP contre l'exploitation des vulnérabilités existantes et connues mais qui ne sont pas encore éliminées.

## 2.2. *Présentation du langage VeTo*

Dans [LAH 10], nous avons présenté un langage de description des vulnérabilités et de spécification des contre-mesures à déclencher. Ce langage, nommé VeTo, est déclaratif. Une spécification consiste en un ensemble de règles. Une protection contre l'exploitation d'une vulnérabilité est décrite par un ensemble de blocs de règles. Chaque règle résulte de la composition d'une ou plusieurs conditions, composition à laquelle est associé un ensemble d'actions. Les actions sont exécutées lorsque les conditions sont satisfaites. Le langage VeTo offre trois types de blocs pour spécifier les protections contre l'exploitation de vulnérabilités. Le premier bloc est celui du contexte de la vulnérabilité. Ce contexte est défini par un ensemble d'attributs relatifs aux cibles à protéger. Ces attributs prennent comme valeur les URI (Uniform Resource Identifier) des cibles et la date d'expiration du contexte (par exemple la date définie pour l'application d'un patch de correction rendant la protection redondante). Le deuxième type de bloc est celui de définition. Ce bloc spécifie les propriétés d'une vulnérabilité en terme des messages échangés et en terme des champs sur lesquels cette vulnérabilité est caractérisée, i.e. les champs utilisés par une attaque pour réaliser un exploit sur un équipement. Le dernier type de bloc est celui de protection. Ce bloc spécifie le comportement intrinsèque d'une exploitation de vulnérabilité en spécifiant la séquence d'événements qui conduit à la réalisation de l'attaque. Ces spécifications VeTo, décrites via une interface d'administration Web ou via un fichier de configuration, alimentent le pare-feu SecSIP pour être appliquées au trafic SIP d'un réseau VoIP.

### 2.2.1. *Le bloc contexte*

Ce type de bloc décrit l'ensemble des informations relatives à l'environnement d'une vulnérabilité. Il contient essentiellement une identification de l'entité exposant une vulnérabilité. Cette identification est donnée sous la forme d'une URI. Un bloc contexte peut-être associé à une ou plusieurs vulnérabilités. Il est utilisé par le moteur d'exécution SecSIP pour déclencher un suivi et une protection appropriés à partir des informations extraites des messages SIP traversant le réseau. La Figure 1 comporte un exemple d'un bloc contexte. On observe notamment l'attribut *target* pour désigner les cibles de la vulnérabilité. L'attribut *include*, nous permet d'inclure un contexte dans un autre.

### 2.2.2. *Le bloc définition*

Le bloc de définition intègre un ensemble de règles. Chaque règle est composée de deux parties. La première partie est la composition d'un ou plusieurs termes qui représentent des champs d'un message SIP, d'un opérateur nommé @match et d'une ou plusieurs expressions régulières qui représentent le patron à vérifier par les données véhiculées dans le champ concerné. Chaque terme et son expression régulière respective représentent une condition. L'ensemble des conditions à vérifier sont

**Rule 1** Spécification d'un bloc contexte VeTo.

```
CONTEXT GlobalCtx BEGIN
TARGET => udp:phone1.example.com:5060;
TARGET => tcp:phone2.example.com:*;
TARGET => *:softphone1.example.com:*;
CONTEXT END
CONTEXT myCtx BEGIN
INCLUDE => GlobalContext;
LIFETIME => 2009-05-07;
CONTEXT END
```

liées par des opérateurs logiques and ou or. La deuxième partie d'une règle définit une action pour traiter (créer, assigner) des variables VeTo. Ces variables sont utilisées par les blocs de protection pour identifier le comportement vulnérable sur un ou plusieurs messages SIP. La Figure 2 illustre un bloc de définition VeTo.

**Rule 2** Exemple d'un bloc de définition.

```
DEFINITION contactDefs BEGIN
LET: SET[SIP:headers.contact] contacts;
WHEN sip:headers.method @MATCH "^INVITE$" ->
                        LET: EVENT ev_INVITE;
DEFINITION END
```

Dans cet exemple, nous avons défini deux règles. La première règle contient seulement la partie action qui consiste à définir une variable contacts de type Set. Cette variable collecte les valeurs du champ contact d'un message SIP observé par le moteur d'exécution SecSIP. La deuxième règle définit une variable nommée *ev _INVITE* de type *EVENT*. Cette variable est définie lorsque SecSIP observe un message SIP contenant la valeur INVITE dans le champ *method*. Ces règles sont définies dans un bloc de définition nommé *contactdefs*.

### 2.2.3. *Le bloc prévention*

Le bloc de protection repose sur des règles ayant comme partie gauche des patterns d'événements. Ces patterns d'événements décrivent un comportement vulnérable dans des messages SIP. Chaque pattern d'événement utilise des variables de type *EVENT* spécifiées dans les blocs de définition décrites précédemment. Chaque bloc de protection possède un identifiant unique. Il est associé à un bloc de contexte et utilise un ou plusieurs blocs de définition. Le contexte d'un bloc de protection est spécifié par le symbole @ suivi du nom du bloc de contexte. Les définitions utilisées par le bloc de protection sont spécifiées par le mot clé *USES*, suivi des identifiants des blocs de définitions. La Figure 3 présente un bloc de protection nommé *myprotection* qui est associé au bloc contexte *myctx* et qui utilise le bloc de définition *contactdefs*. Ce bloc spécifie une seule règle qui met à jour la variable contacts lorsque l'événement *ev_invite* est observé. Il faut noter que ces variables ont été définies dans le bloc de définition *Contactdefs*. Les blocs de protection supportent différents types

**Rule 3** Exemple d'un bloc de prévention.

```
VETO myProtection@{myCtx} USES ContactDefs BEGIN
(ev_INVITE) => STORE:contacts;
VETO END
```

de patrons d'événements parmi lesquels les patrons de type séquence pour exprimer une séquence d'événements. Une séquence est définie sous la forme suivante : (e1,e2,e3,...). Elle signifie que le patron est valide lorsque l'on observe dans l'ordre l'événement e1 suivi de e2 suivi de e3. Nous pouvons également exprimer en VeTo des patrons de type conséquence pour spécifier qu'un événement est la conséquence d'un autre. Ce patron est décrit sous la forme suivante : (e1(e2)). Il signifie que l'événement e2 est une conséquence directe de l'événement e1. Ce patron est utilisé pour représenter l'occurrence de deux événements au niveau d'un même message SIP.

## 3. Méthode pour la génération automatique des spécifications VeTo

Dans cette section, nous présentons notre approche pour générer des expressions régulières caractérisant un champ malformé d'un message d'exploit d'une vulnérabilité existante. Ces expressions régulières sont utilisées pour générer des spécifications VeTo pour prévenir l'exploitation de cette vulnérabilité. L'architecture globale de notre approche est présentée sur la figure 1.

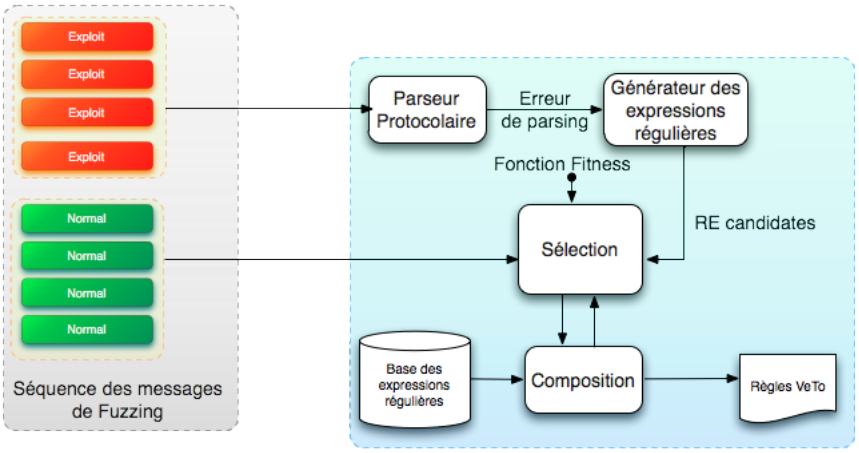


**Figure 1.** *Différents modules utilisés pour la génération automatique des spécifications VeTo.*

Notre méthode s'appuie sur la génération d'un ensemble d'expressions régulières qui décrivent le motif d'un champ malformé dans un message SIP et évaluent leur capacité de caractériser les seuls messages d'exploit. Au départ, l'outil prend comme entrée un ensemble de messages SIP marqués

comme *Exploit* ou *Normal*. Les messages d'exploit sont généralement obtenus depuis un outil de *fuzzing* et ils ont été identifiés comment engendrant une attaque sur un équipement SIP spécifique.

### 3.1. *Génération automatique d'expressions régulières*

Chaque message fourni en entrée à l'outil est analysé par un analyseur syntaxique généré à partir de la grammaire du protocole SIP. Au cours de cette analyse, un message d'exploit déclenche une erreur de syntaxe en indiquant l'identifiant du champ malformé qui a provoqué cette erreur. Une erreur est due soit à une violation syntaxique des règles de la grammaire du protocole SIP, soit à la présence d'un champ inconnu. Tous les messages d'exploit sont groupés en fonction du champ malformé qu'ils exposent. Chaque groupe contient les messages d'exploit qui ont en commun le même identifiant du champ malformé, mais avec des motifs de malformation différents.

Nous générons par la suite, pour chaque groupe, un ensemble d'expressions régulières (*regex*) candidates. Cet ensemble d'expressions régulières, qui caractérisent la chaîne de caractères de la malformation, est généré selon les règles présentées par l'algorithme 1.

**Algorithm 1** Génération des expressions régulières candidates.

**if** chiffre **then**
  remplacer par `\d` ou garder le token (0.5 prob)
**else if** caractère **then**
  remplacer par `\w` or garder le token (0.5 prob)
**else if** séquence de chiffres **then**
  remplacer par `\d+`
**else if** un mot **then**
  remplacer par `\w+`
**else if** un terminal du protocole SIP **then**
  remplacer par ("sip :", "From" , "To", etc)
**else**
  garder le token comme un terminal regex ;
**end if**

Cet algorithme utilise des règles simples pour générer les termes des expressions régulières en analysant la chaîne de caractères de la malformation. La figure 2 montre un exemple d'un champ SIP malformé et l'ensemble des expressions régulières candidates associées.

Dans cet exemple, la chaîne malformée est située au niveau de la valeur du champ $SIP\_URI$ de la requête SIP de type INVITE. Il s'agit du caractère `\255` qui n'est pas compatible avec les règles syntaxiques de ce champ. Comme l'indique la figure 2, nous avons généré un ensemble d'expressions régulières pour caractériser cette malformation. Nous nous intéressons par la suite à l'optimisation de ces expressions régulières candidates pour identifier celles qui caractérisent au mieux un champ malformé tout en respectant ses valeurs dans des messages normaux.

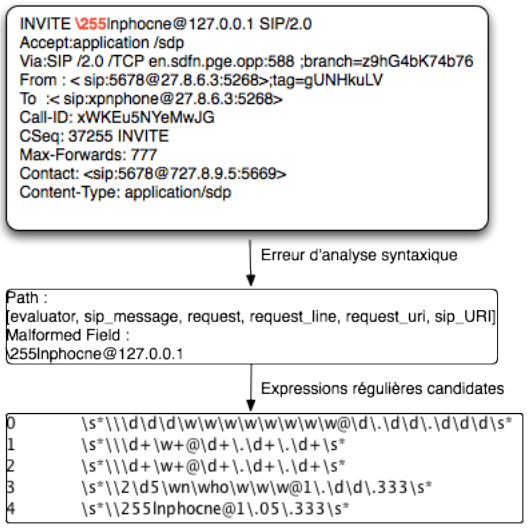


**Figure 2.** *Un exemple d'un champ SIP malformé et ses expressions régulières candidates.*

### 3.2. *Optimisation des expressions régulières*

Notre objectif est d'identifier l'ensemble des expressions régulières qui couvrent les seuls messages d'exploit. Nous avons utilisé en effet, un processus basé sur un algorithme génétique pour optimiser la pertinence de cet ensemble d'expressions régulières candidates. L'algorithme considère chaque expression régulière comme étant un chromosome qui est évalué par une fonction de fitness (*fitness*). Toute expression régulière qui caractérise un message normal est éliminée automatiquement de cet ensemble. Cette étape diminue le risque de générer des expressions régulières qui déclenchent des faux positifs. Nous appliquons ensuite un algorithme de sélection pour identifier les meilleures expressions régulières avec des valeurs de la fonction de *fitness* élevées. Nous appliquons finalement des opérations de croisement et de mutation sur les expressions sélectionnées pour en générer des nouvelles. Ce cycle d'évolution est répété un nombre de fois qui est fixé à 10 dans ce travail. Le cycle d'évolution des expressions régulières est présenté par l'algorithme 2.

#### 3.2.1. *La fonction de fitness*

Nous avons défini une fonction de *fitness* pour mesurer la capacité d'une expression régulière à caractériser un champ malformé. Cette fonction possède deux composantes : le nombre de messages d'exploit couverts par l'expression régulière et sa taille. Cette fonction est définie de la façon suivante :

$$f(regex) = n \times (1 + (\frac{avg - l}{avg})) \quad [1]$$

où $avg$ est la taille moyenne des expressions régulières candidates, $n$ est le nombre de messages d'exploit couverts par $regex$ et $l$ est sa taille respective. Les expressions régulières de petites tailles

**Algorithm 2** Génération génétique des expressions régulières.

Générer un ensemble de Regex en utilisant les règles prédéfinies
**for** Pour i de 1 jusqu'à nombre de génération **do**
Tester l'ensemble de Regex sur les messages normaux et les messages d'exploit
Affecter un score à chacune de regex en utilisant la fonction de fitness
Appliquer l'algorithme de sélection
Recombiner les regex des parents sélectionnés
Appliquer des mutations sur la nouvelle génération de regex
**end for**

et qui couvrent plusieurs messages d'exploit ont la plus grande valeur de $f$. En revanche, les expressions de grandes tailles et qui couvrant plusieurs messages d'exploit ont une valeur de $f$ plus faible score. C'est pourquoi, nous favorisons les expressions régulières de tailles inférieures à la valeur moyenne $avg$.

### 3.2.2. *L'algorithme de sélection*

Le processus génétique que nous avons défini possède pour objectif d'extraire un ensemble d'expressions régulières qui couvrent le maximum de messages d'exploit tout en respectant les messages normaux afin de minimiser les faux positifs. Ce processus génétique s'appuie sur un algorithme de sélection qui utilise la fonction de *fitness* pour identifier ses meilleures expressions régulières. L'algorithme 3 présente ce processus de sélection.

**Algorithm 3** Algorithme de sélection.

P : l'identifiant du champ contenant un motif malformé
MMS(P) = l'ensemble des messages d'exploit contenant une malformation de P
CRES(P) : l'ensemble des expressions régulières candidates caractérisant la malformation de P
CRES(P) = { }
**while** MMS(P) non vide **do**
Évaluer le score de chaque regex
Sélectionner le regex avec le score le plus élevé
Ajouter ce regex à l'ensemble CRES(P)
Supprimer de MMS(P) tous les messages d'exploit couverts par la regex sélectionnée
**end while**

### 3.2.3. *Les opérateurs génétiques*

Pour faire évoluer les expressions régulières, le processus génétique utilise des opérateurs génétiques. Dans notre contexte, nous avons défini deux opérateurs présentés dans la figure 3 à appliquer sur les expressions régulières. La première fonction sert à combiner les expressions régulières parents d'une façon disjointe en utilisant les opérateurs OU, ET et +. La deuxième fonction croise et échange des portions d'expressions régulières.

À la terminaison du processus génétique, nous obtenons les meilleures expressions régulières qui caractérisent une malformation d'un champ spécifique d'un message SIP d'exploit.

```
Premier opérateur : X|Y, XY, X+Y?
Deuxième opérateur : crossover
X: SIP:XXX@\d.\d.\d.\d. ppp
Y: SIP:YYY@\d.\d.\d.\d. fff
XY: SIP:(XXX|YYY)@\d.\d.\d.\d (ppp|fff)
```

**Figure 3.** *Exemples d'opérateurs génétiques.*

## 4. Vulnérabilité incluant plusieurs messages

Le processus que nous avons présenté dans la section précédente est seulement valable pour des malformations syntaxiques au niveau d'un message d'exploit. Ce type de messages est utilisé par un attaquant pour déclencher une attaque instantanée en exploitant un seul message. Dans cette section, nous présentons une extension de notre approche pour générer aussi des spécifications VeTo afin de prévenir des attaques exploitant plusieurs messages pour réaliser un exploit d'une vulnérabilité. Le principe de cette extension est le même que celui présenté dans la section précédente. En revanche, nous allons caractériser sous forme d'expressions régulières une séquence de messages SIP malveillantes ou lieu d'un motif d'une malformation dans un message. Dans un premier temps, l'outil commence par extraire la séquence de messages SIP qui exploite la vulnérabilité. Nous utilisons ainsi un analyseur sous forme d'une machine d'état du protocole SIP pour identifier cette séquence. Nous générons ensuite un ensemble d'expressions régulières candidates qui caractérisent cette séquence malveillante. Nous appliquons le même processus génétique sur ces expressions régulières pour leur optimisation afin d'identifier celles qui couvrent le maximum d'exploits et minimise les faux positifs.

### 4.1. *L'analyseur des machines d'états SIP*

Un dialogue SIP est identifié par une combinaison des champs *Call-ID* et des étiquettes *tag* des champs *From* et *To* d'un message SIP. Une transaction SIP est identifiée par la combinaison des valeurs des champs *branch* et *Cseq*. L'analyseur des machines d'états que nous avons mis en œuvre pour reconstruire les états d'une séquence de messages SIP s'appuie sur deux identifiants. En effet, il crée une entrée d'une transaction au sein d'un dialogue dès l'identification de son premier message qui est de type requête. Chaque message analysé est introduit dans l'entrée correspondante pour chaque transaction. Si ensuite le message analysé respecte son ordre et le nombre de de ses apparitions spécifier par la machine d'états SIP, il sera accepté par l'analyseur, sinon la séquence est identifiée comme malveillante.

### 4.2. *Génération automatique des expressions régulières*

Suite à l'identification d'une séquence malveillante, une expression régulière qui la caractérise est générée. Cette étape de génération utilise l'algorithme 4.

**Algorithm 4** Génération des expressions régulières candidates.

**if** message de type requête **then**
  remplacer par valeur `\method`
**else if** message de type réponse **then**
  remplacer par valeur `\responsecode`
**else if** même requête ou même réponse **then**
  remplacer par `\method{n}` ou `\responsecode{n}`
**else**
  garder le token comme un terminal du regex ;
**end if**

Nous utilisons la concaténation des différentes expressions régulières de différentes transactions pour construire une expression régulière qui caractérise un dialogue SIP. La figure 4 présente un exemple de caractérisation sous forme d'expression régulière d'une transaction INVITE d'un dialogue SIP.

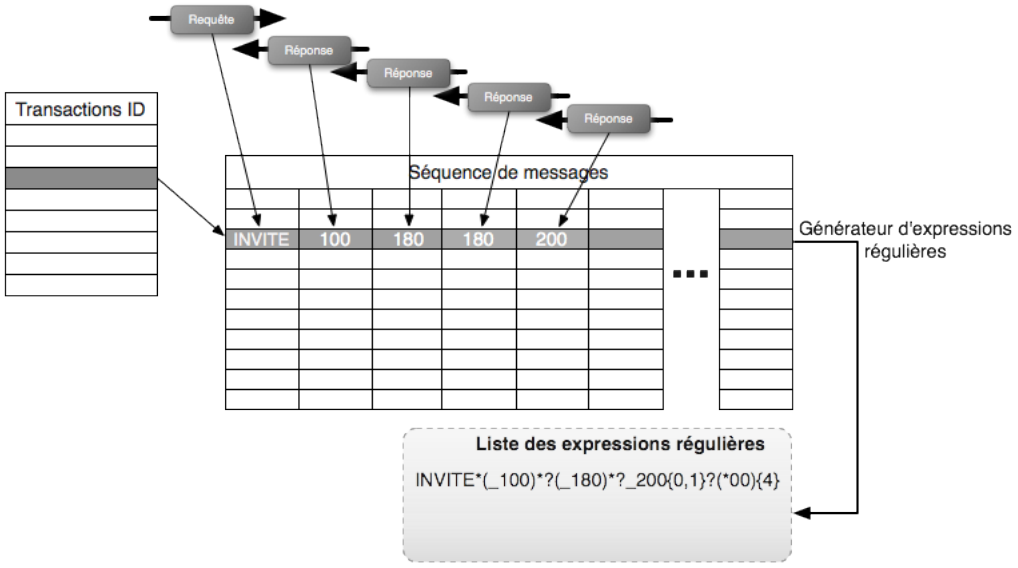


**Figure 4.** *Exemple de génération d'une expression régulière d'une séquence de messages d'une transaction SIP de type INVITE.*

Par exemple, une transaction normale caractérisée par la séquence des messages INVITE, 100, 180,180,200 est transformée en expression régulière $INVITE$_100_180{2}_200. La séquence des messages INVITE, 100, 180, 200, 180, 200 contient une séquence malveillante où le deuxième message 180 est hors d'état.

## 5. Conclusion et perspectives

Dans ce papier, nous avons présenté une méthode de génération automatique de contres-mesures sous forme de spécifications VeTo pour les services basés sur le protocole SIP. Cette méthode s'appuie sur un processus génétique pour générer ces spécification depuis un seul message malformé ou une séquence de messages malveillants. Ces messages sont identifiés par un mécanisme de découverte de vulnérabilités basé sur le *fuzzing*. Nous avons détaillé les différents algorithmes utilisés pour calculer des expressions régulières qui caractérisent ces malformations ou ces séquences exploitables par un attaquant. Cet ensemble d'expressions régulières est optimisé afin d'obtenir les meilleures en termes de couverture des messages d'exploit et un faible nombre de faux positifs.

Actuellement, nous travaillons sur le couplage de la méthode présentée dans cet article, à l'outil de *fuzzing* KIF pour produire d'une manière automatique des politiques de sécurité Veto qui vont alimenter automatiquement le pare-feu SecSIP. Nous nous intéressons aussi à l'optimisation de la taille des expressions régulières caractérisant une malformation ou une séquence malveillante. L'objectif sera d'obtenir des expressions régulières à la fois courtes et pertinentes pour couvrir le maximum des messages d'exploit. Nous travaillons aussi sur l'évaluation de performances de notre méthode en terme de temps de génération et son taux de faux positifs.

## 6. Bibliographie

[ABD 07] ABDELNUR H., FESTOR O., STATE R., « KiF : A stateful SIP Fuzzer », ACM, Ed., *1st International Conference on Principles, Systems and Applications of IP Telecommunications (IPTComm)*, July 2007.

[KER 09] KEROMYTIS A. D., « Voice over IP : Risks, Threats and Vulnerabilities », *In : Proceedings of the Cyber Infrastructure Protection (CIP) Conference*, 2009.

[LAH 09] LAHMADI A., FESTOR O., « SecSip : A Stateful Firewall for SIP-based Networks », *In the proccedings of 11th IFIP/IEEE International Symposium on Integrated Network Management, IM09, Long Island, New York, USA*, Juin 2009.

[LAH 10] LAHMADI A., FESTOR O., « VeTo : An exploit prevention language from known vulnerabilities in SIP services », *IEEE/IFIP Network Operations and Management Symposium, NOMS, 19-23 April 2010, Osaka, Japan*, IEEE, 2010, p. 216-223.

[RES 03] RESCORLA E., « Security holes... who cares ? », *SSYM'03 : Proceedings of the 12th conference on USENIX Security Symposium*, Berkeley, CA, USA, 2003, USENIX Association.

[ROS 02] ROSENBERG J., SCHULZRINNE H., CAMARILLO G., JOHNSTON A., PETERSON J., SPARKS R., HANDLEY M., SCHOOLER E., « SIP : Session Initiation Protocol », RFC 3261 (Proposed Standard), juin 2002, Updated by RFCs 3265, 3853, 4320, 4916.