MASTER EN SCIENCES INFORMATIQUES

SIP et SPAM

Dawirs, Geoffrey

Award date:
2006

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix - Namur
Rue Grandgagnage 21 - B-5000 Namur
Institut d'Informatique

SIP & SPAM

Geoffrey Dawirs

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique

Année académique 2005 - 2006

Résumé

Ce travail consiste en une étude de la problématique du SPAM en SIP. Il dresse tout d'abord un état de l'art des différentes techniques anti-SPAM existantes, certaines provenant du monde du courrier électronique et d'autres étant spécifiques à SIP. Ensuite, deux techniques apparemment prometteuses sont présentées et évaluées : les "Computational Puzzles" et le "Consent Framework". Enfin, la dernière partie du travail se penche sur une solution anti-SPAM originale développée par l'auteur.

Abstract

This document is a study of the issue of SPAM in SIP. First, a state of the art of the main existing anti-SPAM techniques is presented. Some of these techniques were initially designed for e-mail, whereas some others are SIP-specific. Next, two interesting techniques are presented and evaluated : "Computational Puzzles" and "Consent Framework". The last section presents a original anti-SPAM technique designed by the author.

Avant-propos

Je voudrais remercier mes amis et collègues d'Alcatel Massy pour leur accueil chaleureux, leur assitance et leur patience à toute épreuve ! Un remerciement tout particulier va à Thomas Froment pour ses précieux conseils et pour la grande qualité de son encadrement. Grâce à vous tous, j'ai réellement acquis une expérience du monde professionnel. Je vous en suis vraiment reconnaissant.

Merci également à Laurent Schumacher, mon Promoteur des Facultés Universitaires Notre-Dame de la Paix, pour ses relectures avisées et ses judicieuses suggestions d'orientation.

Et "last but not least", merci à mon entourage qui m'a soutenu pendant la réalisation de ce projet.

Acronymes utilisés

AoR	Address of Record
B2BUA	Back to Back User Agent
CPU	Central Processing Unit
DoS	Denial of Service
DTD	Document Type Definition
eSMTP	Extended Simple Mail Transfert Protocol
FAI	Fournisseur d'Accès à Internet
HTTP	HyperText Transfer Protocol
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
IP	Internet Protocol
MIME	Multipurpose Internet Mail Extensions
PSTN	Public Switched Telephone Network
SAML	Security Assertion Markup Language
SCBL	SpamCop Blocking List
SIP	Session Initiation Protocol
TLS	Transport Layer Security
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UE	User Equipment
URI	Uniform Resource Identifier
XCAP	XML Configuration Access Protocol
XML	eXtensible Markup Language

Table des matières

Table des figures

Chapitre 1

Introduction

Le SPAM, défini comme l'envoi de messages non sollicités à des fins généralement commerciales, a perturbé le bon fonctionnement du courrier électronique : la proportion de tels messages est devenue si importante que le coût de leur gestion et de leur élimination a complètement sclérosé ce moyen de communication par lequel ils transitent. Actuellement, on estime que plus des trois quarts [1] des messages circulant sur les serveurs de courrier électronique américains sont du SPAM. En Europe, cette proportion n'est pas si élevée, mais tend chaque jour à augmenter.

De nombreuses techniques visant à protéger le courrier électronique de ces messages non sollicités ont été adoptées. Les plus efficaces parviennent (au mieux) à diminuer le désagrément pour l'utilisateur final, mais à ce jour aucune n'est parvenue à résoudre le problème à la racine. Et pour cause... Le déploiement des services de courrier électronique sur la planète est tel que les solutions au problème du SPAM arrivent probablement trop tard pour être réellement efficaces. Tous les spécialistes se rejoignent sur ce point : il aurait fallu intégrer une protection contre le SPAM dès la naissance du courrier électronique, pour empêcher une telle expansion de ce phénomène plus qu'indésirable.

"Session Initiation Protocol" (SIP), un protocole "orienté texte" permettant l'établissement, la (re) négociation et la clôture de sessions "multimédia", connaît actuellement une expansion très importante et est probablement appelé à devenir un des standards des télécommunications "multimédia". Le risque de voir ce protocole être la nouvelle cible du SPAM est grand, au risque de le voir lui aussi pollué. Il est donc indispensable d'étudier les techniques permettant de combattre le SPAM en SIP, d'identifier les plus efficaces, et de les mettre en oeuvre avant un déploiement planétaire de SIP sur les réseaux ouverts. Ceci constitue l'objet de mon travail, qui vous est présenté dans ce rapport.

Dans un premier temps, une analyse de l'existant a été effectuée : celle-ci fait l'objet de la première partie de ce rapport qui présente les techniques anti-SPAM dérivées du monde du courrier électronique et les techniques dites "nouvelles", c'est-à-dire spécifiques à SIP.

[1] Source : Fontbridge Technologies

Ensuite, les techniques jugées les plus efficaces ont fait l'objet d'une étude approfondie. Si leur étude de faisabilité a rendu un verdict positif, elles ont également été spécifiées et implémentées.

Enfin, l'opportunité de créer un système générique permettant de se prémunir de toutes les formes de SPAM s'est finalement présentée à nous. Son prototype est décrit dans la dernière (et principale) partie de ce travail. Le format de documents de règles de filtrage spécifié dans ce chapitre a été adapté, pour finalement être soumis comme *draft* à l"'Internet Engineering Task Force" (IETF).

En effet, au cours de ce travail, une préoccupation majeure aura guidé les stratégies adoptées et les orientations prises : le souci de respect des standards actuels et futurs de l'IETF ainsi qu'une collaboration étroite avec cet organisme. Cette démarche présente trois avantages majeurs. D'une part, elle permet de garantir que les solutions adoptées sont ou seront standardisées, et qu'elles sont donc interopérables, sans rien imposer aux autres acteurs. D'autre part, elle permet de se tenir informé des derniers choix adoptés par cet organisme de standardisation, ce qui écarte le risque de donner au projet une orientation inadaptée. Enfin, elle rend possible des collaborations fructueuses avec des auteurs de (futurs) standards, ce qui permet d'échanger de nombreuses idées, et de recueillir des avis immédiats sur les stratégies adoptées.

On notera également que certaines décisions d'orientation ont été influencées par les infrastructures disponibles chez Alcatel (outils disponibles, présence préalable de certains serveurs, développement sur la plateforme Alcatel, ...), où le stage de fin d'études a été réalisé. Toutefois, les spécifications des solutions proposées étant standardisées ou en passe de l'être, elles pourraient rester fonctionnelles même en étant réimplémentées dans un autre environnement.

Chapitre 2

État de l'art

La première partie du stage fut consacrée à une étude de l'existant dans le domaine de la prévention du SPAM en SIP. En effet, SIP ne connaissait à l'époque aucune technique largement reconnue comme prévalente pour combattre le SPAM (cette situation n'ayant pas changé à ce jour). Globalement, le paysage des techniques anti-SPAM en SIP se divisait en deux parties. D'une part, on trouvait des propositions d'adaptation des techniques anti-SPAM issues du monde du courrier électronique, même si certaines de ces techniques ne répondaient finalement pas aux spécificités de SIP. D'autre part, des drafts avaient été soumis à l'IETF, proposant de nouvelles techniques, mais elles étaient rarement mûres.

Mais avant de passer à l'étude des techniques proprement dites, plusieurs questions préliminaires devaient être posées...

2.1 Questions préliminaires importantes

2.1.1 Nombre d'URI disponibles en SIP, et dans l'IMS

La première question est de savoir si, en SIP et plus particulièrement dans l"'IP Multimedia Subsystem" (IMS), les utilisateurs peuvent encore obtenir une infinité d'identités (dans notre cas, une infinité d"'Uniform Resource Identifier" (URI)) comme c'est actuellement le cas avec le courrier électronique. Cette question est particulièrement importante pour l'adoption de l'une ou l'autre technique anti-SPAM, car certaines techniques peuvent s'avérer être extrêmement efficaces dans un cas et pas du tout dans l'autre, comme cela sera plus largement détaillé à partir de la section 2.2.1.

Si l'on considère que les seules URI disponibles sont celles fournies par le "Fournisseur d'Accès à Internet" (FAI), alors on peut raisonnablement supposer que le nombre d'URI par utilisateur sera fini, comme c'est le cas avec les comptes "courrier électronique" fournis par les FAI. Toutefois, on peut se demander si on ne risque pas de voir apparaître des services offrant des URI gratuites sur Internet, comme c'est également le cas pour le courrier électronique, ce qui permettrait à tout utilisateur d'obtenir une infinité d'identités SIP.

2.1.2 La question de la falsification du champ “From” en SIP

L’autre problème qui pourrait empêcher la bonne mise en oeuvre de techniques anti-SPAM est la falsification du champ “From” dans les messages SIP, comme c’est également le cas avec le courrier électronique. En SIP, deux mécanismes permettant de garantir l’identité de l’expéditeur de toute requête ont été soumis à l’IETF : “Identity” [3] et une application de “Security Assertion Markup Language” (SAML) [4] pour lutter contre le SPAM [5].

“Identity” se base sur un système de certificats, et sur la présence d’un serveur d’authentification chargé de faire le lien entre l’identité soumise par un utilisateur lors de son REGISTER et celle qu’il utilise dans le champ “From” des messages qu’il expédie. “Identity” fonctionne donc autour de l’idée suivante : ”D’un point de vue des autorisations, si vous prouvez que vous pouvez légitimement vous enregistrer (REGISTER) dans un certain domaine avec une certaine “Address of Record” (AoR), alors vous avez prouvé que vous pouvez légitimement recevoir les messages adressés à cette AoR, et donc que vous pouvez mettre cette AoR dans le champ “From” des messages que vous envoyez”.

SAML, un langage basé sur “eXtensible Markup Language” (XML), définit un protocole pour échanger des informations liées à la sécurité, et notamment la date d’obtention du certificat de l’autorité certifiante, la méthode d’authentification, ... En véhiculant de telles informations entre l’émetteur et le récepteur d’une requête, le récepteur peut non seulement obtenir la garantie que l’émetteur a bien été authentifié, mais il peut également savoir de quelle manière. On peut donc imaginer, chez le récepteur, un système lui évitant de recevoir des messages signés par une autorité certifiante jugée peu sérieuse, ou trop jeune, ou lui permettant de rejeter les messages dont l’émetteur a été authentifié avec une méthode d’authentification trop faible.

SAML est donc un mécanisme plus fort que “Identity”, puisque davantage d’informations de sécurité peuvent être véhiculées et qu’elles peuvent permettre à un récepteur d’adopter un comportement plus adapté à la réception de certaines requêtes. SAML est étudié plus en détails dans la section 2.2.7.

Dans l’IMS, on dispose d’un mécanisme analogue à “Identity”, la “P-Asserted-Identity”. Lorsque le premier *proxy* (le P-CSCF) intercepte un message provenant d’un “User Agent” (UA), ce *proxy* peut vérifier si l’UA a bien peuplé son champ “From” de manière adéquate, et peut le cas échéant ajouter un “P-Asserted-Identity Header” garantissant aux autres intermédiaires que le premier *proxy* responsable du domaine de l’expéditeur a validé l’identité de cet expéditeur. On note tout de même quelques grandes différences par rapport à “Identity” :

- La technique de “P-Asserted-Identity” ne se base que sur l’ajout d’un *header*, et rien ne garantit l’intégrité du message entre ce premier *proxy* et l’UAS. Une connexion “Transport Layer Security” (TLS) entre chaque intermédiaire est donc nécessaire pour garantir la non-altération du message sur tout le reste de la route du message. Dans [3], les champs critiques sont passés dans une fonction de hachage, ce qui garantit leur intégrité.
- Les autres *proxy* ainsi que l’UAS doivent avoir une relation de confiance avec le *proxy* qui a effectué l’ajout du “P-Asserted-Identity Header”. Un domaine peu scrupuleux pourrait très bien garantir des identités alors qu’elles sont erronées, ou garantir une fausse identité par erreur. Dans [3], on se base sur des certificats, ce qui garantit la sécurité de bout en bout et ce qui empêche des serveurs moins sérieux d’ajouter le “P-Asserted-Identity Header” sur n’importe quel message.

Avec la “P-Asserted-Identity”, on a donc affaire à un mécanisme moins sûr, mais qui dispose d’un avantage majeur : il est déjà standardisé, ce qui garantit l’interopérabilité de ce service entre UA. Il conviendra peut-être de commencer par adopter cette solution, en se préparant à utiliser “Identity” une fois sa standardisation effectuée.

2.1.3 Limitations des techniques anti-SPAM provenant du courrier électronique

Il a été mentionné dans les sections précédentes que certaines techniques proposées pour lutter contre le SPAM en SIP provenaient du monde du courrier électronique. Avant de détailler ces techniques, il convient d’identifier quelles sont leurs limitations lorsqu’elles sont utilisées en dehors de leur contexte.

Premièrement, alors que le courrier électronique est un “moyen de communiquer”, SIP est un “moyen d’entrer en communication”. Ceci a une conséquence qui peut paraître logique, mais qu’il est important de souligner : dans un courrier électronique, le contenu est “directement” véhiculé. En d’autres termes, dans un courrier électronique, la “requête” correspondant à l’envoi du courrier et son contenu sont indissociables. SIP, par contre, permet de véhiculer des requêtes qui précèdent l’établissement d’une communication. Il est donc impossible de deviner le contenu qui sera échangé avant que la communication soit établie : les seuls indices permettant au récepteur d’un INVITE (par exemple) de deviner la nature de la communication sont les *headers* présents dans cette requête INVITE. Pour cette raison, les techniques basées sur l’analyse du contenu utilisées dans le courrier électronique ne sont pas utilisables en SIP.

Deuxièmement, la nature du contenu véhiculé par un courrier électronique est habituellement textuelle. Certes, il est possible d’attacher du son, des images ou même des applications à un courrier électronique (grâce aux contenus “Multipurpose Internet Mail Extensions” (MIME)), mais ce n’est pas pour ça que ce service a été initialement conçu. Par contre, SIP a été nativement conçu pour initier et (re)négocier des sessions “multimédia”, c’est-à-dire qui sont de nature à véhiculer du son, de la vidéo, des images… et du texte. Pour cette raison, il n’est pas possible d’appliquer à SIP des techniques anti-SPAM effectuant une analyse du texte contenu dans les messages.

On notera toutefois une exception à ces deux premières limitations : les requêtes SIP de type “MESSAGE”, qui véhiculent bien un contenu (dans le corps de la requête) en même temps que la requête elle-même. De plus, ce contenu est généralement textuel.

Enfin, on peut identifier une troisième et importante limitation. D’un point de vue de l’identification des utilisateurs, aucun mécanisme n’a été spécifié dès la naissance des services de courrier électronique pour identifier l’émetteur d’un courrier et valider la manière dont les champs “From” des messages ont été peuplés. Certains mécanismes sont maintenant proposés (“Extended Simple Mail Transfer Protocol” (eSMTP), …), mais le déploiement du courrier électronique sur la planète est tel qu’il est “trop tard pour faire machine arrière”. Par contre, alors que SIP n’a pas encore atteint le niveau de déploiement du courrier électronique (loin de là !), des techniques d’identification forte sont d’ores et déjà proposées, permettant d’espérer la mise en oeuvre d’une de ces techniques avant un déploiement planétaire de SIP. Pour cette raison, on peut raisonnablement se baser sur l’hypothèse qu’un mécanisme d’identification forte sera un jour ou l’autre disponible, cette hypothèse majeure n’étant pas valable pour le courrier électronique.

2.2 Présentation des techniques anti-SPAM et de langages de filtrage

2.2.1 *Blacklists*

Le principe des *blacklists* est de permettre à chaque utilisateur de tenir à jour une liste de personnes qu'il a identifiées comme "expéditeurs à rejeter". Ce principe est présenté dans [6]. La technique consiste donc, chez l'utilisateur, à effectuer un filtrage sur chaque requête entrante : si la requête provient d'un expéditeur qui est présent dans la liste noire, alors cette requête est rejetée. De cette manière, l'utilisateur ne reçoit que les requêtes provenant d'expéditeurs qu'il n'a pas (encore) identifiés comme "polluants".

Cette technique est encore largement utilisée dans le monde du courrier électronique, et ce malgré un gros point faible : quand des utilisateurs peu scrupuleux peuvent obtenir une infinité d'adresses, le fait d'en changer très régulièrement leur permet de continuer à polluer leurs victimes. Ces victimes ont beau ajouter les nouvelles adresses dans leur liste noire, elles continuent d'être inondées de requêtes non sollicitées provenant de nouvelles adresses.

On suggère tout de même de conserver cette technique parmi les "opportunités éventuelles", car en y adjoignant un mécanisme d'identification forte, elle pourrait être très utile dans le cadre de la lutte contre le SPAM en SIP.

2.2.2 *Whitelists*

Egalement présentées succintement dans [6], les *whitelists* ont un fonctionnement comparable au fonctionnement des *blacklists*, mais dans l'autre sens : alors que les *blacklists* contiennent une liste "d'expéditeurs à rejeter", les *whitelists* mentionnent une liste "d'expéditeurs à accepter". Un utilisateur donné ne reçoit donc que les requêtes provenant de personnes qu'il a identifiées comme "non polluantes". Sans autorisation explicite d'un utilisateur, il est donc impossible d'entrer en contact avec lui.

Le concept de *whitelist* est donc tout à fait analogue à celui du concept de "liste d'amis" largement utilisé dans le monde la messagerie instantanée. Dans ce monde, chaque utilisateur dispose d'une "liste d'amis", et lorsqu'il reçoit une requête provenant d'un utilisateur n'appartenant pas à sa liste, l'application de messagerie instantanée adopte un comportement paramétrable : accepter le message, rejeter le message, ou demander à l'utilisateur s'il souhaite ajouter ce nouvel expéditeur à sa liste d'amis.

Les *whitelists* constituent une protection optimale contre le SPAM, puisqu'un utilisateur ne sera jamais dérangé par des requêtes non sollicitées. De surcroît, la question de "l'infinité d'identités disponibles en SIP" n'est pas un problème : contrairement aux *blacklists*, ce n'est pas la création d'une nouvelle identité qui permettra à un spammeur de toucher davantage de victimes. On note toutefois une faiblesse à ce dispositif : comment entrer en contact avec une "nouvelle connaissance" ? En effet, si deux personnes ne se connaissant pas encore souhaitent établir un premier contact, aucun des deux futurs-interlocuteurs ne pourra entrer en contact avec l'autre pour demander à être ajouté à sa *whitelist* ! Ce problème a été qualifié de "problème d'introduction". Sans mécanisme pour résoudre ce problème, la technique des *whitelists* n'est malheureusement pas exploitable en SIP.

On propose donc de retenir cette technique, mais en y adjoignant une autre technique anti-SPAM pour résoudre le problème d'introduction : tests de Turing, Puzzles, ... Ces mécanismes sont présentés plus loin.

2.2.3 Les listes fédérées

Les deux techniques précédentes dégagent la possibilité d'une extension intéressante : fédérer ces listes entre plusieurs utilisateurs d'un même domaine pour permettre de mieux les protéger contre le SPAM. En effet, dans le cas des *blacklists*, il suffirait qu'un utilisateur ait mentionné l'existence d'un spammeur pour que tous les utilisateurs de ce domaine soient protégés. Dans le cas des *whitelists*, leur fédération permettrait d'établir une liste commune "d'interlocuteurs de confiance".

Le cas de "SpamCop"

Pour comprendre le fonctionnement des listes fédérées, nous allons nous pencher sur un cas particulier : celui de SpamCop [7], une société qui gère le *report* de SPAM par des utilisateurs, tâche de trouver les émetteurs de tels messages, et les place dans une *blacklist* fédérée mise à disposition des utilisateurs de ce service.

Cette *blacklist* fédérée porte le nom de SpamCop Blocking List (SCBL), et liste les adresses "Internet Protocol" (IP) qui ont été mentionnées par les utilisateurs comme source de SPAM. Cette *blacklist* se fournit manuellement (selon le feedback des utilisateurs) et automatiquement (*report* automatique, ...). La source de feedback peut être directe (provenant de l'utilisateur) ou indirecte (*reports* obtenus sur des relais ou des serveurs de mails ayant été abusés).

Pour prendre ses décisions, SpamCop a mis en place un système de moniteur. Son fonctionnement repose sur un système de points de réputation. Au fur et à mesure que SpamCop reçoit des requêtes concernant une adresse IP donnée, le moniteur se charge de calculer une estimation du nombre de SPAM envoyés par chaque adresse référencée. Ceci lui permet de donner une certaine réputation à chaque adresse IP. Les adresses provoquant peu de requêtes auprès de SpamCop gardent une bonne réputation ; celles qui sont souvent mentionnées comme source de SPAM voient leur réputation tomber en flèche...Toute la question étant de savoir à partir de quand il faut choisir de placer un expéditeur sur la *blacklist*.

Pour piéger les spammeurs, SpamCop a également imaginé un système de pièges, appelés *spamtraps*. Il consiste à créer sur le web des adresses e-mail qui ne sont absolument pas utilisées. Ces adresses ne sont même mentionnées nulle part. Il suffit alors de vérifier le courrier entrant sur ces boîtes aux lettres. Si du courrier arrive dans une de ces boîtes, il ne peut s'agir que de courrier non sollicité, donc de SPAM. Les spammeurs sont ainsi "piégés", et leurs adresses ajoutées à la *blacklist*.

SpamCop déclare également mettre un point d'honneur à minimiser les erreurs de "*reports* d'utilisateurs" et leurs conséquences dans le temps. Ils refusent de placer trop rapidement une adresse donnée dans la *blacklist*, et déclarent qu'une adresse ne recevant plus de feedback négatif se voit automatiquement sortir de la *blacklist* au bout de 48 heures, de manière à minimiser les conséquences d'une "condamnation" erronée.

Leurs règles sont les suivantes :

- On ne blackliste un domaine qu'après avoir reçu un grand nombre de *reports*, pour éviter les "condamnations" non justifiées.
- Les points de réputation sont fonction du temps, et les évaluations ont d'autant plus de valeur qu'elles sont récentes.
- On cote les adresses des machines qui ont été piégées par un *spamtrap* beaucoup plus durement que celles qui ont été mentionnées par des utilisateurs, de par la nature des *spamtraps*. Si le nombre de *spamtraps* est inférieur à 6, on multiplie ce nombre par 5. Sinon, on élève ce nombre au carré.
  Exemple : Si une IP a deux spamtraps et 3 reports manuels, sont score est de $(2 * 5) + 3 = 13$
  Si elle a 7 spamtraps et 3 reports manuels, son score est de $7^2 + 3 = 52$

- Un seul *report* ? On ne liste pas l'IP. 2 *reports* ? On la liste pour une durée maximale de 12 heures.
- Si on repère un e-mail pour lequel l'envoi du message pose un problème à un serveur SMTP, SpamCop intervient pour lister l'IP de l'émetteur du message immédiatement.

La possibilité de fédérer ces listes et de travailler à la manière de la société SpamCop est étudiée en détails dans le chapitre 5.

2.2.4 Systèmes de réputation

Les systèmes de réputation consistent à attribuer une "cotation" à chacun des expéditeurs avec lesquels un utilisateur entre en contact, cette note pouvant être revue au fur et à mesure de la réception de nouvelles requêtes.

Concrètement, chaque utilisateur dispose d'une liste d'expéditeurs auxquels sont associés une note. A la réception d'une requête, le système consulte cette liste, et trouve la note associée à l'expéditeur de cette requête (si l'expéditeur n'est pas présent dans la liste, il l'ajoute avec une note initiale paramétrable). En fonction de la note de cet expéditeur, le système peut ou non délivrer la requête à son destinataire, tout en lui permettant de revoir la note qui a été attribuée à cet expéditeur.

Grâce à cette technique, les spammeurs se voient donc obtenir une mauvaise réputation auprès des utilisateurs qu'ils dérangent. De plus, des systèmes permettant le partage d'informations sur les réputations existent, ce qui permet à un spammeur d'être plus vite identifié comme polluant, et donc de l'empêcher plus rapidement de polluer d'autres utilisateurs.

Cette technique présente toutefois plusieurs inconvénients :

1. Dans le cas du partage d'informations entre utilisateurs, une cotation abusivement négative peut ruiner la réputation d'un utilisateur "honnête".
2. Les spammeurs peuvent s'organiser en fédérations pour se donner entre eux des cotations excellentes leur permettant de jouir d'une bonne réputation, et donc de continuer à polluer leurs victimes.
3. La question de l'infinité d'identités disponibles reste problématique. Si des spammeurs ont une réputation négative, ils peuvent changer d'identité et "repartir à zéro", pour recommencer à polluer de nombreux utilisateurs.

2.2.5 Systèmes de consentement (“Consent Framework”)

Les systèmes de consentement, présentés dans [8] et dans [2], consistent à définir un nouveau comportement applicatif face à l’arrivée d’un nouveau message SIP (INVITE, SUBSCRIBE, MESSAGE, …) avant de déclencher le comportement applicatif par défaut (c’est-à-dire par exemple avant que le téléphone SIP ne sonne), pour éviter de déranger inutilement un destinataire donné. Ce mécanisme permet d’obliger l’expéditeur à demander une permission au destinataire, lequel se chargera de “signer”[1] cette permission au moment qu’il jugera opportun. Dans ses versions les plus avancées, ce *framework* propose d’utiliser un serveur jouant le rôle de “serveur de permissions”, qui se charge de “signer” lui-même des autorisations au nom du destinataire, selon des paramètres que ce dernier aura préalablement définis.

Ce sujet, très actuel, n’est toutefois encore qu’au stade de projet. Par exemple, il est dit dans [2] que les documents de permissions seront écrits en XML, pourront être exploités grâce à “XML Configuration Access Protocol” (XCAP), mais aucune “Document Type Definition” (DTD) n’a encore été définie. D’autre part, le comportement “applicatif” permettant de ne plus déranger l’utilisateur n’a pas non plus encore été défini. Cette technique sera étudiée plus largement dans le chapitre 4, où il seront notamment présentées les améliorations de cette technique que nous avons proposées.

2.2.6 Le langage SIEVE

Le langage SIEVE [9] est un langage permettant de filtrer le courrier électronique au moment de sa livraison finale : il peut être implémenté sur le serveur de courrier ou directement chez le destinataire. Une fois le système de filtre implémenté, l’utilisateur final peut rédiger des *scripts* en langage SIEVE (ou automatiser la création de tels *scripts* via une interface), qui lui permettront de paramétrer la manière dont les messages doivent lui parvenir.

Le langage SIEVE permet de filtrer les messages selon les critères suivants :
- la taille du message
- la présence d’un terme dans un *header* du message
- la valeur exacte d’un certain *header* du message
- …

Lorsqu’un critère correspond avec le message entrant, une des actions suivantes peut être déclenchée :
- “reject” : rejeter le message et signaler l’erreur à son expéditeur
- “discard” : rejeter le message sans avertir son expéditeur
- “keep” : accepter le message et le délivrer dans la boîte de réception habituelle du destinataire
- “redirect” : rediriger le message vers une autre adresse
- “fileinto” : délivrer le message dans un répertoire particulier

Chaque règle présente dans le *script* présente une syntaxe assez simple et intuitive. On citera pour exemple une règle permettant de rejeter les messages provenant de l’adresse “trudy@example.com” :

```
if header :contains ["from"] ["trudy@example.com"] { reject; }
```

[1] Le concept de signature n’a pas encore été spécifié dans la dernière version de [2]. On sait simplement qu’il s’agit d’une signature digitale qui identifiera l’émetteur de la requête

Cette solution semble très séduisante, notamment dans sa possibilité de filtrer tous les messages entrants selon des critères variés. La possibilité d'une variété d'actions disponibles face aux différentes formes de SPAM identifiées constitue également un point positif. De plus, cette technique est déjà standardisée (puisqu'il s'agit d'une RFC) à l'IETF.

Elle présente toutefois trois principaux inconvénients :
- La syntaxe des *scripts* de règles n'est pas basée sur XML. Il est donc impossible d'utiliser (comme dans les techniques de consentement) un serveur XCAP, et donc de gérer des fragments de documents. Ceci aura probablement un impact négatif sur les performances du système une fois les *scripts* devenus très volumineux. De plus, Alcatel dispose d'ores et déjà d'un serveur de Presence, offrant lui-même un serveur XCAP. Il serait donc intéressant d'exploiter cette opportunité.
- Alors que chaque technique comporte généralement une réaction face au SPAM, SIEVE permet d'adopter plusieurs types de comportements face au SPAM. Malheureusement, la liste des réactions disponibles étant prédéterminée, cette technique ne pourrait pas être étendue sans une révision de sa RFC, ce qui constitue un désavantage majeur.
- La technique proposée est spécifique au courrier électronique, et ne pourra être utilisée en SIP qu'après une étude prudente de son adaptation.

Cette technique ne sera donc pas utilisée directement dans le cadre de SIP. Certaines des idées présentées dans ce document auront toutefois été utilisées dans la spécification d'une solution anti-SPAM générique, présentée dans le chapitre 5.

2.2.7 Une extension du langage SAML contre le SPAM

Le langage SAML, qui a déjà été introduit dans la section 2.1.2, a été utilisé dans une extension [5] visant à utiliser ce langage pour lutter contre le SPAM. Cette technique propose de mettre en place une infrastructure spécifique et de faire une utilisation particulière du langage SAML. Cette section présente une vue d'ensemble de cette technique, en s'attardant davantage sur la syntaxe utilisée et particulièrement sur la section 5 : "Details of Security Information". Pour plus de détails relatifs aux infrastructures proposées, veuillez consulter [5].

Les trois objectifs poursuivis par le draft [5] sont les suivants :

1. En guise de première protection contre le SPAM, introduire un mécanisme de transmission d'informations liées à l'identité de tout émetteur de requêtes SIP.
2. Présenter un ensemble initial d'attributs de sécurité permettant au récepteur de mieux se protéger contre les appels non sollicités.
3. Identifier un moyen de transmettre ces informations de sécurité dans des messages SIP.

Les informations à propos d'un appel entrant qui peuvent être véhiculés dans les documents SAML sont les suivants :

– "IdentityStrength" caractérise la difficulté relative qu'a eu l'utilisateur à obtenir cette identité, ce qui permet au récepteur d'une requête de se fixer un "niveau de confiance" en l'identité de l'émetteur. Un utilisateur identifié chez un fournisseur de VoIP gratuit obtiendra un niveau de confiance faible. S'il provient d'un opérateur payant, et particulièrement d'un réseau "Public Switched Telephone Network" (PSTN), son niveau de confiance sera plus élevé. L' "IdentityStrength" pourra prendre les valeurs suivantes :

  0 : Inconnu

  1 : Service gratuit

  2 : Service payant (avec paiement ou facturation vérifiée)

  3 : Identité physique vérifiée / Entreprise / PSTN

  4 : L'utilisateur a dû présenter un passeport

– "CostOfCall" renseigne l'utilisateur sur le prix payé par l'émetteur pour contacter son interlocuteur. "CostOfCall" pourra valoir :

  0 : Inconnu

  1 : Gratuit

  2 : Tarif forfaitaire

  3 : Tarif "à la minute"

– "IdentityAssertion" décrit la méthode qui a été utilisée pour garantir l'identité de l'émetteur.

  0 : Violation de l'espace d'adressage détecté

  1 : Inconnu (aucune information d'identité disponible)

  2 : Identité garantie par un tiers de confiance présent à l'intérieur du domaine de l'émetteur

  3 : Identité garantie par un tiers de confiance extérieur

– "AuthenticationOfAccountOpening" permet de mentionner l'existence d'un mécanisme qui vérifie la création de nouveaux comptes sur ce domaine. Ses valeurs possibles sont :

  0 : Pas de validation des nouveaux comptes (il peut donc s'agir de machines)

  1 : Test de Turing (cfr section 2.2.9)

  2 : Carte de crédit (ou autres formes d'identification)

  3 : Passeport présenté pour vérification

– “SPITSuspect” est un score assigné par l’AS du domaine de l’émetteur, basé sur un examen de la requête, selon une grande variété possible de critères :
  – Le nombre d’appels émis par minute
  – Pourcentage d’appels initiés par un utilisateur (par opposition aux appels reçus)
  – Estimation du nombre d’appels émis vers des numéros distincts
  – Nombre d’appels émis ayant la même durée
  – Nombre d’appels émis vers des numéros présents dans une séquence de numéros valides

– “CallCenter” permet de savoir si l’appel a été émis ou non à partir d’un centre d’appels.

  0 : Inconnu

  1 : Reconnu comme un centre d’appels, mais non reconnu comme étant “de confiance”

  2 : Reconnu comme un centre d’appels de confiance

– “AssertionStrength” est un score général basé sur toutes les informations précédentes. Ceci permet de prendre des décisions plus facilement, si l’utilisateur ne souhaite pas entrer de critères riches et très spécifiques. Les valeurs possibles sont les suivantes :

  0 : Niveau de sécurité bas

  1 : Niveau de sécurité moyen

  2 : Niveau de sécurité élevé

Pour garantir la véracité de chaque valeur de champ d’une requête entrante, on introduit une autorité certifiante, jouant le rôle de “tiers de confiance”, et permettant :

– de garantir l’identité de l’émetteur de toute requête (via la valeur du *header* “From”),
– d’introduire les *headers* SAML fournissant un complément d’informations pour le récepteur,
– et de garantir la véracité de ces champs.

Les informations relatives à la sécurité ayant été introduites par un tiers de confiance, elles permettent à tout récepteur de requête d’avoir la garantie que la valeur de ces informations est correcte. Le tiers de confiance joue donc un double rôle : valider des *headers* provenant de l’émetteur, et ajouter des *headers* supplémentaires.

Comment transmettre ces informations supplémentaires dans une requête SIP ? Tout simplement en insérant les informations dans le corps de la requête, soit en mentionnant dans le corps de la requête une adresse où les informations relatives à la sécurité peuvent être téléchargées. On soulignera à ce propos tout l’intérêt d’utiliser le format XML pour stocker ces documents : ainsi, le serveur stockant ces documents peut être un serveur XCAP, avec tous les avantages que cela peut impliquer et qui seront présentés plus loin dans ce travail (cfr section 5.4.1 et draft [10]).

2.2.8 Geopriv

Idée générale/Introduction

Avant d'entrer dans le vif du sujet, il est essentiel de comprendre ce qu'est Geopriv, dans quel contexte il s'inscrit, et quelle est son utilité.

Dans le cadre des services liés à la Presence, un serveur stockant les informations de Presence a été créé ([11] et [12]). Ce serveur joue deux rôles principaux : fournir aux personnes qui le demandent les informations de Presence d'un utilisateur dans la limite des autorisations qu'il a données, et "utiliser un langage commun" à tous les interlocuteurs pour échanger ces informations.

D'autre part, pour tenter de fournir aux utilisateurs une information sur la localisation géographique de leurs interlocuteurs, un service de géolocalisation ([13]) a été proposé. Pour des raisons de respect de la vie privée, il fallait restreindre l'accès à ces informations dans la limite des autorisations données. De plus, l'adoption d'un langage commun était indispensable pour véhiculer ces informations.

On peut donc dire qu'à l'heure actuelle, les politiques de restriction d'accès aux informations de Presence et aux informations de géolocalisation d'un utilisateur ont été mises en place, et qu'elles fonctionnent bien. Ceci permet à tout utilisateur de disposer de la garantie que seules les personnes qu'il a explicitement autorisées peuvent accéder à ces informations.

Les similitudes (gestion des autorisations et utilisation d'un langage commun) entre les deux infrastructures sont évidentes. De plus, on peut aisément imaginer la création de nouveaux services qui nécessiteraient, comme dans le cas de Presence ou de la géolocalisation, de mettre en place un système de permissions et d'utiliser un format de documents commun. C'est la raison pour laquelle Geopriv a été créé.

Geopriv définit un *framework* permettant de créer et de gérer des politiques d'autorisations plus génériques, et d'accéder à des données spécifiques à une application. Geopriv est donc le résultat de la combinaison de deux *frameworks* : les politiques d'accès aux informations de Presence, et le contrôle d'accès à la localisation géographique d'un utilisateur.

La création d'un tel *framework* présente deux avantages principaux :

1. Elle permet, une fois les deux politiques combinées, d'enrichir les informations de Presence par des informations de localisation, et ceci en réutilisant des mécanismes communs (et donc sans alourdir les infrastructures de Presence).
2. Elle fournit un *framework* générique, et donc extensible. Ainsi, l'applicabilité de Geopriv va au-delà de son cadre initial "Presence + localisation" : Geopriv peut être étendu à tout autre service (et dans notre cas, par exemple à un service anti-SPAM).

Définitions

Plusieurs concepts sont utilisés dans le draft [13] et doivent donc être préalablement définis :

PT - Presentity : l'entité à propos de laquelle on demande des informations.

RM - Rule Maker : l'entité qui crée des règles restreignant l'accès aux données.

PS - Policy Server : l'entité qui a accès aux politiques d'autorisations et aux données. Dans le cas de Presence, il s'agirait du "Presence Server", et dans l'IMS, du "Policy Enforcement Point" (PEP)[2].

WR - Watcher : l'entité qui demande l'accès aux données du PT.

Politique d'autorisation : c'est le résultat de l'application d'un ensemble de règles.

Ensemble de règles : liste non ordonnée de règles.

Règle : un triplet "Conditions/Actions/Transformations".

Permission : une "action" ou une "transformation".

Trois modes de fonctionnement

Geopriv supporte trois modes de fonctionnement, en fonction du rôle joué par le PS.

1. **Le PS est serveur :** dans ce cas, il reçoit une requête de la part du WR, et lui renvoie un résultat (éventuellement filtré en fonction des autorisation accordées à ce WR) ou une erreur si le WR ne dispose d'aucune autorisation.
2. **Le PS est client :** le PS peut contacter le WR pour lui faire parvenir des données.
3. **Notification d'évènement :** ceci est un cas particulier du cas précédent. Le WR peut souscrire (SUBSCRIBE) à un évènement auprès du PS, par exemple la mise à jour d'informations d'un PT particulier. Lorsqu'un tel évènement se produit, le PS notifiera (NOTIFY) le WR de cet évènement.

Objectifs et contraintes de Geopriv

Représentation dans une table : chaque règle devra pouvoir être représentable sous la forme d'une ligne dans une base de données relationnelle. Ainsi, une implémentation efficace de ces règles pourra utiliser les techniques d'optimisation de ces bases de données, ce qui permettra d'offrir de meilleures performances.

Permissions uniquement (pas d'interdictions) : les règles ne mentionneront que des permissions, pas des interdictions. L'inverse nécessiterait d'ordonner les règles, ce qui compliquerait la définition d'un ensemble de règles.

[2]Dans un contexte IMS, où un serveur de Presence et un PEP coexisteraient, le serveur de Presence pourrait être considéré comme un "Application Server", et le P-CSCF pourrait jouer le rôle de PEP.

Permissions additives : une requête demandant l'accès à des données est confrontée à toutes les règles pour le WR ayant émis cette requête. Si plusieurs règles correspondent avec la requête, alors les permissions du WR correspondent à l'union des permissions présentes dans les règles qui correspondent avec cette requête.

Mises à jour et versionnage : lorsque le système fonctionne avec plusieurs PS, si des règles sont ajoutées sur un PS et qu'elles n'ont pas encore été propagées vers les autres, ces derniers doivent continuer à adopter l'ancien comportement tant que les nouvelles règles n'ont pas été correctement propagées. De plus, un RM doit savoir quels types de règles sont supportées par les PS.

Indépendance par rapport au protocole : les ensembles de règles doivent supporter les mécanismes de souscription pour les systèmes basés sur les évènements ("event-based") comme Presence. Toutefois, les règles doivent rester indépendantes du protocole utilisé.

Pas de fausse assurance : il vaut mieux demander à un utilisateur d'entrer plus de règles simples que de lui permettre d'entrer des règles complexes mais qui risquent d'être mal interprétées. L'exemple typique est celui d'une règle mentionnant "un jour de la semaine" ou "un jour du week-end", qui peut donner lieu à des confusions selon les utilisateurs et leur culture. L'utilisation de règles mentionnant chaque jour de la semaine est plus simple et plus précis.

Le draft [13] mentionne également des objectifs qui ne sont **pas** poursuivis par Geopriv. Il s'agit principalement des points suivants :

Pas de références externes : des attributs de règles ne peuvent pas référencer un autre ensemble de règles, une base de données, ou d'autres éléments du réseau.

Pas d'expression régulière ou de wildcards

Pas de règles "d'exception" : des règles de la forme "tout sauf …" ne sont pas supportées[3].

Principes de fonctionnement

Un ensemble de règles a été défini comme une liste non ordonnée contenant zéro, une ou plusieurs règles. L'ordre de ces règles n'importe pas. Cet ensemble de règles peut être stocké chez le PS et transporté entre le RM et le PS sous forme d'un seul document, par morceaux de documents, ou "règle par règle". Une règle consiste en trois parties : les conditions, les actions, et les transformations. Ces trois parties sont définies dans les sections suivantes, mais sont déjà introduites ici.

La partie "conditions" est un ensemble d'expressions, chacune pouvant être évaluée à "true" ou "false". Quand un WR demande à un PS des informations relatives à un PT, le PS parcourt chaque règle de l'ensemble. Pour chaque règle, elle évalue les expressions présentes dans la partie "conditions" de la règle. Si toutes les expressions sont évaluées à "true", alors la règle est applicable à la requête du WR. Généralement, chaque expression spécifie une condition basée sur des variables associées au contexte de la requête : l'identité du WR, son domaine, l'heure, ou même des données externes comme la température ou l'humeur du PT.

[3] On pourra toutefois mentionner des règles de la forme "n'importe quel utilisateur **authentifié**, sauf …" (cfr section 2.2.8)

Si la règle est applicable à la requête du WR, les actions et les transformations de la règle spécifient comment le PS est supposé manipuler cette requête. Une action typique lorsqu'un WR souhaite accéder à la Presence ou à la localisation d'un PT est "permit".

Si l'action autorise la requête à continuer son chemin, la partie "transformations" spécifie comment les informations liées au PT doivent être modifiées avant d'être présentées au WR. Quand un PS traite une requête, il prend les transformations spécifiées dans toutes les règles qui correspondent avec la requête, et en calcule l'union (cette technique est définie à la page 20). Cette union résultante constitue un "masque" définissant ce qui est exposé au WR. Notez que si le WR demande un sous-ensemble d'informations uniquement, l'information qui lui sera retournée sera l'intersection du masque avec les données réclamées par le WR.

Les règles seront encodées en XML. Il est important de souligner que le format de document utilisé par Geopriv (et défini plus loin) constitue uniquement le format utilisé pour échanger des informations entre le RM et le PS. Les représentations internes sont laissées au choix des développeurs. Les règles sont telles qu'elles peuvent être représentées sur un ligne d'une base de données relationnelles, ce qui constitue un atout, mais utiliser de telles bases de données n'est pas obligatoire. Il s'agit simplement d'une possibilité d'implémentation jugée très efficace. Notez que chaque règle disposera d'un paramètre qui l'identifie. Cet identifiant sera choisi par le RM, qui devra affubler chaque règle d'un identifiant différent[4].

L'extensibilité de Geopriv se trouve dans la possibilité de définir des conditions, des actions et des transformations qui sont spécifiques à une application donnée. Chaque extension devra donc définir son espace de nommage. Elle ne pourra toutefois pas changer le format défini dans [13].

Définitions des trois parties d'une règle

Les conditions Un accès aux données d'un PT ne peut être offert que si ce PT a créé une règle autorisant un tel accès. Pour tenter d'accéder à ces données, un WR doit envoyer une requête au PS. C'est le PS qui sera chargé de "confronter" la requête avec l'ensemble de règles pour déterminer si le WR est autorisé ou non à accéder aux informations demandées. Chaque règle de cet ensemble de règles peut mentionner une série de conditions qui doivent être rencontrées avant d'exécuter les actions et les transformations de cette règle. Geopriv spécifie uniquement quelques types de conditions : "condition d'identité", "sphere" et "validity". D'autres conditions peuvent être définies dans les extensions de Geopriv.

La "condition d'identité" permet de restreindre les champs filtrés aux champs "authentifiés" identifiant le WR. C'est la présence de l'élément "identity" qui remplit ce rôle. Si cet élément est absent, tous les champs sont évalués, qu'ils soient authentifiés ou non. La condition "identity" est "true" si au moins un de ses fils est évalué à "true" (ce qui signifie que les évaluations des enfants d'une règle sont combinés avec un OU logique).

[4]Ceci rejoint l'idée de possibilité d'implémentation de Geopriv dans une base de données relationnelles

L’élément “one” permet de vérifier l’identité authentifiée de l’utilisateur (contenue dans l’attribut “id”) d’exactement une entité ou un utilisateur. L’exemple suivant illustre une règle où l’on souhaite que l’identité authentifiée soit “alice@example.com”, “+1-212-555-1234” ou “bob@example.net”.

```
<rule id="f3g44r1">
  <conditions>
    <identity>
      <one id="alice@example.com"/>
      <one id="+1-212-555-1234" scheme="tel"/>
      <one id="bob@example.net" scheme="sip mailto xmpp"/>
    </identity>
  </conditions>
  <actions/>
  <transformations/>
</rule>
```

Il est également possible, grâce à “many”, de réclamer un filtrage selon plusieurs valeurs. On citera pour exemple une règle correspondant avec n’importe quel utilisateur authentifié :

```
<rule id="f3g44r5">
  <conditions>
    <identity>
      <many/>
    </identity>
  </conditions>
  <actions/>
  <transformations/>
</rule>
```

Cette règle est différente d’une règle qui correspondrait avec les requêtes provenant de n’importe quel utilisateur, qu’il soit authentifié ou non. Une telle règle serait plutôt de la forme :

```
<rule id="f3g44r5">
  <conditions>
    <identity/>
  </conditions>
  <actions/>
  <transformations/>
</rule>
```

Dans ce dernier cas, l’élément “identity” étant vide, aucune information liée à l’authentification d’un utilisateur n’est réclamée, contrairement à la règle précédente, qui correspondait à tous les utilisateurs, à condition qu’ils soient authentifiés.

Si un utilisateur le souhaite, il peut formuler des exceptions, c'est à dire accepter tous les utilisateurs exceptés ceux provenant d'un domaine donné, ou ceux ayant une certaine identité. Par exemple, si l'on veut rejeter le domaine "example.com" et l'utilisateur "alice@bad.example.net" :

```
<rule id="f3g44r1">
  <conditions>
    <identity>
      <many>
        <except domain="example.com"/>
        <except id="alice@bad.example.net"/>
      </many>
    </identity>
  </conditions>
  <actions/>
  <transformations/>
</rule>
```

L'élément "sphere" permet également de définir des conditions, mais il permet de définir l'état du PT ("bureau", "travail", "voyage", …) dans lequel cette règle est valable. Une condition "sphere" ne correspond donc avec une requête que si le PT est actuellement dans l'état mentionné dans l'attribut de "sphere" de cette règle. Ceci permet à un utilisateur de définir des règles à appliquer selon son état. S'il souhaite n'être dérangé que par des appels professionnels provenant de "andrew@example.com" alors qu'il est au travail, et recevoir des appels d'Alice lorsqu'il est à son domicile, un utilisateur pourra mentionner une série de règles de la forme :

```
<?xml version="1.0" encoding="UTF-8"?>
<ruleset xmlns="urn:ietf:params:xml:ns:common-policy">

<rule id="f3g44r2">
  <conditions>
    <sphere value="work"/>
    <identity>
      <one id="andrew@example.com"/>
    </identity>
  </conditions>
  <actions/>
  <transformations/>
</rule>

<rule id="f3g44r3">
  <conditions>
    <sphere value="home"/>
    <identity>
      <one id="alice@example.com"/>
    </identity>
  </conditions>
  <actions/>
  <transformations/>
</rule>

</ruleset>
```

Pour conclure avec la partie “conditions”, on citera l’élément “validity qui permet de préciser le début et la fin de validité d’une règle, grâce aux sous-éléments “from” et “until”. Un exemple d’application de ces éléments est proposé ici, et illustre qu’on souhaite rendre une règle valide pour un mois, entre le 15 août et le 15 septembre :

```
<rule id="f3g44r3">
  <conditions>
    <validity>
      <from>2003-08-15T10:20:00.000-05:00</from>
      <until>2003-09-15T10:20:00.000-05:00</until>
    </validity>
  </conditions>
  <actions/>
  <transformations/>
</rule>
```

Les actions Si les conditions peuvent être comparées à une clause “if” d’une instruction conditionnelle, les actions sont à mettre en parallèle avec “then”. Les actions déterminent quelles opérations le PS doit effectuer après avoir reçu une requête d’un WR qui correspond avec toutes les conditions de cette règle.

Les actions ne peuvent mentionner de fonctionnalités de type “deny”. Ceci provient du fait que des règles doivent mentionner des autorisations, et pas d’interdictions. Les transformations, décrites dans le point suivant, ne peuvent mentionner que des opérations modifiant la manière dont les données réclamées par le WR lui sont fournies. Les actions, par contre, spécifient tous les autres types d’opérations que le PS doit exécuter. Ces actions sont spécifiques à l’application utilisant les documents.

Les transformations Comme spécifié dans le point précédent, les transformations sont les opérations que le PS doit exécuter et qui modifient le résultat qui sera retourné au WR. Cette fonctionnalité est très utile notamment pour diminuer la granularité des informations (de localisation ou autre) fournies au WR. Comme les actions, les transformations sont spécifiques aux besoins d’une application, et ne sont pas définies par Geopriv.

Comment combiner des permissions ?

Il a été vu que lorsqu’une requête provenant d’un WR correspondait avec plusieurs règles, les permissions du WR correspondent à l’union des permissions présentes dans les règles qui correspondent avec cette requête. Cette procédure permettant d’obtenir une union de permissions n’est pas triviale, et des conflits peuvent se présenter. Cette section présente donc l’algorithme de combinaison de règles et de gestion des conflits.

Soient P un ensemble de règles
$M \subseteq P$ un sous ensemble de P, contenant les règles $r \in P$ correspondant avec la requête entrante
n le nom d'une permission, tq $n \in r \subset M$
$M(n) \subseteq M$ contenant les règles r de M qui ont une permission nommée n
$v(r, n)$ et $d(r, n)$ respectivement la valeur et le type de données, $\forall r \in M(n)$
$V(n)$ la valeur combinée de toutes les valeurs de permissions $v(r, n), r \in M(n)$

L'algorithme pour obtenir la valeur résultante $V(n)$ est le suivant :

Si $\forall r \in M(n)$, $d(r, n)$ est un booléen
Alors **Si** $\exists r \in M(n)$ tq $v(r, n) = true$
Alors $V(n) := true$
Sinon $V(n) := false$

Si $\forall r \in M(n)$, $d(r, n)$ est un entier
Alors **Si** $\forall r \in M(n)$, $v(r, n)$ est indéfini
Alors $V(n)$ est non spécifié
Sinon $V(n) := max\{v(r, n) \mid r \in M(n)\}$

Si $\forall r \in M(n)$, $d(r, n)$ est un ensemble
Alors $V(n) := \bigcup_{r \in M(n)} v(r, n)$ partout où $v(r, n)$ est défini

La combinaison de ces opérations aura pour effet d'obtenir la plus grande valeur dans le cas des entiers, l'opération "ou" dans le cas des booléens, et l'union dans le cas des ensembles.

Exemple

Cette section propose un exemple de document XML représentant une règle assez simple, mais qui illustre la plupart des mécanismes présentés précédemment. Dans cet exemple, un utilisateur souhaite, lorsqu'il est au travail, accepter de fournir ses informations à "bob@example.com" à condition que l'identité de celui-ci ait été authentifiée. La validité de cette règle porte de 17h à 19h, le 24 décembre 2006. Pour ce faire, il devra mentionner un document de la forme :

```
<?xml version="1.0" encoding="UTF-8"?>
<ruleset xmlns="urn:ietf:params:xml:ns:common-policy">

<rule id="f3g44r1">
  <conditions>
    <identity>
      <one id="bob@example.com"/>
    </identity>
    <sphere value="work"/>
    <validity>
      <from>2006-12-24T17:00:00+01:00</from>
      <until>2006-12-24T19:00:00+01:00</until>
    </validity>
  </conditions>
  <actions/>
  <transformations/>
</rule>
</ruleset>
```

Définition du format de document utilisé dans Geopriv

La définition du format de document, baptisé “Common Policy Markup Language” est disponible dans la section 13 du draft [13].

2.2.9 Tests de Turing

Les tests de Turing consistent à soumettre à l’expéditeur d’un message un “challenge”, sous la forme d’un problème aisément soluble par un être humain, mais théoriquement insoluble par une machine. Ceci permet de garantir que l’expéditeur d’un message est effectivement un être humain, ce qui écarte les problèmes de messages non sollicités envoyés automatiquement et en grande quantité par des machines. Leur utilisation est très fréquente sur Internet lors d’inscription à des forums de discussion : il s’agit des images présentant une suite de quelques caractères qu’il faut reconnaître et entrer dans une zone de texte. Cette solution est actuellement utilisée par de nombreux sites Internet, et est notamment proposée par “Captcha” [14]. Cette utilisation semble porter ses fruits pour protéger les forums de discussion du SPAM.

Cette technique a toutefois pour principal désavantage de déranger l’expéditeur d’un message à chaque fois qu’elle est utilisée, qu’il soit bien ou malintentionné. Comme elle dispose du gros avantage d’être théoriquement incontournable, elle pourrait être être utilisée ponctuellement, par exemple lors de l’introduction de nouveaux contacts dans une *whitelist*. On disposerait dès lors d’un système où les utilisateurs sont protégés du SPAM grâce à une *whitelist*, et où des nouveaux contacts peuvent être ajoutés à ces *whitelists* grâce à un test de Turing. Ceci serait une solution possible au “problème d’introduction” des *whitelists*.

Les tests de Turing possèdent un autre avantage non négligeable : ils doivent être lus et réalisés par des humains, et on ne rencontre donc pas de problème de compatibilité entre interlocuteurs.

Par contre, la nature des messages véhiculés pourrait constituer une barrière au déploiement des tests de Turing en SIP : pour les sessions multimédias, comment imaginer des tests de Turing “audio” ou “vidéo” ? Dans le cas de tests audiophoniques, la question de la phonétique entre utilisateurs ne parlant pas la même langue n’est-elle pas problématique ? Même pour des types de tests plus simples (sous forme de texte), comment résoudre les questions d’accessibilité (personnes malvoyantes, ...) et les problèmes d’encodage de caractères ou de différences entre les alphabets de certaines régions (les caractères disponibles sur les claviers pouvant différer d’un paus à l’autre) ?

2.2.10 Computational Puzzles

Actuellement, on estime que le coût d’un télémarketing “nouvelle génération”, où des robots enverraient chaque seconde des milliers d’annonces textuelles ou vocales vers des milliers de personnes, est mille fois moins cher que la moins onéreuse des techniques de télémarketing “classique”, s’appuyant sur des opérateurs humains utilisant les lignes PSTN. Pour cette raison, et si rien ne change, ces nouvelles formes de télémarketings sont appelées à exploser.

Pour tenter de contrer ce problème, il convient de tenter d’augmenter le coût de l’envoi de tels messages, sans pour autant occasionner un coût aux utilisateurs “honnêtes” des nouvelles technologies. Une des techniques visant cet objectif est la technique des “Computational Puzzles”, actuellement traitée dans [1].

Son principe consiste à augmenter le coût (en terme de temps de consommation du processeur, et donc en coût financier pour l’achat de ces derniers !) de l’envoi d’un message, afin de rendre financièrement moins intéressant l’envoi de milliers de SPAMS “à la chaîne”.

Lorsqu’un destinataire reçoit un message, il répond à l’expéditeur de ce message en lui envoyant un “challenge”. Ce challenge consiste en un puzzle, basé sur l’algorithme de hachage SHA1, que l’expéditeur va devoir résoudre pour pouvoir entrer en communication avec son interlocuteur. L’établissement d’une communication va donc lui coûter du temps de calcul, et le rendement des robots envoyant des milliers de messages chaque minute s’en verrait donc fortement diminué.

Cet algorithme permet de régler dynamiquement un “niveau de difficulté” des puzzles envoyés, permettant d’adapter cette difficulté à la puissance de calcul de l’expéditeur.

Cette technique, jugée très efficace, comporte toutefois deux inconvénients majeurs :

- Elle serait tout à fait applicable aux hôtes fixes, mais serait coûteuse en “Central Processing Unit” (CPU) et donc en batterie sur les hôtes mobiles. Il ne faudrait donc appliquer cette solution que ponctuellement... Par exemple pour “le problème d’introduction” dans une *whitelist*.
- Il faudrait s’assurer que l’expéditeur ne peut pas “tricher” sur sa puissance de calcul, se faisant passer pour un petit hôte mobile alors qu’il dispose d’un cluster surpuissant.
  Ce point pourrait constituer un inconvénient critique si la puissance de calcul de chaque UA ne peut être déterminée avec certitude.

Un troisième inconvénient pourrait poser problème dans la mise en oeuvre de cette solution : elle n'est pas encore standardisée, et oblige à une compatibilité entre UA. Comme dans le "Consent Framework", si un des deux interlocuteurs ne supporte pas cette technique, il interprétera le challenge reçu précédemment comme un message d'erreur, ce qui empêchera la communication.

Cette technique est étudiée en détail dans le chapitre 3.

2.2.11 Communications payantes et remboursables

Ce système dont l'idée repose sur la technique "payments at risk" de [6], consiste à faire payer une somme modique lors de l'envoi de tout message, cette somme pouvant être remboursée si le destinataire juge que le message reçu n'est pas du SPAM.

Concrètement, lorsqu'un expéditeur souhaite envoyer un message, il doit tout d'abord garantir qu'il a effectué, lors son envoi, un paiement auprès d'un organisme de micro-paiement (ou auprès de son opérateur). Ce paiement pourrait être garanti via l'obtention d'un certificat. Lorsque le destinataire reçoit le message, il peut s'il estime ne pas avoir affaire à du SPAM autoriser le remboursement de son interlocuteur.

L'idée est assez proche du concept des Computational Puzzles, si ce n'est que le coût de l'envoi de messages n'est plus un coût en temps, mais un coût financier. Cette technique comporte toutefois de gros inconvénients qui risquent d'empêcher sa mise en oeuvre :

- Les sociétés de micro-paiements demandent toujours une rémunération pour leur travail. L'expéditeur "honnête" ne recevrait donc pas, lors de son remboursement, l'intégralité de la somme initialement versée. Il faudrait donc imaginer cette technique comme un service payant (mais à prix modique) auquel les personnes intéressées pourraient souscrire.
- Pour éviter cet intermédiaire payant que sont les sociétés de micro-paiements, on pourrait faire supporter ce service directement au niveau des opérateurs. Il faut noter qu'inclure un nouveau service lié à la facturation chez les opérateurs peut donner lieu à de longues procédures (cette activité constituant un point critique chez les opérateurs), mais que cette solution n'est dans l'absolu pas irréaliste.
- Il n'est pas à exclure que certains émetteurs de SPAM soient prêts à "investir" dans l'envoi de messages publicitaires payant. En effet, plutôt que d'être découragés à l'idée d'être facturés pour l'envoi de tels messages, ils pourraient choisir de payer pour obtenir la garantie que leur publicité parviendra bel et bien à leur destinataire, ce qui n'était pas le cas précédemment[5].

[5] Un tel cas de figure s'estd éjà présenté, comme relaté dans [15]

2.3 Les configurations intéressantes

Pour tenter des dégager les opportunités de chacune de ces techniques, la question préliminaire principale reste manifestement de savoir si une infinité d'identités sera ou non disponible en SIP. Comme aucune réponse n'a encore été apportée à cette question, on peut distinguer les solutions intéressantes dans chacun des deux cas de figure.

S'il s'avère effectivement possible d'obtenir une infinité d'URI en SIP (comme c'est actuellement le cas avec les adresses de courrier électronique), la meilleure solution consiste probablement à s'appuyer sur une *whitelist*, ce qui constituerait une solution optimale contre le SPAM. Pour résoudre "le problème d'introduction", on y adjoindrait les "Computational Puzzles" ou les "tests de Turing".

Si le nombre d'URI disponible s'avère limité, un système de *blacklist* pour s'avérer assez efficace, tout en restant libre d'y adjoindre des protections supplémentaires, comme le "Consent Framework".

Dans les deux cas de figure, plusieurs préoccupations devront rester au centre de notre attention :
- La question de l'identité des émetteurs de requêtes, et particulièrement les opportunités dégagées par "Identity" ([3]) et SAML([5]).
- La possibilité de fédérer les listes pour "unir les utilisateurs d'un domaine contre le SPAM".
- Le niveau de standardisation de chacune des techniques présentées, pour garantir l'interopérabilité entre les UA.

Chapitre 3

Les Computational Puzzles

Après avoir effectué un tour d'horizon des différentes techniques existantes et en avoir présenté les principales forces et faiblesses, la première technique à avoir été plus profondément étudiée fut la technique des "Computational Puzzles". En effet, cette technique (présentée dans le draft [1]) semblait une solution particulièrement intéressante pour lutter contre le SPAM. De plus, elle faisait partie des techniques proposées à l'IETF, et rejoignait donc notre souci de proposer des solutions "standardisées" contre le SPAM.

Après en avoir étudié le fonctionnement, cette technique fut spécifiée puis implémentée. Ensuite, après avoir pris contact avec l'auteur du draft pour lui présenter notre prototype, des exemples de valeurs pour sa technique ainsi qu'un exemple complet lui furent proposés et intégrés dans la dernière version du draft.

3.1 Introduction

La technique des "Computational Puzzles" est une technique visant à lutter contre le SPAM, mais d'une façon un peu particulière. Alors que les autres techniques cherchent à empêcher un spammeur d'expédier son message, à un *proxy* de relayer ce message non désiré, ou à une victime de le recevoir, la technique dite des "Computational Puzzles" agit à un autre niveau. Elle consiste à augmenter le coût (en terme de temps de consommation du/des processeur(s), et donc en terme de coût financier pour l'achat de ce(s) dernier(s)) de l'envoi de messages, afin de rendre financièrement inintéressant l'envoi de milliers de messages "à la chaîne".

En d'autres termes, l'idée de cette technique est la suivante : s'il n'est pas gênant, pour un utilisateur "honnête", de subir un délai de quelques secondes entre le moment où il clique sur le bouton "Envoyer" et le moment où son message est effectivement transmis au serveur, ce délai est extrêmement handicapant pour des utilisateurs malintentionnés tentant d'expédier de façon automatisée une grande quantité de messages à chaque intervalle de temps.

3.2 Idée générale

Lorsqu'un "User Agent Client" (UAC) souhaite entrer en communication avec un "User Agent Server" (UAS), il commence traditionnellement par lui envoyer une requête donnée (par exemple un INVITE). Lors de la réception de ce message l'UAS va demander à l'UAC de résoudre un puzzle, et de lui refaire parvenir son message initial en l'accompagnant de la réponse au puzzle demandé. La résolution de ce puzzle lui aura demandé un certain temps, ce qui permettra à l'UAS de ne plus avoir de doutes sur le sérieux de l'expéditeur du message. Un schéma général des messages échangés est proposé dans la figure 3.1.

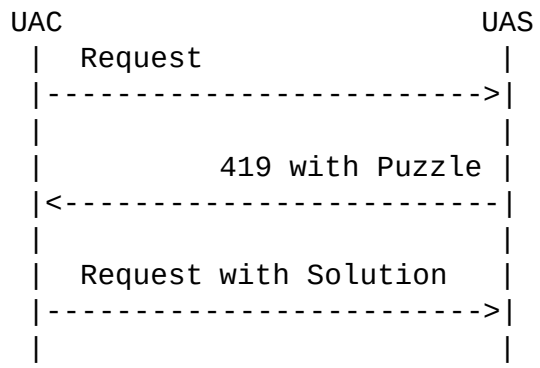


FIG. 3.1 – Schéma général du "Computational Puzzle" (source : [1])

3.3 Définition du concept de "Puzzle"

Quelle est la définition formelle du concept de "Puzzle" ? Un puzzle consiste en quatre valeurs :

1. Un nombre entier, nommé "value" .
2. Un nombre entier, nommé "work" .
3. Une chaîne de caractères, nommée "preImage" .
4. Une chaîne de caractères, nommée "image" .

Il doit exister une chaîne de caractère X (qui est la solution du puzzle) telle que
- X et "preImage" sont identiques, à l'exception de leur "work" bits de poids faible.
- **Soit** Y = hachage de la concaténation de "z9hG4bK" et de X, avec l'algorithme de hachage SHA-1 **Alors** les "value" bits de poids faible de Y correspondent à ceux de "image" .

Le draft [1] propose une version plus mathématique :

Soit

- `low(v,x)` = méthode retournant les v bits de poids faibles de la chaîne de caractères x.
- `zero(v,x)` = méthode retournant la chaîne de caractères x dont les v bits de poids faible ont été mis à 0.
- `SHA1(x)` = méthode retournant le hachage de la chaîne de caractères x, avec l'algorithme de hachage SHA1, défini dans [16].
- `+` = l'opérateur de concaténation de chaînes de caractères.

Alors $\exists$ X tq

- `low(“value” ,“image” ) = low(“value” ,SHA1("z9hG4bK" + X))`
- `zero(“work” ,X) = zero(“work” ,“preImage” )`

On notera une chose intéressante : c'est l'ajout de la seconde contrainte qui permet de régler le niveau de difficulté du puzzle fourni. En effet, sans la deuxième contrainte, le problème posé reviendrait à demander à l'UAC de casser l'algorithme SHA1, ce qui rendrait le système inutilisable. On demanderait à l'UAC de partir d'une chaîne initiale correspondant à une chaîne de "value" bits NULLE, et de boucler en incrémentant cette chaîne jusqu'à ce qu'il trouve la valeur de "preImage" que l'UAS a générée aléatoirement pour former le puzzle.

L'autre extrême consisterait à lui fournir la "preImage" : dans ce cas, la résolution du puzzle serait instantanée.

L'idée est alors de trouver un juste milieu, en demandant de ne résoudre qu'une partie du problème. Ainsi, au lieu de faire démarrer la recherche d'une chaîne de bits **nulle** (difficulté maximale), ou de démarrer à partir de la "preImage" (difficulté minimale), on permet de démarrer à la valeur de la "preImage" ... dans laquelle on a mis un certain nombre de bits à zéro, ce nombre étant la valeur de "work" .

Ainsi, fournir un puzzle avec des petites valeurs de "work" permettra de générer un puzzle facile à résoudre. Utiliser de grandes valeurs de "work" grandes (c'est-à-dire "plus proches"[1] de 160) donnera un puzzle difficile.

Ce point est rendu beaucoup plus clair par l'algorithme qui se cache derrière ce concept. En fait, pour trouver une solution au puzzle, le nombre maximal de boucles nécessaires à l'UAC sera de 2^{work}. Ceci justifie qu'une valeur de "work" élevée fournira un puzzle de difficulté élevée, puisque le nombre d'itérations nécessaires risquera d'être plus grand, la seule méthode possible pour découvrir la solution d'un puzzle étant une recherche exhaustive.

[1] En pratique, des valeurs de "work" supérieures à 20 donneront souvent des problèmes suffisamment lourds pour les puissances de processeurs disponibles actuellement.

3.4 Comportements

3.4.1 Comportement de l'émetteur d'un puzzle

Pour construire et envoyer un puzzle, l'UAS pourra adopter la démarche suivante :

1. Stocker dans "preImage" la valeur d'un String généré aléatoirement, et haché avec l'algorithme SHA1.
   `preImage := SHA1(String aléatoire)`
2. Stocker dans "image" le hachage de la concaténation de "z9hG4bK" et de "preImage" .
   `image := SHA1("z9hG4bK" | preImage)`
3. La valeur de "value" sera arbitrairement fixée à 160, cette valeur correspondant à la longueur fixe de l'output de l'algorithme de hachage SHA1[2] (en bits).
4. La valeur de "work" , qui va déterminer la difficulté du puzzle, pourra être réglée dynamiquement selon la puissance de calcul de l'interlocuteur.
5. Sauvegarder la valeur de la "preImage", qui constitue la solution que son interlocuteur devra trouver.
6. Mettre les "work" bits de poids faible de "preImage" à zéro.
   `preImage := zero(work,preImage)`
7. Créer un "Puzzle Header Field", comme spécifié dans [1]. Attention : conformément à ce draft, les valeurs des chaînes de caractères "preImage" et "image" devront être encodées en base-64.
8. Répondre à la requête ayant provoqué la création d'un puzzle par une réponse "419 Puzzle Required", dont le champ "Puzzle Header" a été initialisé à la valeur obtenue dans l'étape précédente.

3.4.2 Comportement du récepteur du puzzle

Pour résoudre un puzzle et envoyer sa réponse, l'UAC devra effectuer les étapes suivantes :

1. Consulter les valeurs de "value" et de "work" , et décider ou non de résoudre le puzzle fourni.
2. Si l'UAC décide de résoudre ce puzzle, effectuer les étapes suivantes :
   (a) Vérifier que les "work" bits de poids faible de "preImage" sont à zéro. Si ce n'est pas le cas, le puzzle doit être considéré comme mauvais, et le "419 Puzzle Required" doit être considéré comme un message d'erreur.
   (b) Créer une variable Y initialisée à la valeur des "value" bits de poids faible de "image" .
   `Y := low("value" , "image" )`
   (c) Créer une autre variable X qui contiendra la solution, et l'initialiser à la valeur de la "preImage" dont les "work" bits de poids faible ont été mis à zéro (bien que cela ait déjà été fait par l'émetteur du puzzle).
   `X := zero("work" , "preImage" )`
   (d) Commencer à itérer... Tant que `low( value, SHA1("z9hG4bK" | X ))` n'est pas égal à Y, il faut "incrémenter" X (effectuer une addition binaire entre la valeur binaire de X et 1) [3]

[2] Notez qu'une proposition de généraliser l'algorithme de hachage a été faite, la seule contrainte étant d'utiliser un algorithme fournissant des résultats de longueur fixe. "value" devra obligatoirement valoir la longueur de cet output.

[3] Notez que, comme cela a été dit précédemment , le nombre maximal d'itérations sera 2^{work}

(e) Si, au bout de 2^{work} itérations, aucun résultat n'a été trouvé, alors le puzzle doit être considéré comme mauvais, et le "419 Puzzle Required" doit être considéré comme un message d'erreur.

(f) Sinon, une solution a été trouvée. Il faut alors reformuler la requête initiale (généralement INVITE) en y insérant un puzzle *header* field identique à celui reçu dans le "419 Puzzle Required", à l'exception de la valeur "work" qui doit être mise à zéro et de la "preImage" qui doit contenir la solution X du puzzle.

3.4.3 Vérifier la validité du puzzle

Quand l'UAS reçoit la nouvelle requête de son interlocuteur, si celle-ci contient un "Puzzle Header Field", il consultera la valeur du champ "preImage" présent dans ce *header*, et vérifiera également que la valeur de "work" est à 0. Dans ce cas, il comparera la valeur de la "preImage" qu'il avait obtenue pour former son puzzle à la "preImage" fournie par l'UAC et qu'il prétend être la solution de puzzle.

Si ces valeurs sont identiques, alors l'UAC a bien réussi le challenge : l'UAS peut donc répondre favorablement à la requête de son correspondant.

3.4.4 Exemple

En guise d'exemple, nous présentons un exemple d'établissement de communication entre deux utilisateurs, Alice et Bob. Bob utilise les "SIP Computational Puzzles against SPAM".

Tout d'abord, Alice envoie un message INVITE à Bob. Bob, qui veut être sûr qu'Alice n'est pas un spammeur, répond à Alice avec un message "419 Puzzle Required", indiquant qu'il exige qu'Alice réussisse un challenge avant que la communication soit effectivement établie. La réponse de Bob contient un "Puzzle Header Field", qu'il a obtenu en suivant la procédure décrite précédemment.

Si nous supposons que Bob a choisi de fixer la valeur de "work" à 15, et qu'il a généré aléatoirement le string :

```
random string = "itjjyfdubtpneggrdsaavouy"
```

Alors Bob obtient les valeurs suivantes :

```
"preImage" de départ = SHA1(random string)
= SHA1("itjjyfdubtpneggrdsaavouy")
= "VgVGYixbRg0mdSwTY3YIfCBuYmg=" (encodé en base-64)

= "01010110 00000101 01000110 01100010 00101100
   01011011 01000110 00001101 00100110 01110101
   00101100 00010011 01100011 01110110 00001000
   01111100 00100000 01101110 01100010 01101000" (encodé en binaire)
```

```
“preImage” envoyée
= zero(work, preImage) où work = 15 (dans notre cas)
= "VgVGYixbRg0mdSwTY3YIfCBuAAA=" (encodé en base-64)

= "01010110 00000101 01000110 01100010 00101100
   01011011 01000110 00001101 00100110 01110101
   00101100 00010011 01100011 01110110 00001000
   01111100 00100000 01101110 00000000 00000000" (encodé en binaire)

“image”
= SHA1 ("z9hG4bK" | “preImage” de départ)
= "NhhMQ2l7SE0VBmZFKksUC19ia04=" (encodée en base-64)
```

Le “Puzzle Header Field” du “419 Puzzle Required” envoyé à Alice aura donc la forme :

```
Puzzle: work=15; pre="VgVGYixbRg0mdSwTY3YIfCBuAAA=";
        image="NhhMQ2l7SE0VBmZFKksUC19ia04="; value=160
```

Alice recevra la réponse “419 Puzzle Required”, et le point de départ de sa recherche sera une valeur X initialisée à la valeur de la preImage dont les work bits de poids faibles ont été mis à zéro (sans effet si le puzzle était valide). La valeur de X avant de commencer la recherche d’une solution est donc :

```
"01010110 00000101 01000110 01100010 00101100
 01011011 01000110 00001101 00100110 01110101
 00101100 00010011 01100011 01110110 00001000
 01111100 00100000 01101110 00000000 00000000" (encodé en binaire)
```

Pendant sa boucle de recherche d’une solution, si la valeur binaire de X à une itération donnée était :

```
"01010110 00000101 01000110 01100010 00101100
 01011011 01000110 00001101 00100110 01110101
 00101100 00010011 01100011 01110110 00001000
 01111100 00100000 01101110 00010010 10010111"
```

Alice devrait incrémenter cette valeur de 1, c’est-à-dire effectuer une addition binaire entre la valeur binaire de X et 1, et devrait donc obtenir comme nouvelle valeur :

```
"01010110 00000101 01000110 01100010 00101100
 01011011 01000110 00001101 00100110 01110101
 00101100 00010011 01100011 01110110 00001000
 01111100 00100000 01101110 00010010 10011000"
```

Finalement, lorsqu’elle a trouvé la solution, la requête qu’elle envoie est une copie de sa requête initiale (INVITE), dans laquelle elle insère un “Puzzle Header Field” correspondant à celui qu’elle a reçu dans le message “419 Puzzle Required”, à l’exception du champ “work” qu’elle met à 0, et du champ “preImage” dans lequel elle place la solution. Dans son cas, le “Puzzle Header Field” de sa nouvelle requête est le suivant :

```
Puzzle: work=0; pre="VgVGYixbRg0mdSwTY3YIfCBuYmg=";
        image="NhhMQ2l7SE0VBmZFKksUC19ia04="; value=160
```

Pour conclure cet exemple, on propose dans la figure 3.2 un résumé des messages échangés entre Alice et Bob, et la valeur du “Puzzle Header Field” pour les messages ayant utilisé ce *header*.

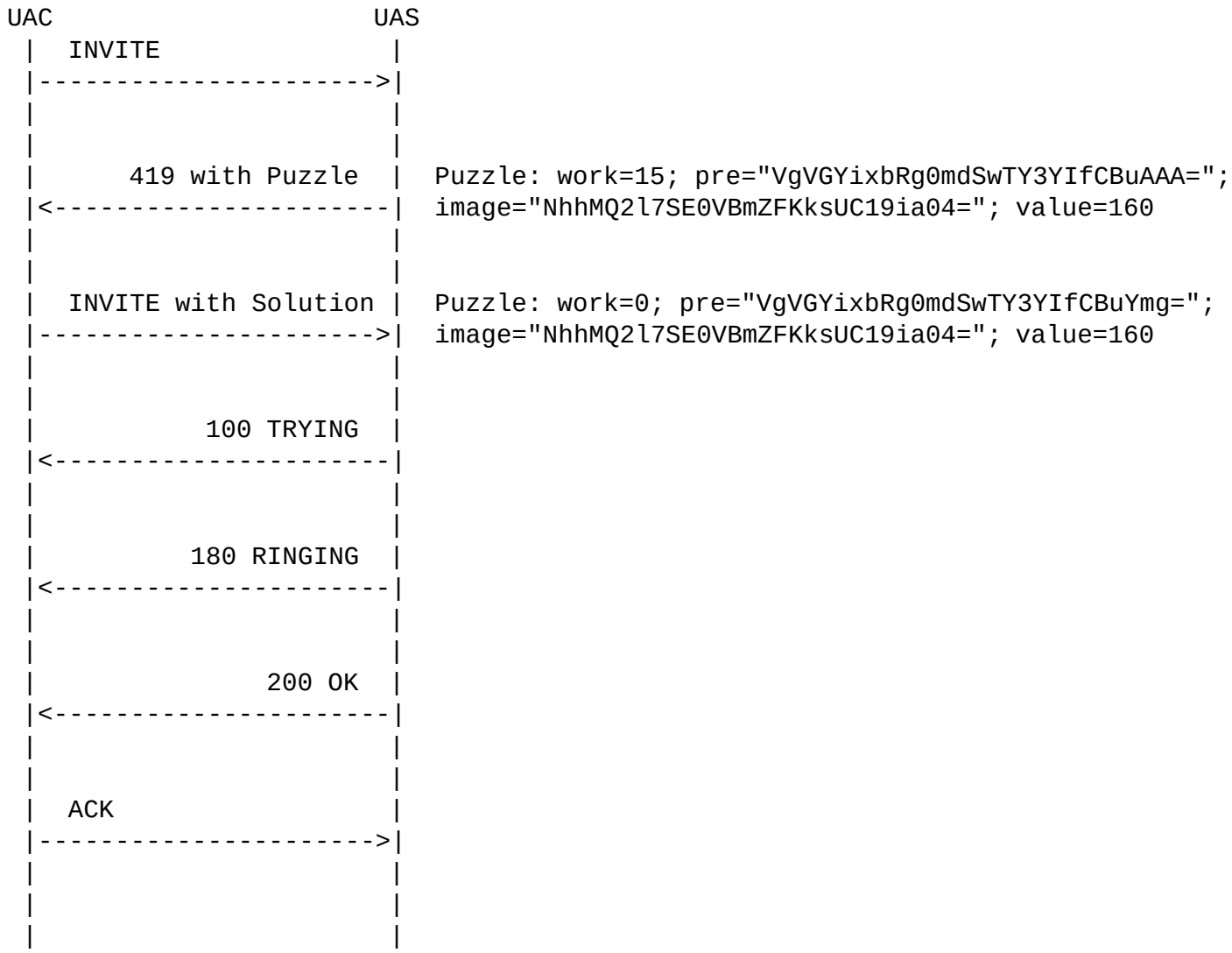


FIG. 3.2 – Exemple de callflow

3.5 Cas d'utilisation de cette technique

Lors du stage, la technique des "Computational Puzzles" a été spécifiée et implémentée. S'il est vrai que les sections précédentes (fonctionnalités attendues, comportement des acteurs, ...) constituent une part importante de la spécification, celle-ci n'aurait pas été complète sans une étude des cas d'utilisation du prototype. Ces cas d'utilisation sont présentés dans cette section.

3.5.1 Use Case Diagram

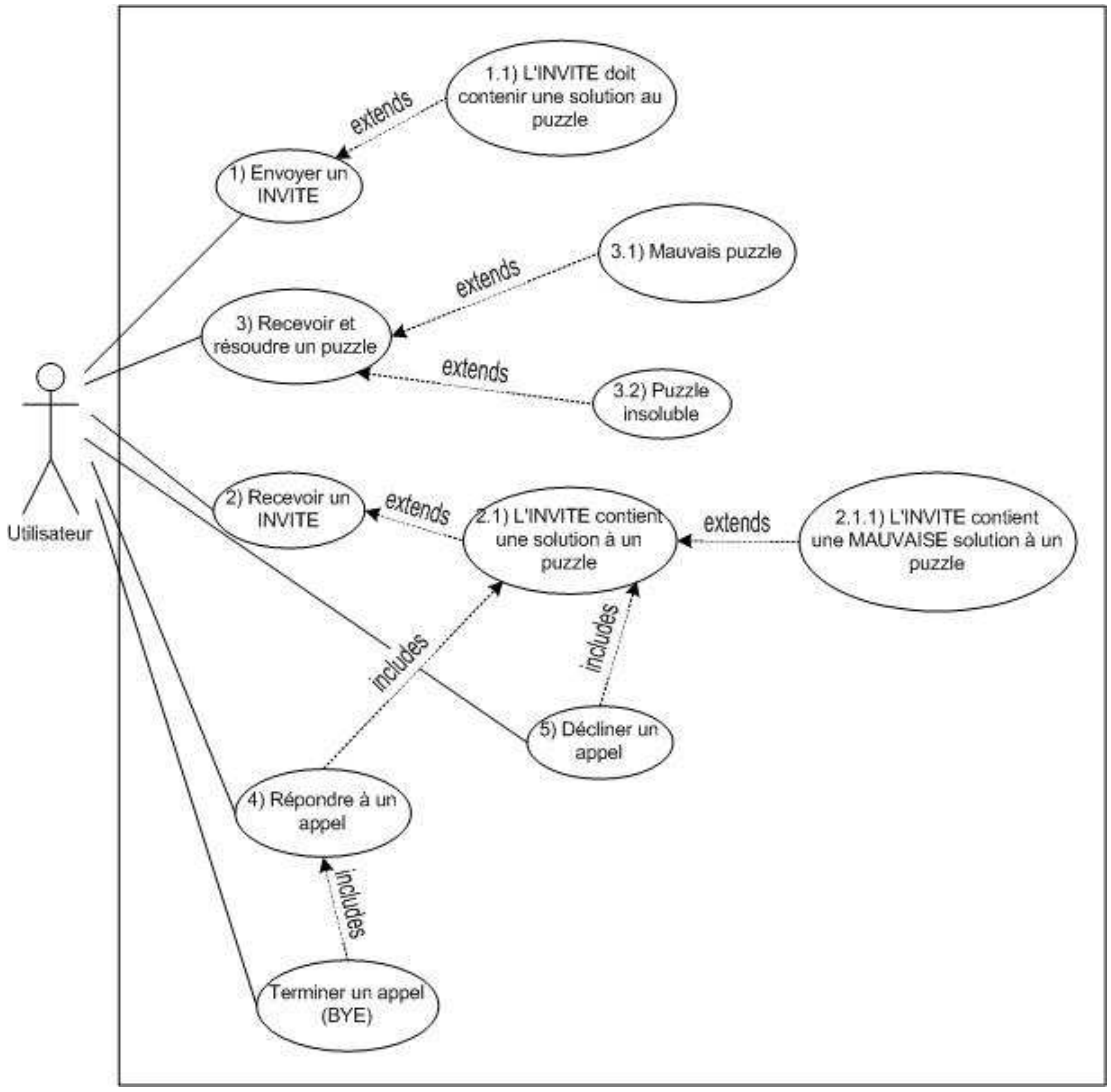


FIG. 3.3 – Use Case Diagram des Computational Puzzles

3.5.2 Scénarios

Use Case 1 : Envoyer un INVITE

UC (1) : Envoyer un INVITE **Description :** L'UAC invite un UAS à établir une session **Précondition :** / **Postcondition :** Le message INVITE a été envoyé à l'UAS

UAC	UAS
1. L'UAC prépare un message INVITE 2. L'UAC envoie cet INVITE à l'AoR de l'UAS	

Cas alternatif 1 : L'INVITE doit contenir une solution au Puzzle

UC (1.1) : L'INVITE doit contenir une solution au Puzzle **Description :** L'UAC invite un UAS à établir une session, après avoir résolu le puzzle qui lui avait été soumis **Précondition :** L'UAC a reçu un message "419 Puzzle Required" contenant un puzzle à résoudre **Postcondition :** Le message INVITE a été envoyé à l'UAS

UAC	UAS
«le début du scénario est identique au cas normal»	
2. L'UAC y ajoute un "Puzzle Header" contenant la solution au puzzle qui lui avait été soumis	
«retour au point 2 du cas normal»	

Use Case 2 : Recevoir un INVITE

UC (2) : Recevoir un INVITE **Description :** Un UAS reçoit un appel **Précondition :** / **Postcondition :** Un puzzle a été envoyé à l'UAC

UAC	UAS
	1. L'UAS reçoit un message INVITE ne contenant pas de "Puzzle Header" 2. L'UAS prépare un puzzle à faire parvenir à l'UAC 2. L'UAS répond à l'INVITE avec un message "419 Puzzle Required" contenant un Puzzle à résoudre

Cas Alternatif 2.1 : L'INVITE contient une solution à un puzzle

UC (2.1) : L'INVITE contient une solution à un puzzle **Description :** Un UAS reçoit un INVITE accompagné d'une réponse à un puzzle **Précondition :** / **Postcondition :** L'UAS peut répondre favorablement à l'appel

UAC	UAS
	1. L'UAS reçoit un message INVITE contenant une solution à un puzzle 2. L'UAS vérifie la réponse au puzzle soumis et celle qui lui est fournie concordent 3. Si les réponses concordent, le téléphone sonne !

Cas Alternatif 2.1.1 : L'INVITE contient une MAUVAISE solution à un puzzle

UC (2.1.1) : L'INVITE contient une MAUVAISE solution à un puzzle **Description :** Un UAS reçoit un INVITE accompagné d'une réponse à un puzzle, mais cette réponse est incorrecte **Précondition :** / **Postcondition :** L'UAS peut répondre défavorablement à l'appel

UAC	UAS
«le début du scénario est identique au cas normal»	
	3. Les réponses ne concordant pas, l'UAS répond à l'INVITE par un "406 Not Acceptable"

Use Case 3 : Recevoir et résoudre un puzzle

UC (3) : Recevoir et résoudre un puzzle **Description :** L'UAC reçoit message "419 Puzzle Required" contenant un puzzle **Précondition :** / **Postcondition :** L'UAC dispose de la solution du puzzle

UAC	UAS
1. L'UAC reçoit un message "419 Puzzle Required" contenant un puzzle à résoudre 2. L'UAC Vérifie que les "work" bits de poids faible sont bien mis à 0 3. L'UAC résout le puzzle en appliquant l'algorithme de résolution 4. L'UAC dispose de la solution du puzzle	

Cas alternatif 3.1 : Mauvais puzzle (les "work" bits de poids faible ne sont pas à 0)

UC (3.1) : Mauvais puzzle (les "work" bits de poids faible ne sont pas à 0) **Description :** Le puzzle reçu n'est pas valide car ses "work" bits de poids faible ne sont pas à 0 **Précondition :** Un puzzle a été reçu **Postcondition :** Le message "419" a été traité comme un message d'erreur

UAC	UAS
«le début du scénario est identique au cas normal»	
3. L'UAC considère le "419 Puzzle Required" comme un message d'erreur	

Cas alternatif 3.2 : Puzzle insoluble

UC (3.1) : Puzzle insoluble **Description :** Le puzzle reçu n'est pas valide : aucune solution ne peut être trouvée **Précondition :** Un puzzle a été reçu **Postcondition :** Le message "419" a été traité comme un message d'erreur

UAC	UAS
«le début du scénario est identique au cas normal»	
4. L'UAC considère le "419 Puzzle Required" comme un message d'erreur	

Use Case 4 : Répondre à un appel

UC (5) : Répondre à un appel **Description :** L'UAS répond à l'appel de l'UAC **Précondition :** / **Postcondition :** La communication est établie	
UAC	**UAS**
«include UC (2.1) : Recevoir un INVITE contenant une solution à un puzzle»	
	4. L'UAS décroche 5. L'UAS envoie un "200 OK" à l'UAC

Use Case 5 : Décliner un appel

UC (6) : Décliner un appel **Description :** L'UAS ne souhaite pas répondre à l'appel de l'UAC **Précondition :** / **Postcondition :** Le téléphone ne sonne plus	
UAC	**UAS**
«include UC (2.1) : Recevoir un INVITE contenant une solution à un puzzle»	
	4. L'UAS signale qu'il ne souhaite pas répondre à cet appel 4. L'UAS envoie un message "603 Decline" à l'UAC

Use Case 6 : Terminer un appel (BYE)

UC (7) : Terminer un appel (BYE) **Description :** Un UA met fin à la communication en cours **Précondition :** Une communication a été établie **Postcondition :** La communication est terminée	
UAC	**UAS**
«include UC (4) : Répondre à un appel»	
	1. L'UAS envoie un message "BYE" à son interlocuteur
2. L'UAC confirme cette demande en répondant par un "200 OK"	

Pentium II		Pentium IV		Pentium XEON	
Niveau	Temps	Niveau	Temps	Niveau	Temps
1	12 ms	1	4 ms	1	3 ms
2	9 ms	2	5 ms	2	1 ms
3	13 ms	3	8 ms	3	3 ms
4	20 ms	4	15 ms	4	2 ms
5	31 ms	5	27 ms	5	4 ms
6	56 ms	6	57 ms	6	7 ms
7	78 ms	7	110 ms	7	14 ms
8	79 ms	8	113 ms	8	26 ms
9	321 ms	9	230 ms	9	43 ms
10	770 ms	10	658 ms	10	94 ms
11	1662 ms	11	1529 ms	11	176 ms
12	3335 ms	12	3296 ms	12	399 ms
13	7748 ms	13	6260 ms	13	817 ms
14	15058 ms	14	11863 ms	14	1443 ms
15	19769 ms	15	24329 ms	15	2904 ms
16	18937 ms	16	23796 ms	16	5557 ms
17	72794 ms	17	81716 ms	17	12593 ms
18	197238 ms	18	184427 ms	18	24938 ms
19	446867 ms	19	413057 ms	19	48292 ms
20	913635 ms	20	775056 ms	20	100824 ms
21	1742859 ms	21	1635815 ms	21	190846 ms
22	3956987 ms	22	3160607 ms	22	409940 ms

TAB. 3.1 – Tableau des résultats du banc d'essai sur les "Computational Puzzles"

3.6 Les "Computational Puzzles" au banc d'essai

Pour comprendre le fonctionnement de cette technique et étudier les impacts du positionnement de la valeur de "work" (c'est-à-dire du niveau de difficulté du puzzle) sur le temps nécessaire à la résolution d'un puzzle, nous proposons ici un banc d'essai du prototype sur les "Computational Puzzles" réalisé au cours du stage.

Ce test a été effectué sur les configurations matérielles suivantes :
- Mono-processeur Pentium Celeron 600Mhz
- Mono-processeur Pentium IV 2.8Ghz avec Hyperthreading
- Bi-processeur Pentium XEON 3Ghz avec Hyperthreading

Ces tests ont été réalisés sur des valeurs de work allant de 1 à 22, en réalisant 100 fois chaque niveau de difficulté (le concept de "niveau de difficulté" étant assimilé à celui de "valeur de "work" "). Les valeurs obtenues sont présentées dans le tableau 3.1 :

Plusieurs remarques importantes sont à formuler.

D'une part, on constate des "sauts" dans les valeurs qui peuvent parfois sembler anormales. Par exemple, les temps nécessaires pour trouver une solution dans les 8 premiers niveaux ne semblent pas cohérents. Ceci est dû au fait que le nombre d'itérations **maximal** est très faible (2 itération maximum au niveau 1, 4 au niveau 2, 8 au niveau 3, ...). De ce fait, le temps nécessaire pour les opérations annexes (manipulation des chaînes de caractères, conversion de ces chaînes en bytes, puis en bits...) n'est plus négligeable, et vient interférer sur les valeurs de calcul.

D'autre part, selon le niveau, on connaît le nombre d'itérations **maximal**. Ceci ne signifie pas qu'un puzzle de niveau 20 ne peut pas être résolu en quelques millisecondes. La probabilité que ceci se produise est certes faible, mais cette éventualité n'est pas à exclure. C'est pour cette raison qu'il faut réaliser un nombre d'expériences statistiquement significatif. D'éventuels "sauts" ou valeurs semblant incohérentes peuvent également provenir du fait que chaque niveau n'a été réalisé que 100 fois, des contraintes de temps ayant empêché d'itérer davantage sur chaque valeur de "work" .

Pour conclure, il faut préciser que ces tests n'ont pas été réalisés dans le but de proposer des valeurs de références. Pour ce faire, il aurait fallu réaliser les tests sur de nombreuses configurations distinctes, dans différentes conditions de charge, ... Il s'agissait surtout ici de donner une vue globale de l'impact de la variation de "work" sur le temps d'exécution moyen nécessaire, et de montrer que dans la plupart des cas, il ne sera pas utile de proposer des puzzles d'une difficulté supérieure à 20, ce niveau demandant déjà plusieurs minutes de calcul à une machine considérée actuellement comme très puissante.

3.7 Forces de cette technique

La force principale de cette technique semble être sa souplesse face à des terminaux disposant d'une capacité de calcul différente. Grâce au positionnement d'un niveau de difficulté variable, on peut soumettre des problèmes à tout type d'UA. Sans cette fonctionnalité, les "Computational Puzzles" ne seraient pas utilisables : donner des puzzles de même difficulté à un poste fixe de 5Ghz ou à une unité mobile de 30Mhz serait évidemment une absurdité, tant au niveau du temps de calcul nécessaire qu'au niveau de la consommation de la batterie de tels appareils.

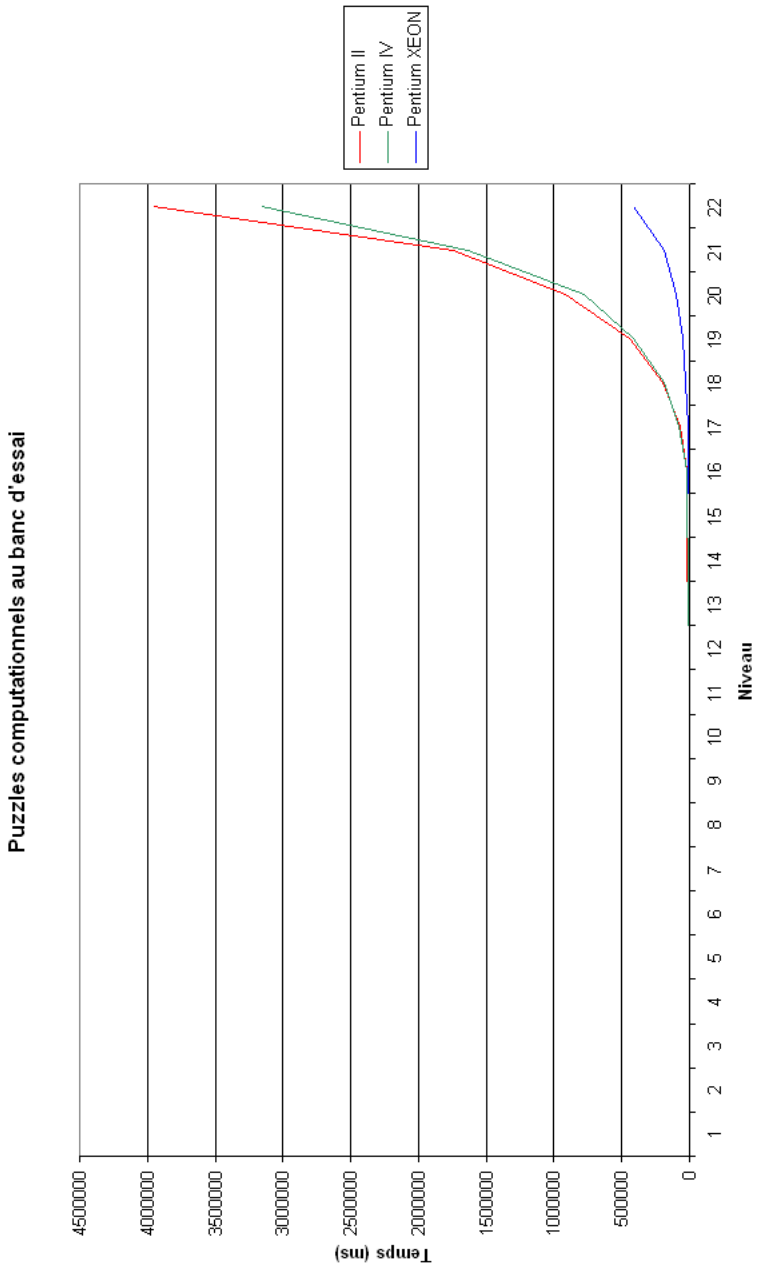


FIG. 3.4 – Résultats du banc d'essai sur les "Computational Puzzles" : échelle linéaire

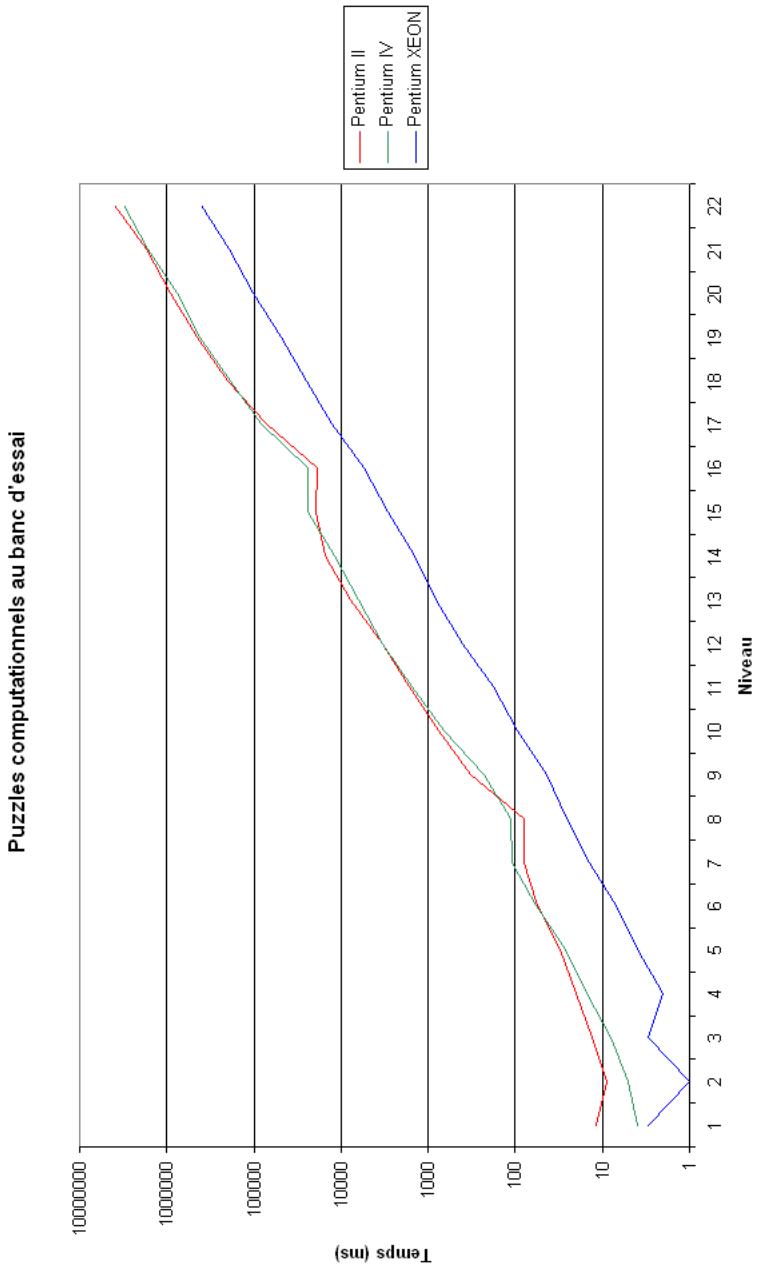


FIG. 3.5 – Résultats du banc d'essai sur les "Computational Puzzles" : échelle logarithmique

3.8 Faiblesses de cette technique

Assez paradoxalement, la force de cette technique constitue également sa faiblesse majeure : le positionnement d'un niveau de difficulté d'un puzzle à envoyer.

En effet, la présence de cette fonctionnalité ne résout pas du tout le problème : toute la question est "comment identifier univoquement le terminal de mon interlocuteur, pour me permettre d'en déduire sa puissance de calcul, et donc de lui fournir un problème de difficulté adaptée ?". En SIP, cette question n'a pas de réponse satisfaisante.

La notion de "Terminal Capabilities" est bien présente, mais rien n'affirme que cette valeur ne peut être intentionnellement modifiée. On pourrait donc, sans le savoir, soumettre à un interlocuteur prétendant posséder un téléphone portable de très faible puissance de calcul, un puzzle de très petite difficulté, alors qu'il possède en fait un cluster de plusieurs machines ultra-performantes !

Dans ce cas, on se retrouverait dans un monde où tous les spammeurs prétendraient posséder un petit terminal de (très) faible capacité, et où le SPAM serait toujours aussi rentable pour ces utilisateurs mal intentionnés…

Chapitre 4

Les mécanismes de consentement

La technique des “Computational Puzzles” ayant été étudiée et implémentée, notre souci de suivi des standards nous amena à nous pencher plus longuement sur la technique des mécanismes de consentement. Cette technique venait d’être mise à jour dans une nouvelle version des drafts la concernant ([8] et [2]).

Nous profitâmes de ce sursaut d’activité sur le sujet pour nous joindre aux débats, analyser cette technique en profondeur, et participer à l’amélioration des drafts en collaborant avec les auteurs.

4.1 Introduction

Actuellement, en SIP, tout utilisateur qui reçoit une requête à laquelle il ne souhaite pas répondre favorablement est en droit de rejeter cette requête. Toutefois, le message SIP constituant cette “invitation à communiquer” a circulé sur le réseau et a été transmis au destinataire sans son consentement explicite.

Cette situation peut donner lieu à deux problèmes majeurs : le SPAM et le “Denial of Service” (DoS).

Le problème du SPAM est principalement dû à la présence d’un comportement applicatif par défaut qui est adopté lors de la réception de certains types de messages, comme par exemple faire sonner le téléphone lors de la réception d’un INVITE. Lors de la réception d’un MESSAGE, le dérangement occasionné est encore plus grand, car le contenu du message est affiché ! Dans tous les cas, l’attention de l’utilisateur est attirée, ce qui pose problème lorsque les messages reçus sont non sollicités.

D’autre part, il est très aisément possible de faire du DoS en SIP, notamment en s’appuyant sur le comportement des noeuds capables de générer plusieurs messages sortants à partir d’un seul message entrant (“forking proxies”, URI-list services, . . .). Ces mécanismes constituent un moyen simple de faire du DoS en SIP.

Le draft [8] vise à établir les exigences nécessaires à la mise en place d’un système de consentement, afin d’éviter que des messages ne soient transmis à un destinataire sans son consentement explicite. Le second draft ([2]) propose un *framework* mettant en place cette solution.

4.2 “Consent Requirements”

Les principales exigences identifiées dans [8] sont les suivantes :

1. Empêcher les relais de délivrer un message SIP à un utilisateur, à moins que cet utilisateur en ait donné l’autorisation explicite.
2. Éviter que des relais SIP ne génèrent plus d’une requête sortante par requête entrante, sauf si le destinataire en a donné l’accord explicite.
3. Avoir la capacité de spécifier que les messages provenant d’un certain utilisateur sont autorisés.
4. Permettre à un utilisateur de déterminer des permissions différentes pour chaque ressource susceptible de relayer des requêtes vers lui.
5. Permettre la spécialisation des permissions selon le type de requête (INVITE, MESSAGE, …).
6. Avoir la capacité de retirer des permissions, ou de limiter leur validité dans le temps.
7. Ne pas contraindre les UA à devoir stocker des informations pour pouvoir retirer des permissions
8. Être compatible avec les méthodes SIP actuelles et futures, avec les “forking proxies”, …
9. Fonctionner en inter-domaine, sans relation pré-établie entre domaines.
10. Autoriser les appels anonymes, à condition que le destinataire accepte de tels appels.
11. Résister aux attaques tentant d’outrepasser ces permissions.
12. Ne pas obliger le destinataire à être en ligne lorsqu’une demande de permission est envoyée.
13. Ne pas obliger l’expéditeur à être en ligne lorsque le destinataire accorde la permission.
14. Ne pas créer un trafic substantiellement plus important.
15. Être échelonnable au déploiement sur Internet.

4.3 Consent Framework

A partir des exigences précitées, un *framework* a été proposé dans [2]. Le problème, ainsi que les solutions proposées, sont présentées en différentes étapes : tout d'abord en peer to peer, puis en introduisant le concept de relais, puis avec des serveurs de permissions. Plusieurs scénarios sont alors présentés.

4.3.1 Communication directe entre UA

Dans ce cas de figure, le plus simple qui soit, on se base sur le principe même de consent : l'UAC doit avoir la permission de l'UAS pour pouvoir communiquer avec lui. Ce cas de figure est très répandu en IM. Un scénario de cas d'utilisation est présenté à la section 4.3.4 de ce chapitre.

4.3.2 Le concept de relais

Un relais est défini comme étant n'importe quel serveur SIP (*proxy*, "Back to Back User Agent" (B2BUA), ou hybride) qui effectue une traduction de l'URI qu'il reçoit en entrée en une nouvelle URI vers laquelle il peut rediriger la requête. C'est ainsi que sont introduits les concepts de "target URI" (l'URI d'une requête entrante dans une entité qui effectuera une traduction) et de "recipient URI" (l'URI d'une requête sortant d'une entité, qui est donc peut-être le résultat d'une opération de traduction), illustrés dans la figure 4.1.

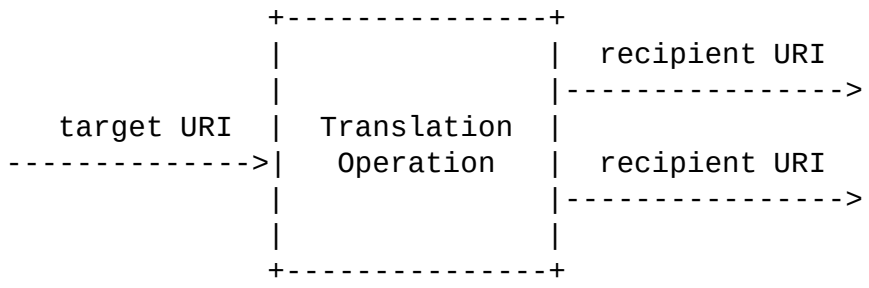


FIG. 4.1 – Opération de traduction (source : [2])

C'est cette traduction effectuée par le relais qui pose tout le problème du consentement. De plus, c'est la traduction d'une "target URI" en plusieurs "recipient URI" qui pose le problème de l'amplification.

Attention ! Ce concept de traduction se présente à deux niveaux distincts :
- la traduction d'une AoR en adresse d'UA.
  Exemple : "bob@example.com" → "bob@pc33.example.com"
- la traduction de l'URI d'une liste de diffusion en une série d'AoR de destinataires.
  Exemple : "bob-friends@example.com" → "alice@a.com", "carol@b.com", ...

Dans le premier cas, la traduction est effectuée grâce aux informations fournies lors du REGISTER. Ainsi, l'expéditeur adresse son message à l'AoR du destinataire, et une fois arrivé au premier *proxy* responsable du domaine du destinataire, ce *proxy* lira les informations fournies par le destinataire lors de son/ses REGISTER, pour effectuer la traduction d'URI nécessaire pour atteindre ce destinataire.

Dans le second cas, c'est le relais responsable de la liste de la diffusion qui se chargera de la traduction de l'URI de cette liste de diffusion en une liste d'URI de destinataires.

4.3.3 Structure d'une permission

Comme précisé, l'idée de [2] est de n'autoriser un relais à effectuer une traduction que s'il en a reçu la permission. Comment le concept de "permission" est-il matérialisé ?

Dans ce draft, on définit le concept de permission comme un objet représenté en XML et pouvant contenir les données suivantes :

Identité de l'expéditeur : l'URI de l'expéditeur à qui les permissions seront accordées.

Identité du destinataire originel : la "target URI".

Identité du destinataire final : la "recipient URI".

Opérations permises : les méthodes auxquelles s'appliquent la permission.

Signature : signature digitale signée par l'entité capable de s'identifier comme "recipient URI".

Un exemple de permission est proposé dans la figure 4.2. Il peut être traduit par : "J'autorise le relais gérant
la liste `bobs-friends@example.com` à relayer à Alice (`alice@example.com`) les requêtes INVITE provenant de n'importe quel expéditeur".

```
<target>
  <identity>
    <id>bobs-friends@example.com</id>
  </identity>
</target>
<sender>
  <identity>
    <any-identity/>
  </identity>
</sender>
<recipient>
  <identity>
    <id>alice@example.com</id>
  </identity>
</recipient>
<operations>
  <method>INVITE</method>
</operations>
```

FIG. 4.2 – Exemple de permission (source : [2])

4.3.4 Scénarios

Peer to peer

Ce scénario (figures 4.3 et 4.4) illustre le cas basique d'une demande de consentement entre deux UA, où A veut obtenir de B la permission de le contacter, c'est-à-dire de lui envoyer un INVITE.

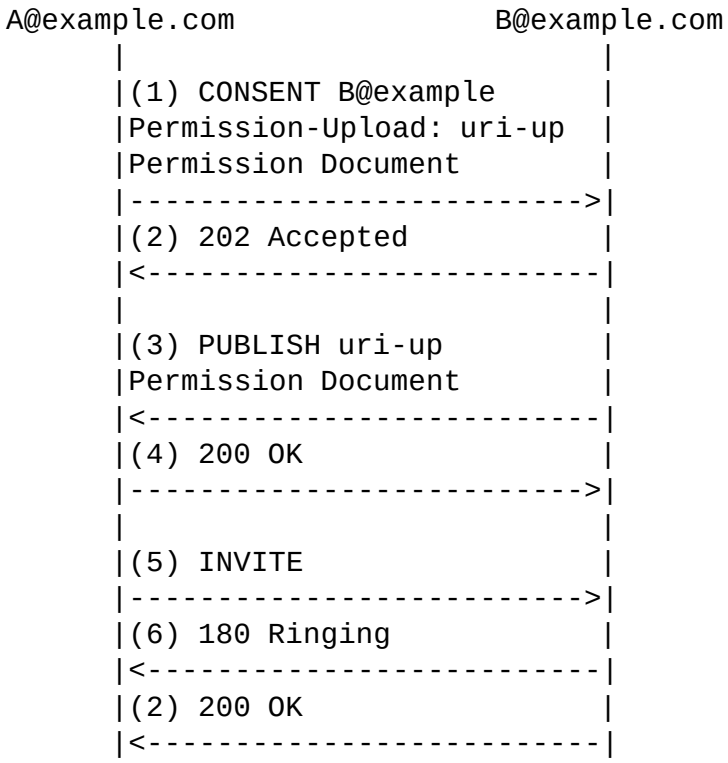


FIG. 4.3 – Scénario peer to peer : diagramme (source : [2])

Le document publié par B est donc le suivant :

```
<target>
  <identity><any-identity/></identity>
</target>
<sender>
  <identity><id>A@example.com</id></identity>
</sender>
<recipient>
  <identity><id>B@example.com</id></identity>
</recipient>
<operations><method>INVITE</method></operations>
```

FIG. 4.4 – Scénario peer to peer : document (source : [2])

Serveur de permission

Que se passe-t-il si A envoie un CONSENT à B alors que celui-ci n'est pas connecté ? Il faut alors trouver un moyen de stocker ces demandes de nouvelles permissions, et d'informer B des permissions qu'il a reçues pendant son absence (comme c'est actuellement le cas en Presence). Pour ce faire, on crée le concept de serveur de permissions : ces serveurs conservent les demandes de permissions, et informent l'utilisateur des demandes de consentement qu'il a reçues pendant son absence, si toutefois il a souscrit au "grant-permission event package".

Le serveur, pour stocker les permissions et permettre à l'utilisateur de gérer ses permissions (ajout, édition, suppression), dialoguera avec un serveur XCAP.

Le scénario est proposé dans la figure 4.5.

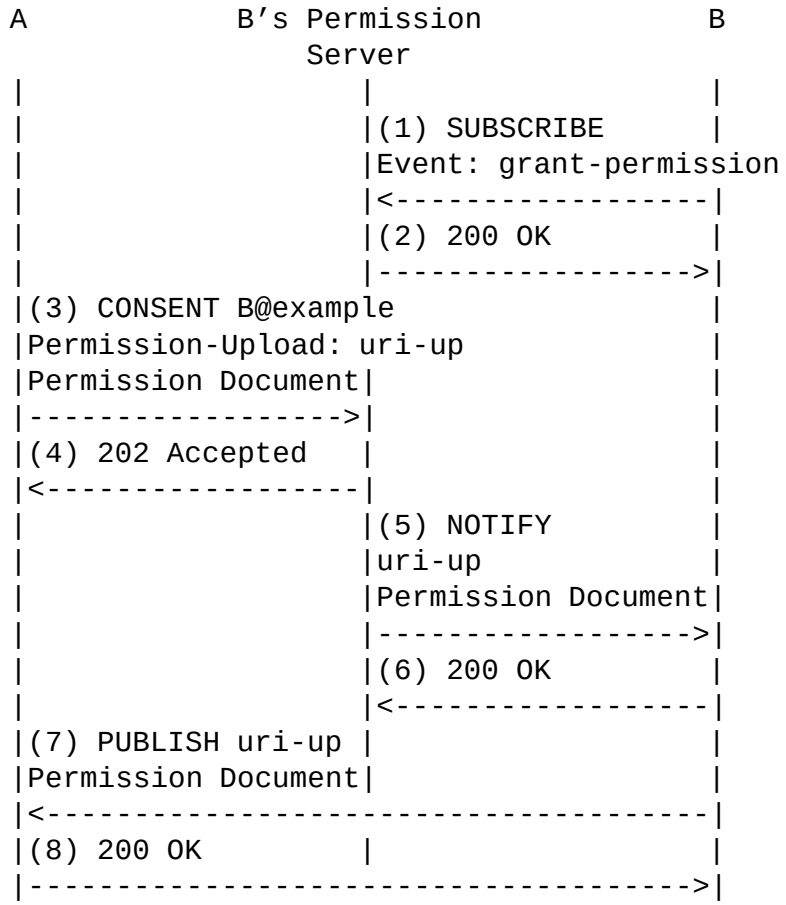


FIG. 4.5 – Scénario avec serveur de permissions (source : [2])

Le cas des listes de diffusion

Le cas des listes de diffusion mérite d'être présenté dans un scénario particulier, et est illustré par la figure 4.6. En effet, lorsque A souhaite ajouter B à sa liste de diffusion, il doit procéder en deux étapes :

1. Ajouter B à la liste de diffusion sur le relais. A ce stade, le relais n'est pas encore autorisé (par B) à effectuer la traduction "supplémentaire" due à l'inscription de B.
2. Autoriser le relais à effectuer la traduction supplémentaire due à l'inscription de B en :
   – Demandant autorisation à B, par A.
   – Recevant la permission signée provenant de B.
   – Publiant cette permission sur le relais.

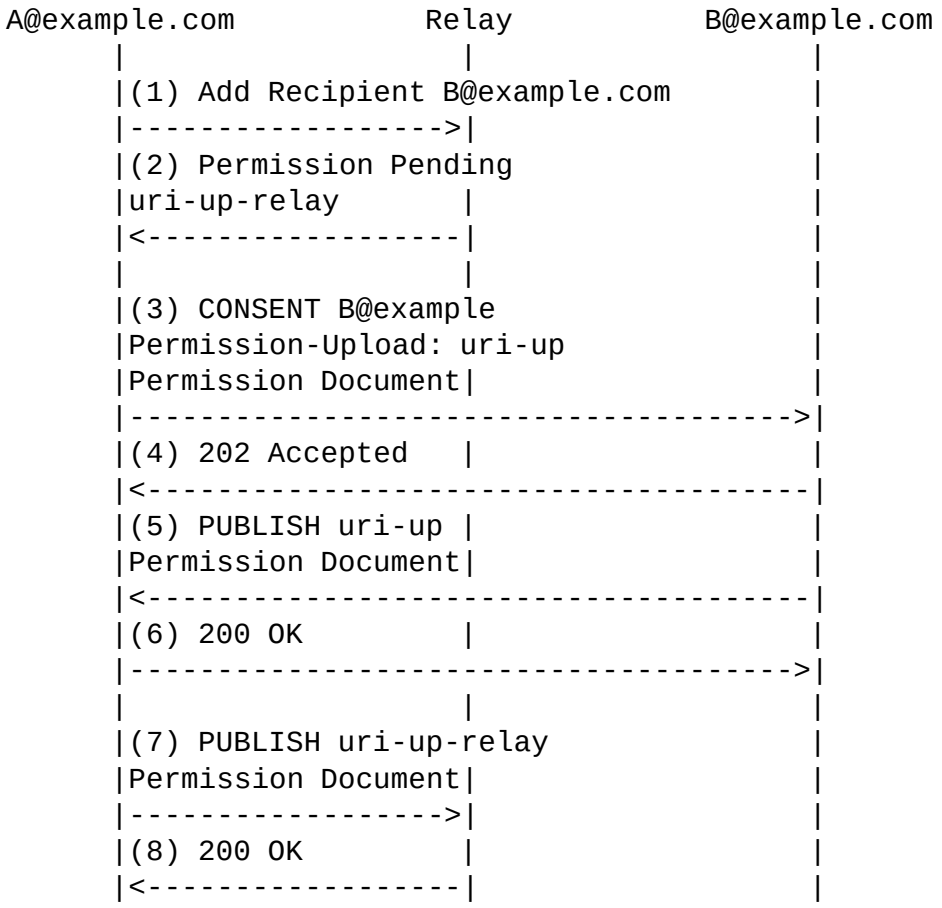


FIG. 4.6 – Ajout d'un utilisateur à une liste de diffusion (source : [2])

On peut même imaginer un système plus sophistiqué, où le relais se chargerait lui-même d'obtenir les permissions nécessaires à l'ajout d'un utilisateur sur une liste de diffusion.

Attention ! Si on se contentait d'adapter le schéma précédent en spécifiant que c'est le relais qui contacte B pour lui demander la permission, une faille apparaîtrait vis-à-vis du DoS : il suffirait à A d'ajouter un très grand nombre d'utilisateurs à une liste, et l'envoi d'**un seul** message au relais avec cette nouvelle liste contenant n personnes provoquerait l'envoi par le relais de n demandes de consentement !

Pour contrer ce phénomène d'amplification, on impose à A de consommer autant de bande passante que le relais n'en consommera pour effectuer ses demandes de permissions. Pour ce faire, en plus de l'envoi d'une nouvelle version de la liste par A, on demande à A d'envoyer un message REFER défini comme une demande explicite au relais d'envoyer un CONSENT à un utilisateur donné. Ainsi, si A ajoute n personnes à sa liste, il devra envoyer un message avec les nouveaux destinataires qu'il souhaite ajouter à sa liste de diffusion, ainsi que n REFER demandant au relais d'envoyer un total de n CONSENT. La figure 4.7 illustre ce cas d'utilisation.

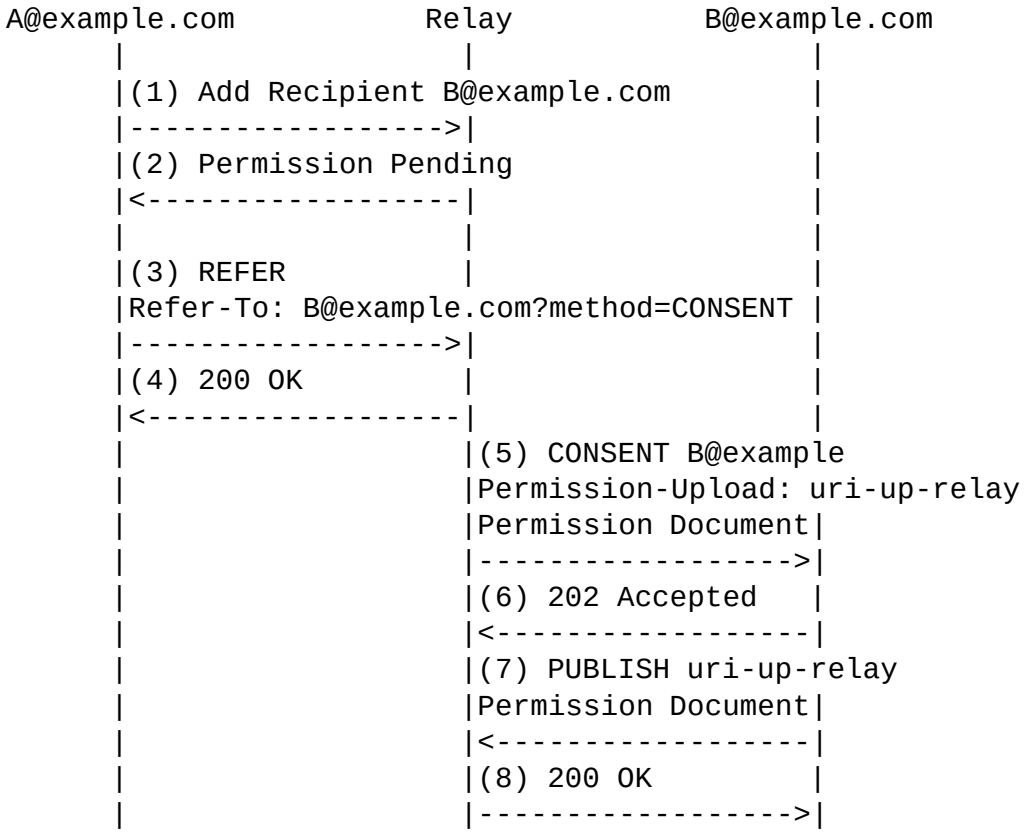


FIG. 4.7 – Relais obtenant des permissions avec prévention d'amplification (source : [2])

Un scénario d'établissement de communication complet

Nous pouvons maintenant nous pencher sur un scénario complet (figure 4.8), présentant l'établissement d'une communication entre un UAC et deux UAS dont un de ceux-ci n'a pas encore autorisé l'UAC à communiquer avec lui.

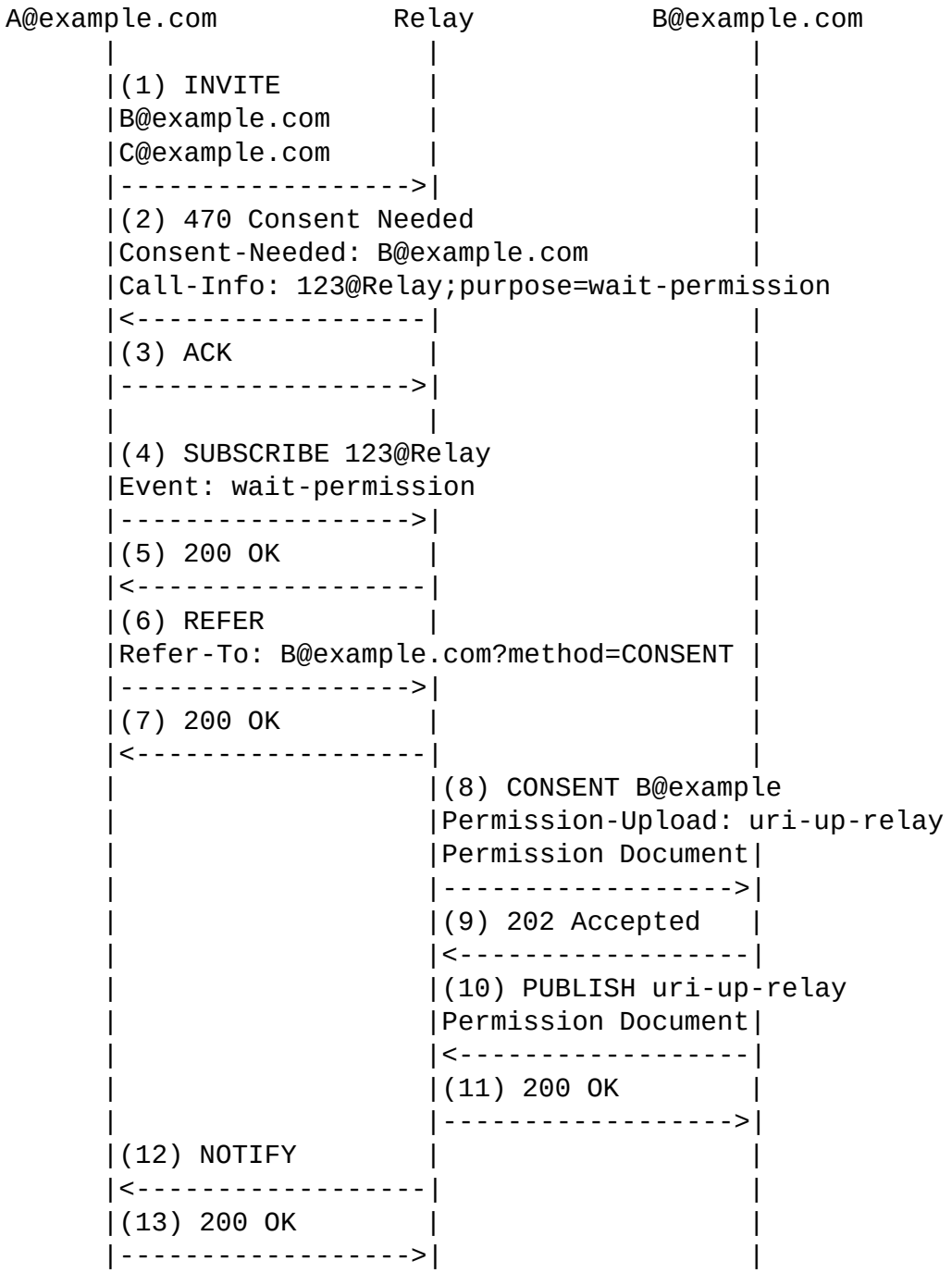


FIG. 4.8 – Établissement d'une communication (source : [2])

Dans ce scénario, A invite B et C (1), mais le relais signifie à A que le consentement de B est requis (2). A invite le relais à solliciter le consentement à B (6), ce qu'il fait (8). B autorise finalement A à le contacter, en publiant auprès du relais une permission "signée" (10). Parallèlement à l'obtention du consentement, A souscrit à l'évènement "wait-permission" (4), ce qui lui permettra d'être averti par le relais (12) de la présence d'une nouvelle autorisation le concernant.

4.3.5 Le cas du REGISTER

Le cas du REGISTER est un cas tout à fait différent. En effet, on pourrait supposer aux premiers abords qu'effectuer un REGISTER auprès de son registrar pourrait s'effectuer sans consentement, puisqu'aucune communication avec un autre utilisateur n'est nécessaire, et que ne se posent ici ni le problème du SPAM, ni celui du DoS.

Toutefois, le callflow proposé dans [17] pourrait donner lieu à un schéma d'attaque : une personne malintentionnée (Trudy) effectue un REGISTER entre **son** AoR et l'URI d'une victime (Alice). A partir de là, Alice recevra tout le trafic adressé Trudy !

Le *framework* proposé peut combler cette faille de sécurité : comme le montre la figure 4.9, lors d'un REGISTER, le registrar demande autorisation à l'URI de l'UA d'effectuer cette nouvelle traduction. Une traduction due à un REGISTER ne peut donc plus se faire sans le consentement explicite du propriétaire de l'UA cible.

On notera l'utilisation de la méthode REFER malgré l'absence d'une liste de diffusion. REFER est pourtant utilisé dans un but analogue : demander au serveur gérant les permissions (dans ce cas de figure, le registrar) de se charger d'obtenir les permissions.

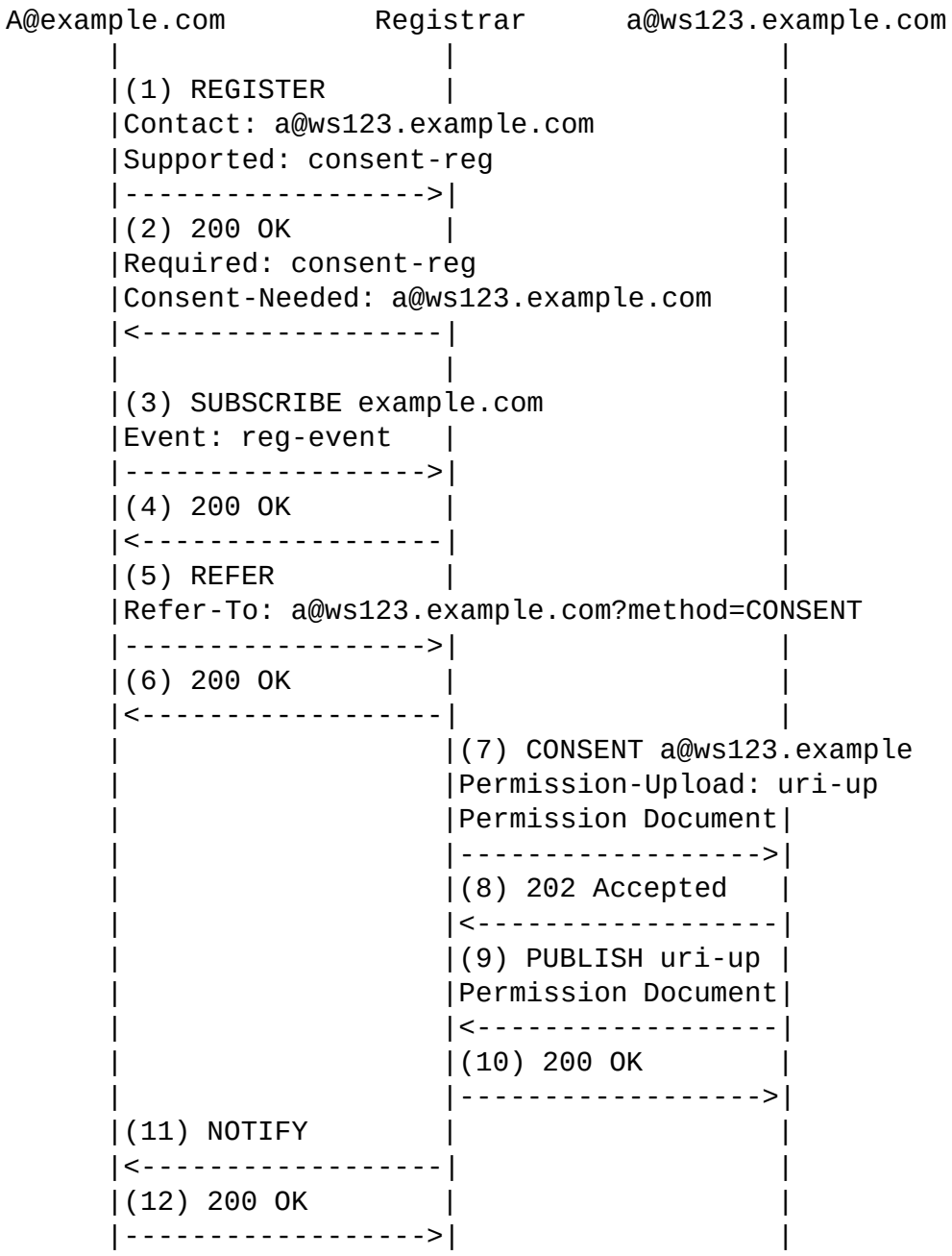


FIG. 4.9 – Un REGISTER avec consentement (source : [2])

4.3.6 Un point de vue “sécurité”

Pour garantir, lors de la publication de documents de permissions, l’identité de l’émetteur de ce document, deux techniques sont possibles : “SIP Identity”, ou un “return routability test”.

“SIP Identity” a été présenté dans la section 2.1.2 de ce travail. La seconde technique n’offre, selon les auteurs du draft, pas le même niveau de sécurité. Elle peut néanmoins offrir un niveau de sécurité acceptable si “SIP Identity” n’est pas disponible. Le relais génère une URI théoriquement impossible à deviner (c’est-à-dire dont la partie “user”, à gauche du “@” a été générée aléatoirement) et la place dans la requête CONSENT. Le destinataire doit publier la permission à cette URI. Cela permet d’être sûr que seuls ceux qui ont manipulé le CONSENT savent où obtenir le document de permission.

4.3.7 La révocation de permissions

Pour révoquer une permission, un utilisateur doit attendre d’avoir reçu une nouvelle requête relative à cette permission. Cette requête contiendra un *header* “Permission Used” contenant une URI renseignant l’emplacement du document de permission utilisé pour effectuer la traduction, et une URI où l’utilisateur peut uploader un nouveau document de permission (par exemple une nouvelle version des permissions, ou une révocation de celles-ci).

4.4 Critiques

Bien que cette technique semble couvrir de nombreux cas de figure et résister à de nombreux types d’attaques, elle n’est pas non plus exempte de défauts. Les failles les plus probantes de cette technique sont les suivantes :

- Le cas le plus simple d’application de cette technique est le scénario “peer to peer”, présenté dans la section 4.3.4 de ce chapitre. Dans ce scénario, on suppose que l’UAC commence toute communication par l’envoi d’un CONSENT. On peut raisonnablement supposer qu’un UAC malhonnête (un spammeur en fait), n’aura pas la politesse d’initier le scénario par l’envoi d’un CONSENT. Dans ce cas de figure, aucun scénario pour protéger le récepteur n’est prévu.
- Alors que l’exigence “19” du draft [8] précise que la solution ne devra pas créer un trafic substantiellement plus important, on oblige tout UAC, pour contourner le problème d’amplification, à envoyer une requête REFER pour chaque utilisateur ajouté à une liste de diffusion (cfr section 4.5). Ceci double tout bonnement le trafic utilisé pour ajouter un utilisateur à une liste de diffusion.
- Cette exigence “19” du draft [8] est une nouvelle fois violée dans le cas d’une protection de l’UAC lors du REGISTER (cfr section 4.3.5). En effet, alors qu’un REGISTER provoque l’envoi de 2 requêtes dans un scénario classique, la version proposée ici nécessite l’envoi de 12 requêtes, soit six fois plus !
- Le draft traite de “documents de permissions”, sans avoir spécifié la syntaxe de ces documents. Malgré l’apparition de nouvelles versions du draft [2], aucun “DTD” ou “XML Schema” n’a été introduit.

De plus, et au delà d'une étude sur "le fond" du sujet, la forme actuelle des drafts [8] et [2] n'est pas satisfaisante. En effet, si la technique semble répondre à de nombreux cas d'utilisation (scénarios "peer to peer", introduction de relais, listes de diffusions, cas isolé du REGISTER, ...), ces cas ne sont pas présentés clairement et distinctement, et il devient finalement impossible de savoir où la technique de consentement pourra être concrètement utilisée.

Enfin, le problème le plus gênant est, au delà du manque de maturité du draft [2], le manque de participations actives autour de ce sujet. Nous avons fait de nombreuses propositions à l'IETF et auprès des auteurs du draft pour aider leur travail à évoluer, et notamment :

- En soumettant une proposition définition formelle (DTD) du format de document de permissions.
- En proposant une extension de XCAP utilisant la "constraint failure" définie dans [10] pour empêcher l'ajout intempestif d'utilisateurs à une liste de diffusion.
- En utilisant le mécanisme de "throttle" (également défini dans [10]) pour éviter qu'un serveur soit noyé par la réception de requêtes identiques envoyées durant un court intervalle de temps.

Malheureusement, aucune réponse n'ayant été donnée, nous n'avons pas pu envisager l'implémentation de cette technique dans un prototype.

Chapitre 5

Vers une plate-forme anti-SPAM générique

Après avoir étudié deux techniques anti-SPAM “autonomes”, nous proposâmes de passer à un troisième mécanisme, utilisant les *blacklists* ou *whitelists* “basiques”, mais en mettant à profit les infrastructures dont nous disposions pour fédérer ces listes. En effet, le “Presence Server” réalisé chez Alcatel nous permettait d’utiliser un serveur où stocker ces documents, et puisqu’il s’agissait d’un serveur XCAP ([10]), il permettait même de gérer le contrôle d’accès et la possibilité d’ajouter, d’éditer ou de supprimer des fragments de documents.

Une fois la spécification de ce service terminée, nous réalisâmes que cette solution n’était pas suffisamment complète. En effet, certains services supplémentaires auraient été désirables (dont certains inspirés des mécanismes de consentement étudiés dans le chapitre précédent), et notamment :

- la possibilité de filtrer les requêtes au niveau d’un *proxy* (et non plus chez le destinataire), pour arrêter au plus tôt le trafic inutile
- la possibilité de définir des critères de filtrage plus riches que la valeur du *header* From
- la possibilité de définir un comportement spécifique selon le type de SPAM identifié (bloquer, accepter, mettre un *flag*, …)
- la possibilité de ne pas laisser tous les utilisateurs intervenir sur une seule et même liste (pour éviter l’ajout abusif de critères)
- …

Après avoir intégré ces fonctionnalités, le service était correctement spécifié mais une nouvelle opportunité se présenta : plutôt que de nous contenter d’implémenter quelques réactions “basiques” face au SPAM, nous pouvions proposer une interface qui permettrait à n’importe quel développeur d’implémenter une quelconque nouvelle réaction face au SPAM correspondant à ses besoins, sans même accéder à nos sources ou les modifier.

La concrétisation de cette opportunité nous amena à la spécification d’un *framework* générique, filtrant les requêtes entrantes au niveau de n’importe quel relais et permettant d’adopter tous les comportements souhaités face au SPAM, spécifiés dans n’importe quelle technique anti-SPAM actuelle ou future. Cette spécification vous est présentée dans le présent chapitre.

5.1 Introduction

Dans le cadre de la lutte contre le SPAM en SIP, Alcatel propose de réaliser un *framework* basé sur la présence de règles stockées dans des documents, permettant de lutter contre la livraison de requêtes non sollicitées. Différents composants permettent d'une part de fournir ces documents de règles, et d'autre part de les exploiter pour prendre différentes décisions de filtrage, et au besoin d'exécuter des actions déterminées pour combattre le SPAM.

5.2 Vue d'ensemble de la solution

Le principe de ce *framework* consiste en un service supportant les fonctionnalités suivantes :

- Service de filtrage des requêtes entrantes basé sur la présence des règles placées dans des "documents de règles".
- Support d'un document de règles personnel à chacun des utilisateurs.
- Support d'un document de règles commun à tous les utilisateurs.
- Hiérarchisation possible des documents de règles pour les utilisateurs de différents domaines/sous-domaines.
- Système automatique de tentative de mise en commun entre les documents personnels et le document commun.
- Dans les documents de règles, gestion de ces règles selon différents critères :
  1. Valeur de certains *headers* (ex : "From", "Contact", ...)
  2. Valeur de la request URI (ex : "INVITE **sip :bob@example.com**")
  3. Méthode invoquée par le caller (ex : "INVITE", "MESSAGE", ...)
  4. Présence de certaines expressions dans le *body*
  5. Support des *wildcards* (ex : "*@spammer.com")
  6. Support de deux expressions particulières : "all" et "allow"[1]
  7. Absence d'un certain *header*
  8. ou toute combinaison logique (and, or, not[2]) de ces critères
- Support d'une interface d'administration permettant à un administrateur de configurer le système.
- Service d'identification automatique de toute nouvelle forme de SPAM.
- Possibilité d'introduire des "quotas", interdisant aux utilisateurs de posséder un document de règles trop long.
- Possibilité de demander au système une réaction différente selon le type de SPAM identifié.
- Solution indépendante du protocole (SIP, "HyperText Transfer Protocol" (HTTP), ...).
- Possibilité, pour le client, d'intégrer ses propres réactions face aux différents types de SPAM, ou d'adapter les réactions existantes, sans adaptation du *framework*.

[1] Leur utilisation permettra d'offrir plusieurs services intéressants, cfr sections 5.3.5 et 5.3.5

[2] Ceci permettra une extension très intéressante de ce *framework*, cfr section 5.3.5

5.3 Principes

Comme décrit dans la section précédente, notre solution supportera une liste par utilisateur, et une liste commune aux utilisateurs du domaine[3]. Pour garantir la persistance des données, et permettre la migration de données d'une liste vers une autre (fusion de documents, migration de règles, ...), le format XML sera choisi pour les documents de règles, stockés sur un serveur XCAP.

5.3.1 Intégration du format de règles à l'existant

Avant de proposer notre propre format de spécification de règles, il fallait se tourner vers les standards existants, et notamment vers SAML (section 2.2.7) et Geopriv (section 2.2.8), pour tenter d'offrir un nouveau service qui soit en harmonie avec l'existant.

D'une part, Geopriv définit un *framework* générique qui doit être étendu par chaque nouveau service applicatif. Les formats de documents utilisés pour offrir de nouveaux services liés à SIP doivent être spécifiés comme des extensions du format défini dans Geopriv ("Conditions"/"Actions"/"Transformations"). Ceci permet de garantir que les infrastructures sur lesquelles repose chaque service restent cohérentes entre elles, et que les nouveaux services continuent à respecter les règles définies dans [18]. La possibilité de rendre les services interopérables est donc grandement améliorée.

D'autre part, il conviendrait de véhiculer des informations relatives à la sécurité, initialement véhiculées dans des documents SAML. Ainsi, le prorotype se verrait enrichi par la présence de ces informations, ce qui permettrait de définir des critères de filtrage plus riches, donc mieux adaptés aux besoin des utilisateurs.

Plusieurs points apparaissent toutefois comme problématiques pour définir un format de documents répondant à nos besoins comme une extension de Geopriv.

Tout d'abord, la possibilité d'offrir une réaction "par défaut" face au SPAM ne semble pas compatible avec Geopriv. En effet, si chaque action peut être placé dans l'élément "actions" de Geopriv, où pourrait se positionner une action par défaut dans un document Geopriv ? En effet, aucun emplacement en tête de document (comme c'est le cas avec notre solution, cfr. plus bas) n'est prévu pour offrir ce genre de fonctionnalité. Ceci constitue un premier obstacle à définir un format de document comme une extension de Geopriv.

[3]Le fonctionnement de notre système pourra toutefois être plus générique grâce à une hiérarchisation des documents de règles en sous-domaines, cfr section 5.6.1

Ensuite se pose un problème de gestion des conditions. Le prototype spécifié dans ce chapitre souhaite adopter une politique simple face à la présence de plusieurs règles correspondant avec une requête entrante : on retient la première règle, et on écarte les autres. Ce comportement, qui peut paraître simpliste, semble toutefois présenter de nombreux avantages. En effet, il rend non seulement possible de supporter très aisément un mécanisme de surcharge des règles (cfr. section 5.3.5), mais il permet en plus d'éviter de parcourir l'entièreté des documents alors qu'une règle correspondante a été trouvée. Dans Geopriv, la politique de gestion des conditions est plus élaborée : on parcourt l'entièreté des règles, et on utilise l'algorithme présenté dans la section 2.2.8 pour construire une règle représentant l'union de toutes celles qui correspondent avec une requête entrante. L'application de cet algorithme empêche de supporter "nativement" le mécanisme de surcharge (introduit précédemment et présenté dans la section 5.3.5) que nous souhaitons.

Pour ces raisons, le format de documents spécifié ci-dessous n'a finalement pas été défini comme une extension de Geopriv. Quant à la possibilité dégagée par SAML de véhiculer des informations liées à la sécurité, la première version du service anti-SPAM ne l'a pas exploitée. Toutefois, cette opportunité n'a pas été négligée lors de la création de notre draft [19], dont la dernière version (qui aborde plus largement les problèmes précités et tente d'y apporter des réponses) est proposée en annexe A.

5.3.2 Structure d'un document de règles

Un document de règles est un document XML qui devra être bien formé et qui devra respecter la DTD fournie dans ce document. Il devra être basé sur XML 1.0 et être encodé en UTF-8.

L'élément "racine" du document est "rules-document", et il ne comporte pas d'attributs. Sous cet élément, on trouve un premier sous-élément, facultatif : "default-action". Il contient la réaction à adopter par défaut en cas de SPAM détecté[4].

Après cet élément, on trouve une série d'éléments "rule". Chaque élément "rule" peut contenir un premier élément, facultatif : "action", qui contient la réaction à adopter en cas de SPAM détecté sous CETTE règle.

Dans "rule", on trouve UN des éléments suivants :
- "and"
- "or"
- "not"
- "field"
- "all"

[4]Cette fonctionnalité est décrite dans la section 5.3.4

Les éléments “and” et “or” doivent contenir au moins deux éléments “field”. L’élément “not” doit contenir un et un seul élément “field”. L’élément “field” contient :

– soit un élément “Header”, contenant deux éléments obligatoires :

  1. “name” : contenant le nom du *header* concerné, auquel on peut adjoindre un attribut facultatif, “position”, permettant de détecter la présence d’un terme à une position déterminée (pour les *headers* multivalués)
  2. “value” : mentionnant la valeur de ce *header*, OU “missing”, un élément vide signifiant que l’on souhaite détecter l’absence du *header* concerné dans la requête

– soit un élément “Body”, “RequestURI” ou “Method”, contenant un élément “value” qui mentionne la valeur du champ “Body”, “RequestURI” ou “Method” concerné.

Les valeurs mentionnées par l’utilisateur, à l’exception du nom de la méthode SIP invoquée, pourront contenir des *wildcards*. En d’autres termes, les éléments “value” qui ne sont pas enfants d’un élément “Method” pourront contenir des *wildcards*.

5.3.3 DTD d’un document de règles

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT rules-document (default-action?, rule*)>
<!ELEMENT default-action (#PCDATA)>

<!ELEMENT rule (action?,(and | or | not | field | all))>

<!ELEMENT action (#PCDATA)>
<!ELEMENT and (field, field+)>
<!ELEMENT or (field, field+)>
<!ELEMENT not (field)>
<!ELEMENT all (EMPTY)>

<!ELEMENT field  ((Header,name,(value|missing)) |
                 ((RequestURI|Method|Body),value))>

<!ELEMENT missing (EMPTY)>

<!ELEMENT name (#PCDATA)>
<!ATTLIST name position CDATA "0">

<!ELEMENT value (#PCDATA)>
```

5.3.4 Actions à entreprendre

Il a été précisé dans la vue d'ensemble de la solution qu'une réaction spécifique à chaque type de SPAM identifié pourrait être entreprise, grâce aux éléments "action" et "default-action" définis dans le point précédent. En quoi ceci consiste-t-il ?

Comme cela a été vu dans la structure du document de règles, chaque règle pourra mentionner le comportement à adopter en cas de SPAM détecté sous cette règle. Pour respecter les fonctionnalités attendues du système, le *framework* devra également permettre à l'utilisateur :

- de demander au système de déclencher des réactions "standard"
  Exemple : au départ, le système permet à l'utilisateur d'adopter un comportement consistant à bloquer certaines requêtes, tout en répondant à l'émetteur par un message de refus
  Application : bloquer le message et répondre "403 Forbidden"

- de "surcharger" certaines réactions "standard" pour répondre à des besoins spécifiques
  Exemple : spécialiser le comportement précédent (réaction de refus) en y adjoignant une raison particulière
  Application : bloquer le message et répondre "403 Forbidden" en y adjoignant, selon le pays, un message signalant que le SPAM est passible de poursuites judiciaires conformément à la loi XXX.XX de ce pays

- d'ajouter de nouvelles réactions sans devoir réadapter le *framework* anti-SPAM
  Exemple : Supporter une nouvelle technique anti-SPAM
  Application : Demander un test de Turing lors de la réception de certains messages.

Pour offrir ce type de service, notre *framework* fonctionnera de la façon suivante :

1. Les actions à entreprendre seront identifiées par un mot-clé, présent dans l'élément "action" ou "default-action" décrits dans la section précédente.
2. Notre *framework* offrira une interface contenant une méthode générique "doAction".

```
public interface AntiSPAMFramework {
  public void doAction(String action, String protocol, Object request);
}
```

3. Pour offrir quelques premiers services "de base", le *framework* sera fourni avec une classe implémentant cette interface : "AntiSPAMFrameworkImpl".

```
public class AntiSPAMFrameworkImpl implements AntiSPAMFramework {
  public void doAction(String action, String protocol, Object request) {
    if (action.equals(''block'')) doBlock(...);
    if (action.equals(''polite block'')) doPoliteBlock(...);
    ...
    else printErrorMessage(''Unsupported Action'');
  }
}
```

A partir de là, si un client souhaite ajouter des comportements selon ses besoins, il devra alors commencer par choisir un mot clé identifiant ce nouveau comportement, puis implémenter un nouvel objet

- supportant ce nouveau comportement
- implémentant notre interface de départ “AntiSPAMFramework”
- étendant notre “AntiSPAMFrameworkImpl” pour continuer à supporter les autres services de base

Pour adapter un des comportements “par défaut” du système, il fera de même, mais sans définition d’un mot clé supplémentaire.

Le paramétrage du système se fera via notre interface d’administration, qui sera présentée plus loin.

Exemple :

Reprenons le cas énoncé dans le point précédent, où un utilisateur souhaite spécialiser le comportement par défaut d’une action de blocage, en mentionnant à l’émetteur quelle loi nationale il transgresse en envoyant du SPAM.

Supposons que notre classe “AntiSPAMFrameworkImpl” soit de la forme :

```
public class AntiSPAMFrameworkImpl implements AntiSPAMFramework {
  public void doAction(String action, String protocol, Object request) {
    if (action.equals(‘‘block’’)) doBlock(...);
    if (action.equals(‘‘polite block’’)) doPoliteBlock(...);
    ...
    else printErrorMessage(‘‘Unsupported Action’’);
  }
}
```

Dans le cas où une règle donnée mentionne l’action “block”, le système se chargera d’effectuer l’appel de méthode “doBlock(...)” offrant un comportement de blocage par défaut, par exemple rejeter la requête et répondre “403 Forbidden” à l’émetteur.

Si l’utilisateur souhaite offrir un comportement plus riche (dans notre cas informer l’émetteur de la loi qu’il enfreint), l’utilisateur va créer un objet contenant une méthode “specificBlock(...)” dont cette méthode adopte le comportement spécifique souhaité (ici, fournir les informations annexes souhaitées) grâce à une classe de la forme :

```
public class MyClass extends AntiSPAMFrameworkImpl
                      implements AntiSPAMFramework {

  public void specificBlock(...) {
    //comportement original...
    super.doBlock( ...);

    //comportement spécifique supplémentaire
    showBlockReason(...);
  }

  public void doAction(String action, String protocol, Object request) {
    if (action.equals(''block'')) specificBlock(...);
    else super.doAction(action, protocol, request);
  }
}
```

Il lui suffira alors, via l'interface d'administration, de spécifier au *framework* que la classe implémentant maintenant les services anti-SPAM est maintenant "MyClass", et plus "AntiSPAMFrameworkImpl". A cet instant, le nouveau comportement souhaité sera adopté par le *framework*, et toutes les règles mentionnant une réaction "block" déclencheront le comportement de blocage qui a été spécialisé par l'utilisateur.

Dans le cas de l'ajout de nouveaux comportements, il lui suffira d'ajouter dans la classe implémentant "AntiSPAMFramework" une instruction "if" testant la présence du nouveau mot-clé et effectuant le nouvel appel de méthode implémentant le nouveau comportement. Pour en supprimer, il remplacera les appels de méthodes dans l'instruction "if" correspondant aux comportements à supprimer par des messages précisant que ce comportement n'est plus supporté.

5.3.5 Exemples d'utilisation du *framework*

Exemple 1 : Une *blacklist* "basique"

Le premier exemple est un document illustrant une mise en *blacklist* "basique".

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE exemple SYSTEM "rules-document.dtd" []>
<rules-document>
  <default-action>flag</default-action>
  <rule>
    <or>
      <field>
        <type>Header</type>
        <name>From</name>
        <value>sip:trudy@example.com</value>
      </field>
      <field>
        <type>Header</type>
        <name>From</name>
        <value>sip:advertiser@discount.com</value>
      </field>
      <field>
        <type>Header</type>
        <name>From</name>
        <value>sip:*@freeoffer.com</value>
      </field>
    </or>
  </rule>
</rules-document>
```

Ce document de règles illustre le comportement suivant :

- Il sera identifié comme SPAM toute requête provenant des URI suivantes : "sip :trudy@example.com", "sip :advertiser@discount.com", et "sip :*@freeoffer.com".
- Par défaut, lorsqu'une requête est identifiée comme SPAM, la réaction à adopter consiste à mettre un *flag* sur la requête. En effet, ce comportement est spécifié dans "default-action" et aucun champ "action" (spécialisant ce comportement par défaut) n'est mentionné dans "rule".

Exemple 2 : Un système “intelligent”

Un second exemple illustre “l’intelligence” qui peut être placée dans un document de règles, avec des critères d’identification variés et des réactions spécifiques à chaque type de SPAM identifié.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE exemple SYSTEM "rules-document.dtd" []>
<rules-document>
  <rule>
    <action>flag</action>
    <field>
      <type>Header</type>
      <name>From</name>
      <value>sip:*@spammer.com</value>
    </field>
  </rule>
  <rule>
    <action>politeblock</action>
    <and>
      <field>
        <type>Header</type>
        <name>To</name>
        <value>sip:alice@example.com</value>
      </field>
      <field>
        <type>Header</type>
        <name>Contact</name>
        <value>sip:satan@hell.org</value>
      </field>
    </and>
  </rule>
```

```
  <rule>
    <action>flag</action>
    <and>
      <field>
        <type>Header</type>
        <name>From</name>
        <value>sip:*@bavard.com</value>
      </field>
      <field>
        <type>Method</type>
        <value>MESSAGE</value>
      </field>
    </and>
  </rule>
  <rule>
    <action>block</action>
    <and>
      <field>
        <type>RequestURI</type>
        <value>sip:alice@example.com</value>
      </field>
      <or>
        <field>
          <type>Body</type>
          <value>mp3</value>
        </field>
        <field>
          <type>Body</type>
          <value>free</value>
        </field>
      </or>
    </and>
  </rule>
</rules-document>
```

Ce document de règles illustre donc le comportement suivant :

1. mettre un *flag* sur les requêtes dont le champ “From” contient le domaine “spammer.com”
2. rejeter les requêtes dont les messages sont adressés à “sip :alice@example.com” et dont le champ “Contact” contient “sip :satan@hell.org”
3. mettre un *flag* sur les requêtes de type “MESSAGE” et provenant du domaine “bavard.com”
4. répondre par un “403 Forbidden” aux requêtes dont les messages sont adressés (via la request URI) à “sip :alice@example.com” et dont le *body* contient les termes “mp3” ou “free”

Exemple 3 : Support des *whitelists*

Dans notre solution, le support de l'élément "not" permet une extension très intéressante du système : le support des *whitelists*.

En effet, supposez que vous ne souhaitiez recevoir que les requêtes provenant d'une seule URI : "sip :darling@example.com". En d'autres termes, vous souhaitez mettre en place une *whitelist* contenant un seul utilisateur : "sip :darling@example.com".

Ceci peut être très aisément mis en place dans notre système. Un exemple de document de règles permettant cette fonctionnalité est proposée ici :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE exemple SYSTEM "rules-document.dtd" []>
<rules-document>
  <rule>
    <action>block</action>
    <not>
      <field>
        <type>Header</type>
        <name>From</name>
        <value>sip:darling@example.com</value>
      </field>
    </not>
  </rule>
</rules-document>
```

En effet, ce document bloquera tous les messages ne provenant pas de "sip :darling@example.com", ce qui équivaut à dire que l'on n'accepte QUE les messages provenant de "sip :darling@example.com". En mentionnant plusieurs règles de cette forme, un système de *whitelists* est offert à l'utilisateur.

Exemple 4 : L'action `allow`

Cet exemple montre l'utilité de l'action "allow". Supposez que dans la liste commune se trouve un critère consistant à rejeter les messages provenant de "promos@supermarche.be". Si un utilisateur est intéressé par les messages provenant de cette adresse, comment peut-il s'y prendre ?

Pour résoudre ce problème, il faut imaginer une possibilité de "surcharger" une règle commune existante avec un critère particulier de son document de règles. Pour rester dans le cadre de l'exemple précédent, en supposant qu'une des règles de la liste commune est la suivante :

```
<rule>
  <action>block</action>
  <field>
    <type>Header</type>
    <name>From</name>
    <value>sip:promos@supermarche.be</value>
  </field>
</rule>
```

Il suffira à l'utilisateur qui souhaite malgré tout continuer à recevoir de telles requêtes d'ajouter à son document de règles la règle suivante :

```
<rule>
  <action>allow</action>
  <field>
    <type>Header</type>
    <name>From</name>
    <value>sip:promos@supermarche.be</value>
  </field>
</rule>
```

Notre système se chargera de tenir compte de la présence de ce "allow" dans la liste personnelle de l'utilisateur, et respectera le critère de cet utilisateur en lieu et place de celui de la liste commune.

Exemple 5 : Utilité de `<all/>`

L'élément "all" est un peu particulier, et mérite que l'on illustre son utilisation.

Premièrement, cet élément permet de ne recevoir aucun message (ce qui constitue une solution particulièrement radicale contre le SPAM !). Un document de la forme

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE exemple SYSTEM "rules-document.dtd" []>
<rules-document>
  <rule>
    <action>block</action>
    <all/>
  </rule>
</rules-document>
```

permet en effet de rejeter tous les messages.

Il existe un autre usage de "all" : l'utilisateur qui ne souhaiterait pas laisser échapper le moindre message, quitte à filtrer lui-même les messages non-sollicités, pourrait publier le document de règles suivant :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE exemple SYSTEM "rules-document.dtd" []>
<rules-document>
  <rule>
    <action>allow</action>
    <all/>
  </rule>
</rules-document>
```

Ce document permettrait, grâce à la présence du "all" et du mécanisme de surcharge inhérent à "allow", de recevoir tous les messages, quels que soient les autres critères de ce document de règles ou du document de règles commun.

Exemple 6 : Utilité de `<missing/>`

L'élément "missing" permet de définir une règle devant être déclenchée lorsqu'une requête entrante ne contient pas un certain *header*. Un exemple typique d'utilisation de cet élément est la détection de l'absence d'un "P-Asserted-Identity Header" [5].

En effet, certains champs d'une requête SIP peuvent être sans valeur dans le cadre d'un système de filtrage si leur véracité n'est pas garantie. L'exemple le plus probant est celui du champ "From". En effet, il serait inutile de tenter de filtrer des messages en se basant sur la valeur du champ "From" si celle-ci peut être modifiée par l'émetteur lors de chacun de ses envois.

Une première étape pour se prémunir de ce genre de désagréments est de vérifier la présence du "P-Asserted-Identity Header" dans les requêtes entrantes. Ceci peut être aisément mis en place grâce à la présence d'une règle de la forme :

```
<rule>
  <field>
    <type>Header</type>
    <name>P-Asserted-Identity</name>
    <missing/>
  </field>
</rule>
```

Une seconde étape consisterait bien sûr à vérifier la véracité de ce *header*, fonctionnalité qui pourrait être supportée plus tard par ce *framework* (grâce à ses interfaces). La détection de l'absence de "P-Asserted-Identity" constitue toutefois une première étape pour y parvenir.

[5] Le mécanisme de "P-Asserted-Identity est présenté dans la section 2.1.2 (page 4).

5.4 Architecture

5.4.1 Principaux composants de notre solution

SPAM Filter

Le “SPAM Filter” effectuera le lien entre la requête entrante et le déclenchement de la réaction attendue du système face à cette requête. Il sera chargé, lors du relais d’une requête, d’effectuer le “filtrage” permettant de relayer (ou ne pas relayer) les messages de la façon attendue par l’utilisateur.

Pour ce faire, il consultera le document de règles commun et celui de l’utilisateur à qui la requête est adressée. Il vérifiera que la requête ne correspond à aucun critère de filtrage, ni de la liste commune, ni de la liste personnelle du destinataire de la requête. Si la requête est identifiée comme SPAM, le “SPAM Filter” entreprendra l’action spécifiée dans le document de règles (par exemple mettre un *flag* et relayer, bloquer, . . .). Sinon, la requête sera relayée à son destinataire comme si rien ne s’était passé.

Le SPAM Filter consultera régulièrement le document de règles commun. Pour lui éviter d’avoir à effectuer une requête XCAP systématique (ce qui provoquerait une baisse des performances telle que notre système deviendrait inutilisable), il gardera le document en mémoire, et un thread se chargera, à intervalles réguliers, d’effectuer une requête XCAP pour récupérer la toute dernière version du document de règles commun, et ainsi de “rafraîchir” le document de règles présent en mémoire.

En bref, le SPAM Filter sera chargé de :
- Chercher une correspondance entre la requête courante et une des règles des documents de règles commun et du destinataire.
- Si une telle correspondance existe, appeler la méthode “doAction” ce qui permettra à l’implémentation de l’interface fournie avec le *framework* d’adopter le comportement attendu, spécifique pour une action et/ou un protocole donné.

SPAM Detector

Le composant “SPAM Detector” aura le rôle d’effectuer une détection automatique du SPAM. Cela signifie que même en l’absence de critères dans un quelconque document de règles, il sera capable d’identifier des requêtes supposées non sollicitées. Sa tâche se déroulera donc toujours en deux étapes :

1. Identifier des requêtes comme étant non sollicitées
2. Ajouter les règles dans le document de règles commun pour protéger les futurs destinataires

Quand le SPAM Detector ajoutera des règles au document de règles commun, il spécifiera, comme nouvelle action à entreprendre contre cette nouvelle forme de SPAM, l’action par défaut présente dans le document de règles commun. Ceci signifie, du point de vue du document de règles, que l’élément “action” de la nouvelle règle contiendra l’action présente dans l’élément “default-action” du document de règles commun. Pour éviter au SPAM Detector d’effectuer une requête XCAP lui permettant d’obtenir la “default-action” lors de toute nouvelle détection de SPAM, le comportement adopté sera le même que celui spécifié dans le point précédent : un thread se chargera d’obtenir, à intervalles réguliers, la toute dernière version du document de règles commun.

SPAM Detector Backend

Le “SPAM Detector Backend” sera (comme son nom l’indique) une application qui s’exécutera automatiquement et en tâche de fond, à intervalles réguliers. Son rôle sera de chercher à identifier des règles présentes dans suffisamment de documents personnels pour prendre la décision de les migrer vers le document de règles commun. L’intérêt de cette migration est double :

1. Éviter d’alourdir les listes personnelles avec des règles redondantes.
2. Protéger tous les utilisateurs de requêtes majoritairement reconnues comme étant du SPAM.

Comment le “SPAM Detector Backend” prendra-t-il la décision de migrer certaines informations de listes personnelles vers la liste commune ? Il va parcourir les règles personnelles et regarder dans combien de listes chaque règle est présente. Connaissant le nombre total de listes personnelles, l’application pourra en déduire un “pourcentage de listes contenant chaque règle”, c’est-à-dire un “taux de fréquentation” de chaque règle dans les listes. Si ce pourcentage est suffisamment élevé (il s’agira en fait d’un paramètre configurable), l’application prendra la décision de migration spécifiée ci-dessus.

XCAP Server (et XCAP AntiSPAM Usage)

Le serveur XCAP sera celui qui est actuellement utilisé par le serveur de Presence. Il dialoguera avec une base de données contenant les documents de règles (ce qui garantit la persistance des données), et interprétera les requêtes XCAP.

Le “XCAP AntiSPAM Usage” permettra d’offrir quelques services supplémentaires :
- Mécanisme de contrôle d’accès, garantissant que l’émetteur d’une requête XCAP accède bien à des ressources auxquelles il est autorisé.
- Système de quotas, permettant de contrôler la taille maximale du document de règles de chacun des utilisateurs du système. Ceci permet d’éviter de surcharger le système avec des utilisateurs possédant des listes extrêmement longues, et même d’offrir à des clients la possibilité d’avoir des listes plus longues (et donc une meilleure protection) selon leur type de contrat.

Admin GUI

Notre solution proposera une interface d’administration. Elle lui permettra d’intervenir sur tous les documents de règles, et de configurer
- le “SPAM Detector” (méthodes de détection automatique du SPAM, . . .)
- le “SPAM Filter” (fréquence de rafraîchissement des documents[6], . . .)
- le “SPAM Detector Backend” (seuil à partir duquel la migration peut être entreprise, intervalles de lancement, . . .)
- le “XCAP AntiSPAM Usage” (quota des utilisateurs, contrôle d’accès, . . .)

Elle permettra également de paramétrer le système pour spécialiser des actions existantes ou ajouter des nouvelles actions, comme décrit dans la section 5.3.4. Elle offrira une propriété “Classe implémentant l’interface AntiSPAMFramework”, dont la valeur initiale sera la classe de base livrée avec le *framework* (“AntiSPAMFrameworkImpl”) et implémentant les comportements face au SPAM livrés par la plate-forme.

[6] cfr section 5.4.1

5.4.2 Questions sur le fonctionnement des listes

Comment fonctionnera la "dualité" entre le document commun et les documents personnels ?

- Lorsque le "SPAM Detector" identifiera automatiquement un SPAM, elle ajoutera les critères permettant de placer ce type de requêtes au niveau d'une *blacklist* commune.
- Lorsque l'utilisateur ajoutera une règle lui permettant d'ajouter certaines requêtes à son document de règles, cet ajout sera effectué au niveau de sa liste personnelle. Ceci permettra d'éviter de laisser un utilisateur poser des choix communs, qui pourraient être jugés "abusifs" par d'autres utilisateurs[7].

  Par contre, le "SPAM Detector Backend" pourra, si une certaine règle s'avère suffisamment fréquente dans les listes personnelles, prendre la décision de migrer cette règle personnelle vers une liste commune, comme spécifié dans la section 5.4.1.

Comment le SPAM Filter détermine-t-il le document de règles personnel à consulter ?

En effet, le SPAM Filter doit consulter le document de règles du destinataire, et il a donc besoin de déterminer quel document il doit consulter, ou l'emplacement où l'obtenir.

Pour les "initial requests" ("INVITE", "MESSAGE", ...), la partie "username" de l'URI mentionnée dans le champ "To" fournira le nom d'utilisateur dont le SPAM Filter doit consulter le document (et donc, en cas de requêtes XCAP, l'emplacement du document à obtenir).

Pour la requête particulière "REGISTER" (dont le champ "From" et le champ "To" contiennent l'URI concernée par ce "REGISTER"), une extension de cette spécification pourra être proposée, permettant au registrar de posséder un document de permissions, bloquant les requêtes "REGISTER" provenant de certaines URI[8].

[7] En effet, nous acceptons tous certains messages publicitaires, mais nous n'acceptons pas tous les mêmes. On ne peut donc pas accepter un système qui les rejetterait tous en bloc.

[8] Notez toutefois que des requêtes adressées (même abusivement) à un registrar ne constituent pas du SPAM à proprement parler, mais plutôt du DoS. Intégrer cette fonctionnalité dans cette spécification éloignerait donc le *framework* de ses objectifs initiaux

Globalement, la figure 5.1 permet de fixer définitivement les idées sur “qui lit/écrit (sens des flèches) sur quelle ressource”.

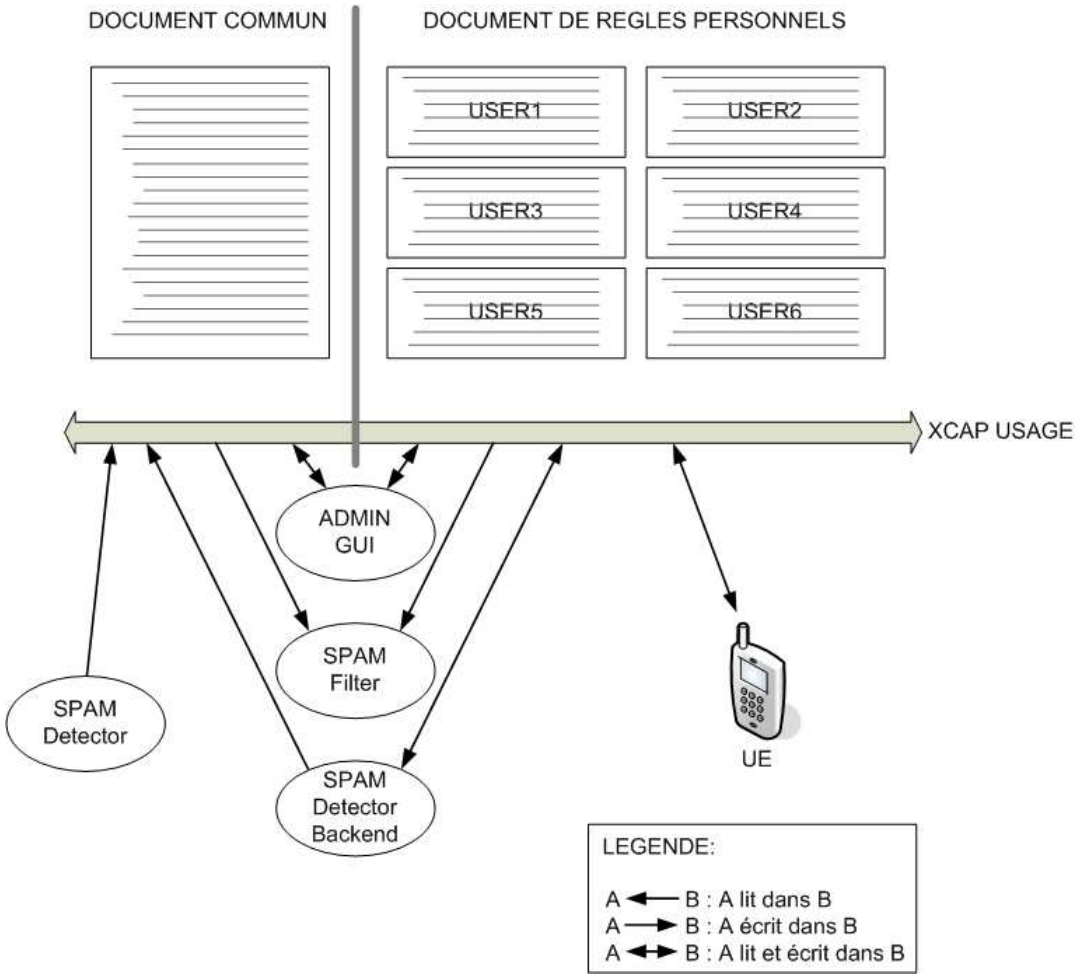


FIG. 5.1 – Interactions autour des documents de règles

5.4.3 Scénario général

On présente ici un scénario général représentant le cas de l'arrivée d'un message destiné à un des utilisateurs de notre système. Avant de présenter le scénario, quelques remarques importantes doivent être formulées :

- Dans ce scénario, on souhaite garder une vision très "haut niveau" du système, et simplement mieux appréhender le rôle des différents intervenants de notre service "anti-SPAM". Pour cette raison, le parcours précis du message entre les différents *proxies* ne sera pas représenté ici.
- Pour fixer les idées, on suppose que la réaction par défaut du système est "block".
- On fournira un meilleur niveau de détail sur les opérations de chacun des intervenants dans les Use Cases.

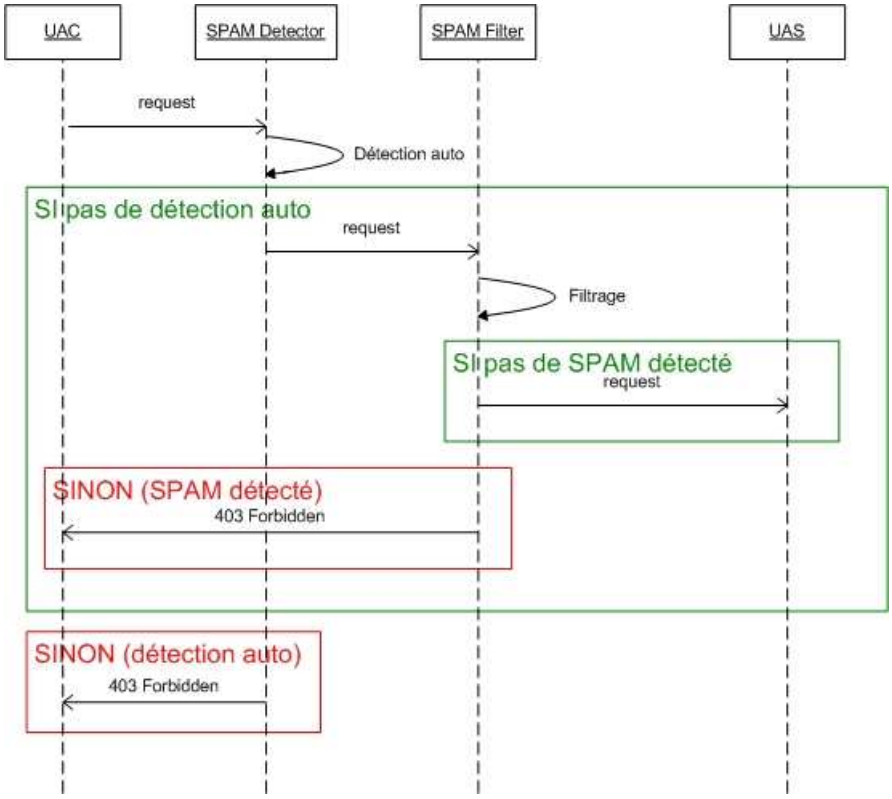


FIG. 5.2 – Callflow d'un scénario de message entrant

Arrivée d'un message **Description :** Arrivée d'un nouveau message adressé à un client de notre domaine **Précondition :** Le message ne correspond à aucun critère de filtrage présent ou à venir **Postcondition :** Le message est transmis à l'UAS			
UAC	**SPAM Detector**	**SPAM Filter**	**UAS**
1. Un UAC envoie un message adressé à l'UAS	2. Le SPAM Detector analyse le message pour tenter d'identifier un nouveau SPAM, ce qui n'est pas le cas 3. Le SPAM Detector transmet le message au SPAM Filter	4. Le SPAM Filter vérifie si le message correspond à un critère de filtrage existant, ce qui n'est pas le cas 5. Le SPAM Filter transmet le message à l'UAS	6. L'UAS reçoit le message

Arrivée d'un message correspondant à un critère de filtrage existant **Description :** Arrivée d'un nouveau message adressé à un client de notre domaine **Précondition :** Le message correspond à un critère de filtrage existant ! **Postcondition :** Le message n'est PAS transmis à l'UAS			
UAC	**SPAM Detector**	**SPAM Filter**	**UAS**
1. Un UAC envoie un message adressé à l'UAS 6. L'UAC reçoit le message d'erreur	2. Le SPAM Detector analyse le message pour tenter d'identifier un nouveau SPAM, ce qui n'est pas le cas 3. Le SPAM Detector transmet le message au SPAM Filter	4. Le SPAM Filter vérifie si le message correspond à un critère de filtrage existant, ce qui est le cas ! 5. Le SPAM Filter répond à l'UAC par un "403 Forbidden"	

Arrivée d'un message nouvellement identifié comme SPAM **Description :** Arrivée d'un nouveau message adressé à un client de notre domaine **Précondition :** Le message ne correspond à aucun critère de filtrage existant, mais est identifié comme SPAM par le SPAM Detector **Postcondition :** Le message n'est PAS transmis à l'UAS			
UAC	**SPAM Detector**	**SPAM Filter**	**UAS**
1. Un UAC envoie un message adressé à l'UAS			
	2. Le SPAM Detector analyse le message pour tenter d'identifier un nouveau SPAM, ce qui est le cas ! 3. Le SPAM Detector répond à l'UAC par un "403 Forbidden"		
4. L'UAC reçoit le message d'erreur			

5.4.4 La localisation des services et les méthodes d'interaction

Après avoir présenté les différents composants et illustré la manière dont ils interopèrent, il est intéressant de se pencher sur la question de la localisation de ces différents composants et de la manière dont ils interagissent dans le système, de manière à le rendre le plus efficace possible.

La localisation du SPAM Detector et du SPAM Filter

Pour limiter le trafic "inutile" dans notre domaine, il faut tenter d'identifier et d'arrêter le relais des requêtes identifiées comme SPAM le plus tôt possible dans leur route. Il faut donc faire intervenir le SPAM Detector et le SPAM Filter le plus tôt possible dans la chaîne de *proxies* que va parcourir la requête avant d'atteindre sa cible. Pour ce faire, il convient donc de distinguer deux cas :

Si le message provient de notre réseau : les composants SPAM Detector et SPAM Filter devraient idéalement être situés dans le premier *proxy* rencontré par une requête SIP (dans l'IMS, au niveau du P-CSCF).

Si le message provient d'un autre réseau : les composants SPAM Detector et SPAM Filter devraient idéalement être situés dans le premier *proxy* que nous administrons et qui interceptera la requête SIP entrante (dans l'IMS, au niveau du I-CSCF).

Si la localisation proposée pour ces composants était problématique, une solution alternative consisterait à les placer au niveau de l'AS. Cette solution serait moins optimale, mais également acceptable.

Comment le "User Equipment" (UE) émet-il des requêtes XCAP ?

Le *framework* permet à l'UE d'intervenir de sa propre initiative sur son document de règles personnel, notamment pour ajouter/modifier/supprimer ses règles de filtrage. Comment cet UE émet-il des requêtes XCAP ? Il émettra de telles requêtes via des *proxies* HTTP. L'OMA a à ce titre spécifié un premier serveur HTTP dans la file de *proxy* : l' "agregation proxy". C'est par cet intermédiaire que l'UE émettra ses requêtes XCAP. Dans l'IMS, le proxy jouant le rôle d' "agregation proxy" sera le P-CSCF, comme mentionné dans l'exemple de la section suivante.

Exemple de déploiement dans l'IMS

On propose ici un exemple de déploiement de notre solution dans l'IMS (figure 5.3). Comme précisé précédemment, la localisation des composants telle que représentée ici n'est pas obligatoire.

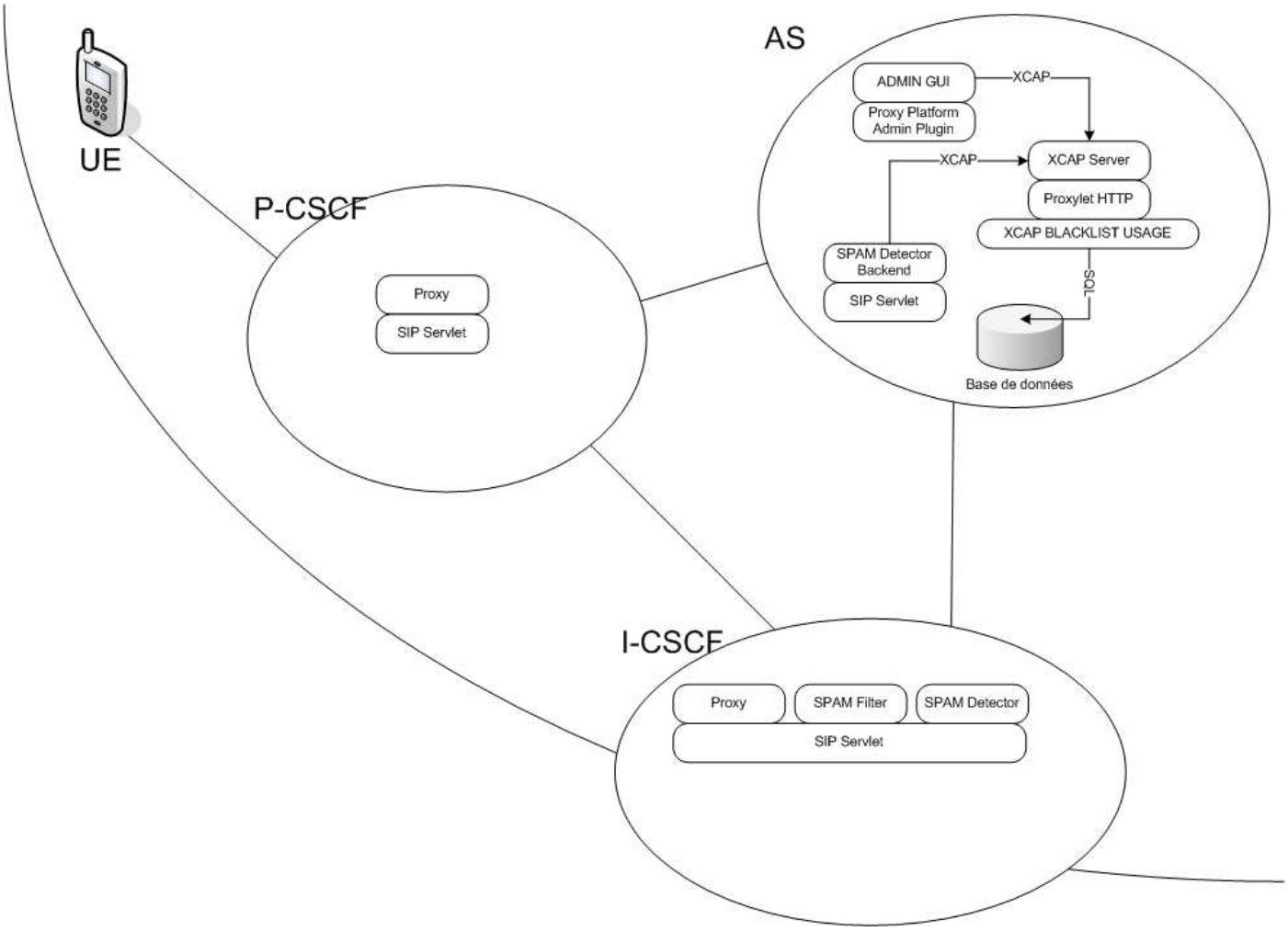


FIG. 5.3 – Exemple de déploiement dans l'IMS

5.5 Use Cases

Il est à présent temps de présenter un Use Case Diagram, permettant de structurer les différents acteurs, et les actions précises qu'ils sont autorisés à effectuer. Ensuite, on proposera un scénario détaillé de chacun des cas d'utilisation du système.

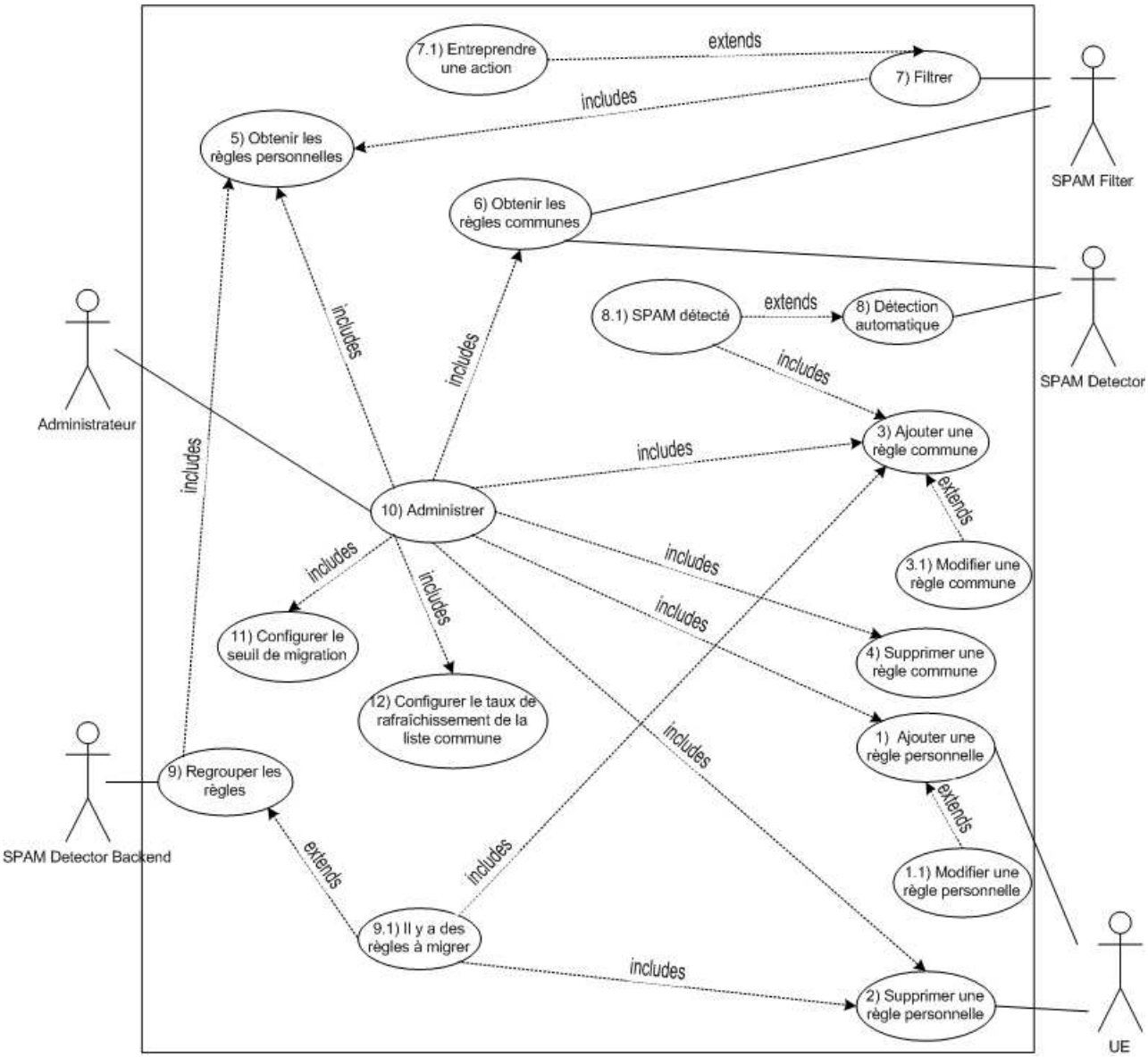


FIG. 5.4 – Use Case Diagram de l'architecture

Use Case 1 : Ajouter une règle personnelle

UC (1) : Ajouter une règle personnelle **Description :** L'UE souhaite ajouter une règle à son document de règles personnel **Précondition :** / **Postcondition :** La règle a été ajoutée au document de règles personnel de l'utilisateur	
UE	**Serveur XCAP**
1. L'UE émet une requête XCAP `PUT` avec la règle qu'il souhaite ajouter	
	2. Le serveur XCAP traite cette requête, et ajoute la règle au document de l'UE 3. Le serveur XCAP émet une réponse "201 Created" qu'il envoie à l'UE
4. L'UE reçoit la réponse favorable à sa requête	

Cas alternatif 1.1 : Modifier une règle personnelle

UC (1.1) : Modifier une règle personnelle **Description :** L'UE souhaite modifier une règle existante dans son document de règles personnel **Précondition :** / **Postcondition :** La règle a été modifiée dans le document de règles personnel de l'utilisateur	
UE	**Serveur XCAP**
1. L'UE émet une requête XCAP `PUT` avec la règle qu'il souhaite modifier	
	2. Le serveur XCAP traite cette requête, et modifie la règle au document de l'UE 3. Le serveur XCAP émet une réponse "201 Modified" qu'il envoie à l'UE
4. L'UE reçoit la réponse favorable à sa requête	

Use Case 2 : Supprimer une règle personnelle

UC (2) : Supprimer une règle personnelle **Description :** L'UE souhaite supprimer une règle de son document de règles personnel **Précondition :** / **Postcondition :** La règle a été supprimée du document de règles personnel de l'utilisateur	
UE	**Serveur XCAP**
1. L'UE émet une requête XCAP `DEL` avec la règle qu'il souhaite supprimer	
	2. Le serveur XCAP traite cette requête, et supprime la règle du document de l'UE 3. Le serveur XCAP émet une réponse "201 Deleted" qu'il envoie à l'UE
4. L'UE reçoit la réponse favorable à sa requête	

Use Case 3 : Ajouter une règle commune

UC (3) : Ajouter une règle commune **Description :** Le client souhaite ajouter une règle au document de règles commun. Le client XCAP peut être le SPAM Detector ou l'administrateur **Précondition :** / **Postcondition :** La règle a été ajoutée au document de règles commun

Client	**Serveur XCAP**
1. Le client émet une requête XCAP PUT avec la règle qu'il souhaite ajouter	
	2. Le serveur XCAP traite cette requête, et ajoute la règle au document commun 3. Le serveur XCAP émet une réponse "201 Created" qu'il envoie au client
4. Le client reçoit la réponse favorable à sa requête	

Cas alternatif 3.1 : Modifier une règle commune

UC (3.1) : Modifier une règle commune **Description :** Le client souhaite modifier une règle dans le document de règles commun. Le client XCAP peut être le SPAM Detector ou l'administrateur **Précondition :** / **Postcondition :** La règle a été modifiée dans le document de règles commun

Client	**Serveur XCAP**
1. Le client émet une requête XCAP PUT avec la règle qu'il souhaite modifier	
	2. Le serveur XCAP traite cette requête, et modifie la règle au document commun 3. Le serveur XCAP émet une réponse "201 Modified" qu'il envoie au client
4. Le client reçoit la réponse favorable à sa requête	

Use Case 4 : Supprimer une règle commune

UC (4) : Supprimer une règle commune **Description :** Le client souhaite supprimer une règle du document de règles commun. Le client XCAP peut être le SPAM Detector ou l'administrateur **Précondition :** / **Postcondition :** La règle a été supprimée du document de règles commun

Client	**Serveur XCAP**
1. Le client émet une requête XCAP `DEL` avec la règle qu'il souhaite supprimer	
	2. Le serveur XCAP traite cette requête, et supprime la règle du document commun 3. Le serveur XCAP émet une réponse "201 Deleted" qu'il envoie au client
4. Le client reçoit la réponse favorable à sa requête	

Use Case 5 : Obtenir le document de règles personnel d'un utilisateur

UC (5) : Obtenir le document de règles personnel d'un utilisateur donné **Description :** Un utilisateur (ou l'administrateur) souhaite obtenir le document de règles personnel d'un utilisateur donné. Le client peut être l'UE lui-même ou le SPAM Filter **Précondition :** / **Postcondition :** Le client a obtenu le document de règles personnel de l'utilisateur donné

Client	**Serveur XCAP**
1. Le client émet une requête XCAP `GET` pour obtenir le document de règles personnel d'un utilisateur donné	
	2. Le serveur XCAP traite cette requête, et extrait le document de règles de cet utilisateur, via la base de données 3. Le serveur XCAP émet une réponse "200 OK" qu'il envoie au client, et envoie également le document demandé
4. Le client reçoit la réponse favorable à sa requête, et le document demandé	

Use Case 6 : Obtenir le document de règles commun

UC (6) : Obtenir le document de règles commun **Description :** Un utilisateur (ou l'administrateur) souhaite obtenir le document de règles commun. Le client peut être l'administrateur ou le SPAM Filter **Précondition :** / **Postcondition :** Le client a obtenu le document de règles commun

Client	**Serveur XCAP**
1. Le client émet une requête XCAP `GET` pour obtenir le document de règles commun	
	2. Le serveur XCAP traite cette requête, et extrait le document de règles commun, via la base de données 3. Le serveur XCAP émet une réponse "200 OK" qu'il envoie au client, et envoie également le document demandé
4. Le client reçoit la réponse favorable à sa requête, et le document demandé	

Use Case 7 : Filtrer

UC (7) : Filtrer **Description :** Le SPAM Filter veut savoir si la requête actuelle correspond ou non à un ou plusieurs critères existants dans la liste commune ou dans celle du destinataire de cette requête **Précondition :** La requête n'est pas du SPAM **Postcondition :** La requête continue sa route vers son destinataire
SPAM Filter
«include UC(5) : Obtenir le document de règles personnel d'un utilisateur» *«include UC(6) : Obtenir le document de règles commun»* 1. Le SPAM Filter reçoit une nouvelle requête 2. Pour chaque règle du document de règles personnel, le SPAM Filter vérifie si la requête courante correspond à cette règle. 3. Pour chaque règle du document de règles commun, le SPAM Filter vérifie si la requête courante correspond à cette règle. 4. La requête courante ne correspondant à aucun critère, le SPAM Filter relaie la requête

Cas alternatif 7.1 : Entreprendre action

UC (7.1) : Entreprendre action **Description :** La requête courante correspond à un critère d'un document de règles **Précondition :** La requête courante correspond à un critère d'un document de règles **Postcondition :** L'action mentionnée dans le critère correspondant à cette règle a été entreprise. Sinon, l'action par défaut a été entreprise
SPAM Filter
«le début du scénario est identique au cas normal» 4. Le SPAM Filter regarde l'action à entreprendre mentionnée dans le champ "action" de la règle correspondant à la requête courante. Si l'élément "action" est absent de cette règle (puisque cet élément est facultatif), le SPAM Filter regarde l'action à entreprendre dans le champ "default-action" du document contenant cette règle. 5. Le SPAM Filter entreprend l'action consultée dans le point précédent.

Use Case 8 : Détection automatique

UC (8) : Détection automatique **Description :** Le SPAM Detector tente de détecter automatiquement toute nouvelle forme de SPAM **Précondition :** Le message n'est pas un SPAM **Postcondition :** Le message continue sa route vers son destinataire
SPAM Detector
1. Le SPAM Detector reçoit une nouvelle requête 2. Le SPAM Detector entreprend une analyse de la requête entrante pour tenter d'identifier, le cas échéant, toute nouvelle forme de SPAM. 3. La requête ne correspond à aucun critère de SPAM. Le SPAM Detector relaie cette requête vers son destinataire

Cas alternatif 8.1 : SPAM détecté

UC (8.1) : SPAM détecté **Description :** Le SPAM Detector a identifié une nouvelle forme de SPAM dans la requête courante **Précondition :** Le message est un SPAM **Postcondition :** Le message est intercepté. L'action par défaut est entreprise.	
SPAM Detector	**Serveur XCAP**
«le début du scénario est identique au cas normal»	
3. Le SPAM Detector consulte l'action par défaut, mentionnée dans l'élément "default-action" du document de règles commun. Cette action sera utilisée pour peupler le champ "action" de la nouvelle règle à ajouter au document commun	
«include UC(3) : Ajouter une règle commune»	

Use Case 9 : Regrouper les règles

UC (9) : Regrouper les règles **Description :** Tenter de trouver des règles suffisamment fréquentes dans les documents de règles personnels pour les migrer vers le document de règles commun **Précondition :** Aucune règle ne doit être migrée vers le document de règles commun **Postcondition :** Aucune règle n'a été migrée vers le document de règles commun
SPAM Detector Backend
«Pour chaque utilisateur du système :» *«include UC(5) : Obtenir le document de règles personnel d'un utilisateur»* 1. Le SPAM Detector Backend parcourt les règles personnelles, en appliquant l'algorithme suivant : `Pour chaque règle :` `  Si il s'agit d'une nouvelle règle` `  Alors créer un compteur lié à cette règle, initialisé à 0` `  Sinon incrémenter le compteur de cette règle` 2. Connaissant le nombre total d'utilisateurs, le SPAM Detector Backend divise le compteur de chaque règle identique par ce nombre, pour obtenir le taux de fréquentation de chaque règle dans les document de règles personnels. 3. Le SPAM Detector Backend compare le taux de fréquentation de chaque règle au "seuil de migration". 4. Aucune règle ne doit être migrée.

Cas Alternatif 9.1 : Il y a des règles à migrer !

UC (9.1) : Il y a des règles à migrer ! **Description :** Certaines règles ont un taux de fréquentation supérieur au "seuil de migration" **Précondition :** Certaines règles doivent être migrées vers le document de règles commun **Postcondition :** Les règles dont la fréquence est supérieure au "seuil de migration" ont été migrées vers le document de règles commun
SPAM Detector Backend
«le début du scénario est identique au cas normal» 4. Pour chaque règle dont le taux de fréquentation est supérieur au "seuil de migration", le SPAM Detector Backend ajoute cette règle au document de règles commun : *«include UC(3) : Ajouter une règle commune»* 5. Le SPAM Detector Backend supprime toutes les occurences de cette règle dans les documents de règles personnels : *«include UC(2) : Supprimer une règle personnelle»*

Use Case 10 : Administrer le système

UC (10) : Administrer le système **Description :** Ce Use Case permet à un administrateur de configurer et d'administrer le système **Précondition :** / **Postcondition :** L'administrateur a modifié les éléments de configuration souhaités

Administrateur	**Système**
1. L'administrateur signale qu'il souhaite accéder à la configuration du système	
	2. Le système demande à l'administrateur sur quel élément de la configuration il souhaite agir : Ajout/Edition/Suppression de règles dans un document de règles personnel, Ajout/Edition/Suppression de règles dans le document de règles commun, ou Configuration du "seuil de migration"
3. L'administrateur effectue son choix	
«Selon le choix de l'administrateur :»	
«include UC(1) : Ajouter une règle personnelle» *«include UC(2) : Supprimer une règle personnelle»* *«include UC(3) : Ajouter une règle commune»* *«include UC(4) : Supprimer une règle commune»* *«include UC(5) : Obtenir le document de règles personnel d'un utilisateur»* *«include UC(6) : Obtenir le document de règles commun»* *«include UC(11) : Configurer le "seuil de migration"»* *«include UC(12) : Configurer le taux de rafraîchissement du document commun»*	

Use Case 11 : Configurer le "seuil de migration"

UC (11) : Configurer le "seuil de migration" **Description :** Configurer le seuil à partir duquel une règle personnelle sera jugée "suffisamment fréquente" pour être migrée vers le document de règles commun **Précondition :** / **Postcondition :** Le "seuil de migration" a été configuré

Administrateur	**Système**
1. L'administrateur signale qu'il souhaite configurer le "seuil de migration"	
	2. Le système demande à l'administrateur d'entrer une nouvelle valeur pour ce paramètre
3. L'administrateur entre la nouvelle valeur	
	4. Le système confirme la prise en compte de ce nouveau paramètre

Use Case 12 : Configurer le taux de rafraîchissement du document commun

UC(12) : Configurer le taux de rafraîchissement du document commun **Description :** Configurer la fréquence à laquelle le SPAM Detector et le SPAM Filter effectueront une requête XCAP pour obtenir la toute dernière version de la liste commune **Précondition :** / **Postcondition :** Le taux de rafraîchissement a été configuré

Administrateur	**Système**
1. L'administrateur signale qu'il souhaite configurer le taux de rafraîchissement du document commun	
	2. Le système demande à l'administrateur d'entrer une nouvelle valeur pour ce paramètre
3. L'administrateur entre la nouvelle valeur	
	4. Le système confirme la prise en compte de ce nouveau paramètre

5.6 Fonctionnalités supplémentaires

5.6.1 Hiérarchisation des documents de règles

La solution telle que présentée ici offre deux niveaux de documents de règles : le document de règles commun et celui des utilisateurs. Dans ce contexte, des services de migration d'un document de règles personnel vers le document de règles commun sont proposés. Il faut toutefois remarquer que XCAP pourrait permettre, si le besoin s'en faisait sentir, de généraliser ce comportement à n niveaux de documents (avec possibilité de migrations d'une liste à une autre)... Ceci à très faible coût.

On pourrait par exemple imaginer, au niveau le plus bas, les documents de règles des utilisateurs. Ensuite viendraient des documents de règles qui auraient autorité sur un sous-domaine donné. D'autres documents pourraient ensuite être valables pour un ensemble de sous-domaines. Enfin, au niveau le plus élevé, on trouverait le document de règles commun à tous les utilisateurs du domaine. Un exemple d'utilisation de cette fonctionnalité serait celui d'un opérateur multinational, qui dispose de règles valables pour l'ensemble de ces utilisateurs (niveau hiérarchique le plus élevé), de règles hiérarchiquement inférieures dépendant de politiques nationales ou régionales, elles-mêmes amendées par des règles personnelles (niveau hiérarchique le plus bas).

Dans ce type configuration, le SPAM Detector Backend pourrait prendre la décision de migrer des règles et de fusionner des morceaux de documents entre deux niveaux quelconques.

Cette généralisation de deux à n niveaux pourrait se faire à faible coût, moyennant les opérations suivantes :

1. Spécifier l'ordre dans lequel le SPAM Filter consulte les différents niveaux de listes pour filtrer le SPAM.
2. Affiner le contrôle d'accès de XCAP, pour définir qui est autorisé à intervenir sur quel(s) document(s).
3. Généraliser le fonctionnement de l'action "allow" à n niveaux.
4. Déterminer, lorsque le SPAM Detector identifie une nouvelle forme de SPAM, dans quel document celui-ci doit insérer une nouvelle règle.

5.7 Performances

Au cours du stage, le prototype présenté dans ce chapitre a été implémenté sous forme d'une servlet SIP spécialisant le comportement par défaut d'un *proxy* SIP classique, et a été déployée sur la "Proxy Platform" d'Alcatel[9]. Avant d'envisager une utilisation autre qu'expérimentale de ce service anti-SPAM, il convient de s'interroger sur son coût d'utilisation en terme de ressources.

Il est évident que l'ajout d'un quelconque service filtrant des requêtes SIP consomme davantage de ressources et ralentit le service initial (dans notre cas un simple relais de requêtes), mais il est important de montrer que cette consommation de ressources est "raisonnable", c'est-à-dire qu'il serait effectivement possible pour un client d'utiliser cette fonctionnalité supplémentaire sans pour autant devoir multiplier déraisonnablement son investissement en mémoire et en processeurs.

On notera que la mesure des performances du service anti-SPAM sera relative aux performances de la "Proxy Platform" d'Alcatel, et qu'elle ne pourra en aucun cas être considérée "dans l'absolu". En effet, la servlet anti-SPAM étant déployée sur cette plateforme, on ne pourra mesurer ses performances que par rapport aux performances d'un *proxy* "de base" n'offrant pas le service de filtrage de requêtes.

Quelle sera l'unité de mesure adoptée ? Dans notre cas, il s'agira du nombre de tentatives d'appels par seconde, soit de "Call Attempts Per Second" (CAPS). A un "Call Attempt" (CA) correspond le relais par le *proxy* des requêtes d'un appel SIP complet, de la tentative d'appel à la fin de la communication. Les requêtes correspondant à un "Call Attempt" sont schématisées parla figure 5.5.

[9] Présentée notamment dans [20]

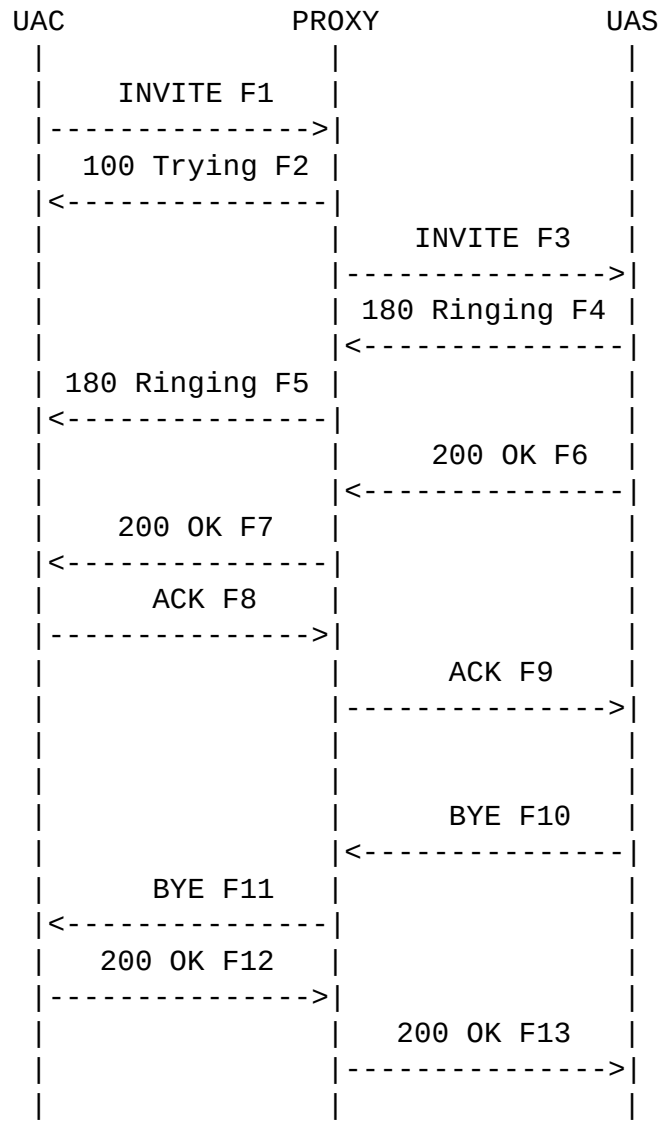


FIG. 5.5 – Les 13 requêtes correspondant à un “Call Attempt”

Le nombre de CAPS que le *proxy* est capable de traiter est la mesure déjà utilisée par Alcatel lors des bancs d’essais sur leurs produits. Pour cette raison, c’est cette mesure qui sera choisie pour la mesure des performances du service anti-SPAM. Il s’agira donc d’injecter une quantité croissante de CA dans le *proxy*, et de mesurer la quantité maximale de CAPS que le *proxy* est capable de relayer sans “prendre du retard” (globalement, c’est le nombre de retransmissions nécessaires qui permettra de constater le bon fonctionnement du *proxy*). Pour injecter une quantité croissante de CA dans le *proxy*, on utilisera l’outil “SIPp”[21]. “SIPp” est un générateur de trafic SIP supportant divers scénarios de tests, et notamment une mesure des CAPS traités par un *proxy*.

Il faut préciser que “SIPp” injecte du trafic en quantité variable, mais que le scénario utilisé ne fait intervenir qu’un seul destinataire. Bien sûr, dans un cas réel, les requêtes relayées par un *proxy* sont adressées à un grand nombre de destinataires différents. En se basant sur l’hypothèse que tous les documents de règles de tous les destinataires sont déjà présents en mémoire, les performances seraient tout à fait équivalentes à notre “cas d’école”, puisque un accès (en mémoire) à un document ou à un autre nécessite le même temps d’accès.

Cette hypothèse consistant à considérer que tous les documents de règles nécessaires sont déjà présents en mémoire est bien sûr optimiste. Dans un cas réel, où tous les documents de règles ne sont pas forcément déjà chargés en mémoire, le service anti-SPAM doit alors effectuer des requêtes XCAP pour les récupérer. Ceci a une incidence sur les performances du service, puisqu’un certain pourcentage des requêtes à relayer nécessite l’envoi d’une requête XCAP. Compte tenu des plus faibles performances du serveur XCAP (estimées à une cinquantaine de requêtes par seconde par l’équipe Alcatel), celui-ci devient donc un goulot d’étranglement sur cette proportion de requêtes. Ne disposant pas des infrastructures nécessaires pour réaliser des mesures plus poussées, cette section s’attardera uniquement sur les performances du *proxy* pour des accès à des documents de règles présents en mémoire.

On procèdera aux mesures de performances du service anti-SPAM en deux temps :

1. Mesures sur un *proxy* “de base” (service anti-SPAM désactivé).
2. Mesures sur un *proxy* supportant le service anti-SPAM, avec un nombre de règles croissant dans le document de règles commun [10].

   Les règles introduites dans le document de règles commun seront de la forme :

   ```
   <rule>
     <action>block</action>
     <field>
       <type>Header</type>
       <name>From</name>
       <value>sip:UNUSED@spammer.com</value>
     </field>
   </rule>
   ```

   Aucune règle ne correspondant à aucune requête SIP relayée par le *proxy*, ceci permettra de garantir que l’entièreté des règles sont parcourues pour toute requête SIP relayée.

   On précisera également que comme spécifié dans ce chapitre, une requête XCAP ne sera pas effectuée par le *proxy* pour chaque requête à relayer, sans quoi ses performances s’avéreraient catastrophiques. Pour effectuer ces mesures de performances, les documents de règles sont automatiquement “rafraîchis” toutes les 10 secondes par un *thread*. Ce délai semble être un compromis correct entre les performances du service et son interactivité.

[10] Pour effectuer ces mesures, on maintiendra tous les documents de règles personnels vides.

Service anti-SPAM	Nombre de règles	CAPS	Baisse de performance (%)
Désactivé		220	
Activé	0	220	0%
Activé	1	213	3.18%
Activé	5	213	3.18%
Activé	10	213	3.18%
Activé	20	212	3.63%
Activé	50	211	4.09%
Activé	100	209	5%
Activé	1000	200	9.09%
Activé	2000	191	13.18%
Activé	3000	180	18.18%
Activé	4000	171	22.27%
Activé	5000	158	28.18%

TAB. 5.4 – Mesure des performances du service anti-SPAM

Les mesures de performances ont été effectuées sur une machine de référence disposant d'un processeur de type "Intel Pentium Centrino" cadencé à 2Ghz et de 1Go de mémoire vive. Il sera important de réfléchir aux incidences d'une variation de la puissance de calcul et de la mémoire vive disponible sur les performances du service. En effet, le "débit" de requêtes que le *proxy* est capable de traiter sera limité par la disponibilité de ces deux ressources (mémoire et processeur). De plus, si le nombre de règles présent dans le document de règles est élevé, la quantité de mémoire vive utilisée sera importante. Pour cette raison, les résultats pour un très grand nombre de règles, nécessitant plus de mémoire vive que disponible, devront être extrapolés à partir des performances effectivement mesurées.

Les premières mesures de performances ont fourni les résultats présentés dans le tableau 5.4.

La figure 5.6 donne une représentation graphique de ces résultats :

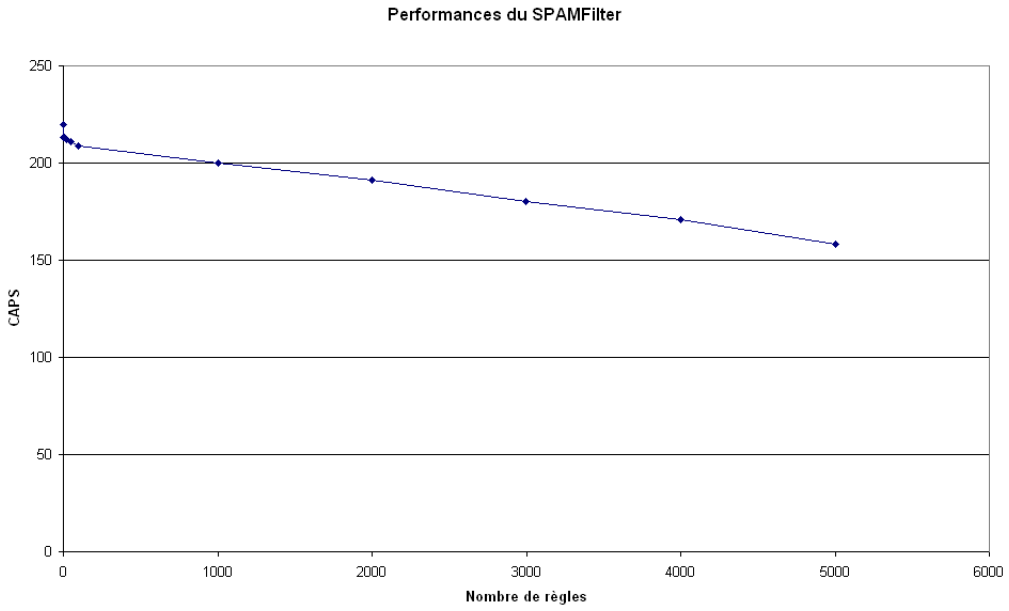


FIG. 5.6 – Performances du service anti-SPAM (pour 0 à 5000 règles)

Pour améliorer la lisibilité de ce graphique, on propose également une version (figure 5.7) davantage centrée sur sa partie gauche, c'est-à-dire où nous nous penchons uniquement sur les cas où le nombre de règles de filtrage est faible (inférieur à 100) :

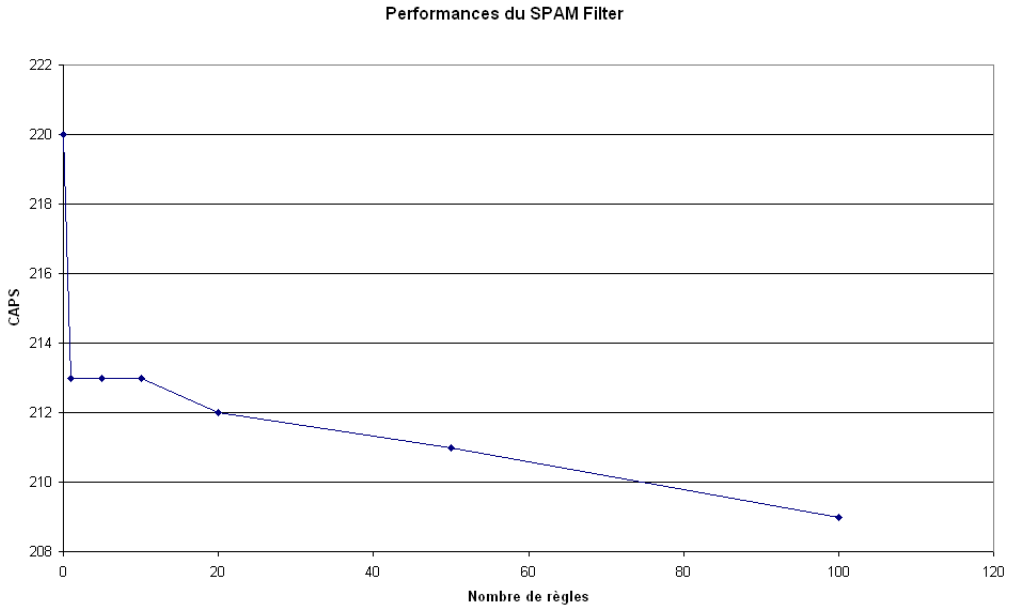


FIG. 5.7 – Performances du service : gros plan sur les faibles valeurs

A la lecture de ces graphiques, plusieurs constatations peuvent être faites :

1. Lorsqu'il n'y a pas de document de règle commun (et donc qu'il n'y a aucune règle de filtrage), les performances sont identiques à celles du *proxy* où le service anti-SPAM a été désactivé. En fait, la seule différence entre les deux cas est la présence du *thread* qui rafraîchit le document de règles commun toutes les 10 secondes. La présence de ce *thread* semble ne pas avoir d'incidence sur les performances du système.
2. On constate une chute brutale des performances dès qu'une première règle est présente dans le document de règles. Ceci s'explique par le fait que, pour la première fois, le *parsing* du document de règles (c'est-à-dire la conversion d'un document de règles textuel en objets Java stockés en mémoire) doit être effectué. Ce *parsing* n'était en effet pas réalisé dans le cas précédent.
3. Entre 1 et 20 règles, les performances sont quasiment constantes : il semble donc que le coût du parcours d'une règle en mémoire est presque négligeable face à celui du *parsing* décrit dans le point précédent.
4. A partir de 20 règles, la diminution des performances est presque linéaire. Pour les "grandes valeurs", on constate que la baise de performances du service est d'environ 10 CAPS par 1000 règles ajoutées.
   Attention : Au-delà de 5000 règles, la mémoire vive allouée à Java était saturée (cfr. paragraphe suivant). On ne peut donc extrapoler sur base de cette "linéarité des résultats" que si la mémoire vive disponible est suffisante.

Nombre de règles	CAPS
0	$= 220$
1000	$= 200$
2000	$= 191$
3000	$= 180$
4000	$= 171$
5000	$= 158$
6000	~ 150
7000	~ 140
8000	~ 130
9000	~ 120
10000	~ 110

TAB. 5.5 – Extrapolation des résultats à 10000 règles

A la lumière de cette mesure des performances, on peut donc donner une estimation des ressources qui seront nécessaires au fonctionnement du service anti-SPAM spécifié dans ce chapitre.

D'un point de vue de la consommation de mémoire vive, 400Mo avaient été alloués à la machine virtuelle Java sur laquelle le service anti-SPAM était déployé. Au delà d'environ 5500 règles, l'entièreté de la mémoire disponible était saturée. Lorsqu'un *proxy* tourne à plein régime et que le service anti-SPAM est désactivé, la mémoire utilisée est habituellement estimée à 300Mo. L'espace disponible pour stocker les règles est donc de 100Mo. On peut donc estimer que 100Mo de mémoire vive peuvent accueillir environ 5000 règles[11]. Cette estimation permet d'évaluer la mémoire vive à installer selon le nombre de règles maximal que l'on souhaite héberger.

Dans tous les cas, ce n'est probablement pas la mémoire vive qui constituera une barrière à la mise en place de notre service anti-SPAM. En effet, alors que dans le monde du courrier électronique (où, pour rappel, le nombre d'identités est illimité), le nombre de règles de filtrage d'un utilisateur dépasse très rarement les quelques centaines, ce nombre devrait dès lors être plus petit dans le contexte de l'IMS (où le nombre d'identités disponible devrait être limité). La possibilité d'héberger 5000 règles (et donc de satisfaire plusieurs dizaines utilisateurs) dans un espace aussi restreint que 100Mo de mémoire vive est donc un résultat positif de ce banc d'essai.

D'un point de vue de la puissance de calcul nécessaire (processeur), il a été constaté, après une baisse de performance "initiale" de 10 CAPS due à la conversion des règles textuelles en objets stockés en mémoire, une baisse moyenne de 10 CAPS pour 1000 règles ajoutées. A condition que la mémoire vive disponible soit suffisante, on peut donc extrapoler sur la diminution des performances du service :

[11] Puisque la mémoire s'est retrouvée saturée avec 5500 règles, la marge de 500 règles constituant une "réserve" pour les règles plus volumineuses (utilisant des opérateurs logiques, …)

Lorsque 10000 règles sont présentes dans le document de règles commun, les performances du *proxy* sont donc exactement divisées par 2. Il convient également de rappeler que même avec 10000 règles, les performances du *proxy* restent bien au-delà des performances du serveur XCAP, qui n'interfère donc pas sur les valeurs proposées ci-dessus. Ces résultats signifient que pour maintenir les performances du *proxy* alors que le service anti-SPAM est activé, si l'utilisateur souhaite héberger un maximum de 10000 règles, la puissance de calcul disponible doit être doublée. Plus généralement, on dira que si un utilisateur souhaite héberger $k * 10000$ règles, il lui faudra multiplier sa puissance de calcul par $k + 1$.

Pour conclure cette section, on notera une limitation importante relative à l'échelonnage du service : pour augmenter la puissance de calcul disponible, on peut remplacer le processeur par un processeur plus puissant, ou ajouter un nouveau processeur en plus de l'ancien. Toutefois, si les accès à plusieurs documents de règles peuvent bien être "distribués" sur plusieurs machines virtuelles (et donc sur plusieurs processeurs), l'accès à un seul document doit être géré par une seule machine virtuelle, et doit donc être géré par un seul processeur. Pour les documents de règles personnels, par définition nombreux et peu volumineux, aucun problème d'échelonnage ne sera rencontré. Pour le document de règles commun, unique et probablement très volumineux, une limite à son échelonnage existe bel et bien : cette limite sera fonction de la puissance de calcul maximale disponible dans un seul processeur.

Chapitre 6

Conclusion

L'objectif de ce travail était d'effectuer une étude sur la problématique du SPAM en SIP. En effet, ce protocole est appelé à connaître un développement considérable au cours des prochaines années, et il risque donc d'être visé par diverses formes d'excès, dont le SPAM. Il est dès lors indispensable de proposer des techniques visant à protéger ce nouveau moyen d'entrer en communication avant un déploiement planétaire de celui-ci, sans quoi il pourrait être trop tard.

Pour ce faire, un état de l'art fut tout d'abord réalisé. Après une étude des forces et des faiblesses de chacune des techniques identifiées, deux d'entre elles furent retenues pour une analyse approfondie, analyse pouvant donner lieu à la réalisation de deux prototypes : les "Computational Puzzles" et le "Consent Framework".

La technique des "Computational Puzzles" fut spécifiée et prototypée, et elle présenta d'assez bons résultats : elle permettait non seulement de s'adapter à la puissance de l'UE de chacun des interlocuteurs, mais était en plus présentée à l'IETF, ce qui rejoignait notre souci de respecter les standards. Malheureusement, l'identification de la capacité de calcul d'un interlocuteur posa problème : à l'heure actuelle, rien ne permet de déterminer de façon univoque la capacité de traitement d'un correspondant, ce qui empêche de fournir à celui-ci un puzzle de difficulté adaptée.

Le "Consent Framework" nécessita un examen approfondi : cette technique semblait en effet répondre à de nombreux besoins, mais dans de nombreux cas de figure, elle paraissait encore peu mûre et était présentée d'une façon assez confuse. Malgré l'envoi de nombreuses questions auprès de ses auteurs, aucune précision ne fut obtenue. Il fut dès lors impossible d'écrire une spécification rigoureuse de cette technique, et donc d'en réaliser un prototype.

Le besoin d'étudier une troisième technique se fit ressentir : en effet, après discussions avec de nombreuses personnes intéressées, il apparut qu'un système de *blacklists* était indispensable dans tout système anti-SPAM. Même si cette technique semblait nous éloigner du monde de la standardisation (puisqu'aucune *blacklist* n'était spécifiée à l'IETF), cette situation ne s'avéra finalement pas problématique, puisque notre technique n'imposait aucun comportement particulier chez les UA.

Plusieurs opportunités intéressantes se présentèrent alors pour améliorer notre technique de filtrage des requêtes. La première fut de fédérer ces *blacklists*, afin d'augmenter la puissance du système et donc de mieux protéger ses utilisateurs. Les idées suivantes provoquèrent la réalisation de nombreuses révisions de notre système, et nous amenèrent à la spécification d'un service anti-SPAM complet, générique et extensible. Une solution robuste pour combattre le SPAM semblait avoir été obtenue.

Mais une question importante présentée au début de ce travail n'était pas encore résolue : comment garantir l'identité de l'émetteur de toute requête SIP ? Dans ce cadre, SAML semblait être une solution très prometteuse. Des contacts avec Hannes Tschofenig, l'auteur de ce langage, permirent de mettre en évidence la complémentarité entre l'identification des expéditeurs et la lutte contre le SPAM. Pour améliorer l'intégration de notre format à l'existant, la redéfinition de ce format comme une extension de Geopriv fut également envisagée. Un format de document commun, composé d'un regroupement de notre DTD et de celle de SAML, a été soumis à l'IETF [19].

Quelques étapes ont donc été franchies au cours de ce travail dans le cadre de la lutte contre le SPAM en SIP : des techniques proposées à l'IETF ont été analysées, un prototype générique a été réalisé, et un nouveau format de document regroupant le filtrage des requêtes et l'identification de leur expéditeur a été soumis. Pour l'avenir, on propose de tenter d'améliorer l'unification du langage de descriptions de règles à celui utilisé dans Geopriv, afin d'obtenir un formalisme qui regrouperait le filtrage des requêtes indésirables, le transport d'informations relatives à la sécurité (SAML) et celui lié à la Presence et à la localisation des individus (Geopriv).

Bibliographie

[1] C. JENINGS. Computational Puzzles for SPAM Reduction in SIP, mars 2006. (draft-jennings-sip-hashcash-04).

[2] J. ROSENBERG, G. CAMARILLO, and D. WILLIS. A Framework for Consent-Based Communications in the Session Initiation Protocol (SIP), février 2006. (draft-ietf-sipping-consent-framework-04).

[3] J. PETERSON and C.JENINGS. Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP), octobre 2005. (draft-ietf-sip-identity-06).

[4] H. TSCHOFENIG. Using SAML for SIP, mars 2006. (draft-tschofenig-sip-saml-05).

[5] H. TSCHOFENIG. SPAM for Internet Telephony (SPIT) Prevention using the Security Assertion Markup Language (SAML), octobre 2005. (draft-schwartz-sipping-spit-saml-00).

[6] J. ROSENBERG and C.JENINGS. The Session Initiation Protocol (SIP) and SPAM, mars 2006. (draft-ietf-sipping-spam-02).

[7] SPAMCOP. What is the Spamcop Blocking List (SCBL)? http ://www.spamcop.net/fomserve/cache/297.html.

[8] J. ROSENBERG, G. CAMARILLO, and D.WILLIS. Requirements for Consent-Based Communications in the Session Initiation Protocol (SIP), avril 2006. (draft-ietf-sipping-consent-reqs-01).

[9] T. SHOWALTER. RFC 3028 : Sieve : A Mail Filtering Language, janvier 2001.

[10] J. ROSENBERG. The Extensible Markup Language (XML) Configuration Access Protocol (XCAP), février 2006. (draft-ietf-simple-xcap-06).

[11] J. ROSENBERG. Presence Authorization Rules, octobre 2005. (draft-ietf-simple-presence-rules-04).

[12] J. PETERSON. RFC 4079 : A Presence Architecture for the Distribution of GEOPRIV Location Objects, juillet 2005. (draft-ietf-geopriv-pres-02).

[13] H. SCHULZRINNE, J.MORRIS, H. TSCHOFENIG, J. CUELLAR, J. POLK, and J. ROSENBERG. ADocument Format for Expressing Privacy Preferences for Location Information, février 2006. (draft-ietf-geopriv-policy-08).

[14] CARNEGIE MELLON UNIVERSITY. The CAPTCHA Project. http ://www.captcha.net/.

[15] P2P.NET. AOL flips 'certified mail' switch. http ://p2pnet.net/story/8759/.

[16] R. EASTLAKE, D. JONES, and P. JONES. RFC 3174, US Secure Hash Algorithm 1 (SHA1), septembre 2001.

[17] J. ROSENBERG, H. SCHULZRINNE, G. CAMARILLO, A. JOHNSTON, J. PETERSON, R. SPARKS, M. HANDLEY, and E. SCHOOLER. RFC 3261 : SIP (Session Initiation Protocol), juin 2002.

[18] H. SCHULZRINNE, J. MORRIS, H. TSCHOFENIG, J. CUELLAR, J. POLK, and J. ROSENBERG. A Document Format for Expressing Privacy Preferences, octobre 2005. (draft-ietf-geopriv-common-policy-06).

[19] G. DAWIRS, T. FROMENT, and H. TSCHOFENIG. Authorization Policies for Preventing SPIT, février 2006. (draft-froment-sipping-spit-authz-policies-00).

[20] D. NOËL. Conception et Implémentation de Passerelles SMS-MMS-IMS. Master's thesis, University of Namur, juin 2005.

[21] O. JACQUES. SIPp. http ://sipp.sourceforge.net/.

[22] J. ROSENBERG. Extensible Markup Language (XML) Formats for Representing Resource Lists, février 2005. (draft-ietf-simple-xcap-list-usage-05).

Annexe A

Annexe A : “Authorization Policies for Preventing SPIT”

Cette annexe propose la dernière version du draft soumis à l’IETF, qui propose la spécification d’un format de documents de règles intégrant nos besoins spécifiés dans le chapitre 5 et les informations relatives à la sécurité véhiculées dans les documents SAML.

Network Working Group
Internet-Draft
Expires: August 24, 2006

G. Dawirs
University of Namur
T. Froment
Alcatel
H. Tschofenig
Siemens
February 20, 2006

Authorization Policies for Preventing SPIT
draft-froment-sipping-spit-authz-policies-00.txt

Status of this Memo

Copyright Notice

Abstract

SPAM, defined as sending unsolicited messages to someone in bulk, might be a problem on SIP open-wide deployed networks. The responsibility for filtering or blocking calls can belong to different elements in the call flow and may depend on various factors. This document discusses mechanisms to establish policies to react on potentially unwanted communication attempts.

These policies match a particular SIP communication pattern based on a number of attributes. The range of attributes includes information provided, for example, by the Session Initiation Protocol, SIP identity mechanism, SIP-SAML and SPIT-SAML.

An important topic for investigation is to decide whether the problem is worth analyzing, the choice of a policy language for describing authorization policies and to provide a mechanisms to create and modify authorization policies that are stored in XML documents.

Table of Contents

1. Introduction

The problem of SPAM for VoIP seems to become a very big challenge and only "the combination of several techniques can provide a framework for dealing with spam in SIP" (as stated in [I-D.jennings-sip-hashcash]).

One important building block is to have a mechanism to offer a mechanism to instruct some entities in the network to "filter" incoming requests according to user or to network-wide policies. Different entities, such as users or system administrators, might create and modify authorization policies and might even share these policies between domains.

Some attributes in a SIP communication play a more important role than others. For example, there is reason to believe that applying authorization policies based on the authenticated policies is an effective way to accept a communication attempt in order to compat SPIT. The same is true for policies that are applied to deployment friendlier SIP security solutions, such as the SIP identity mechanism [I-D.ietf-sip-identity].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Framework

The framework of the discussed anti-SPIT authorization policies looks as follows:

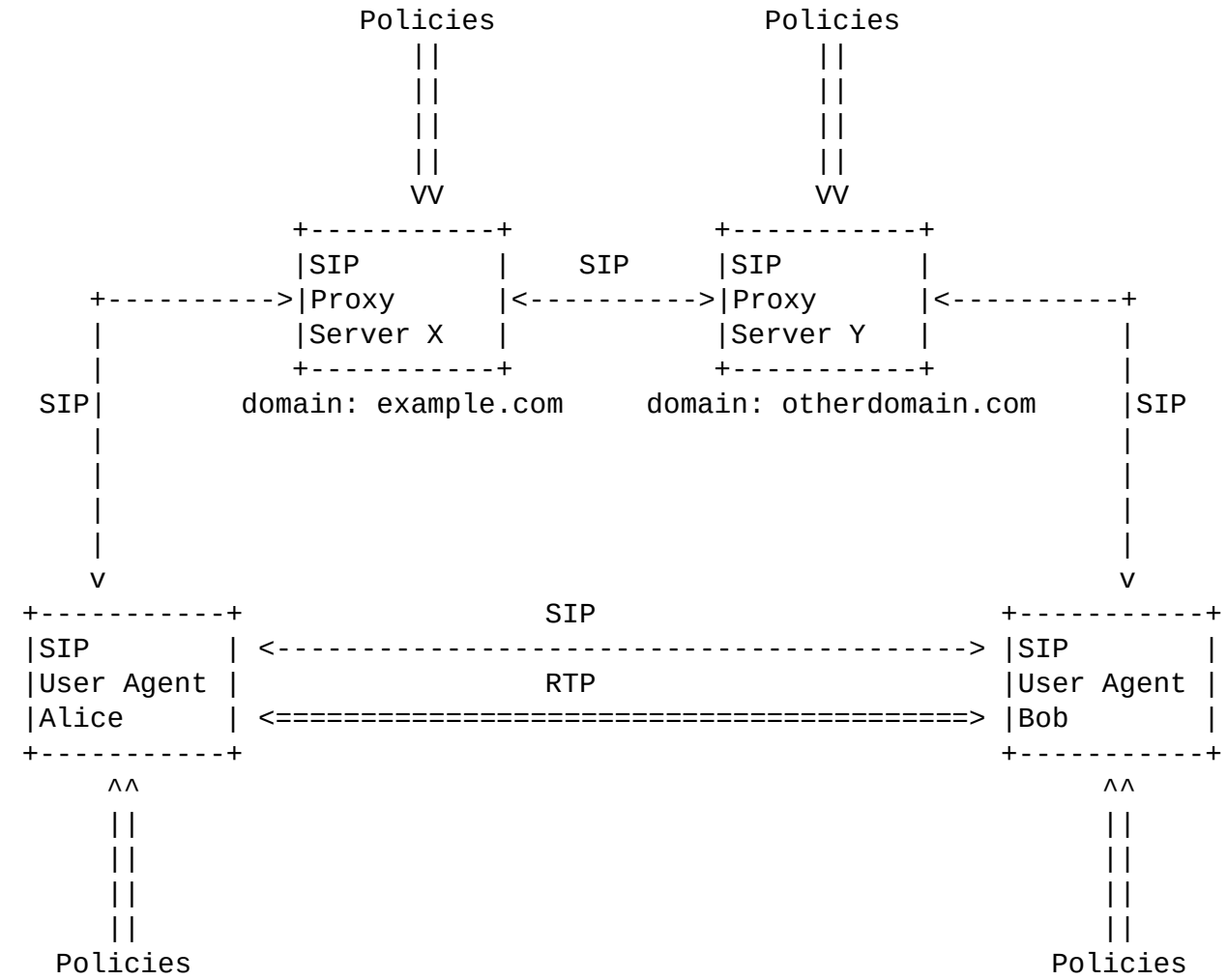
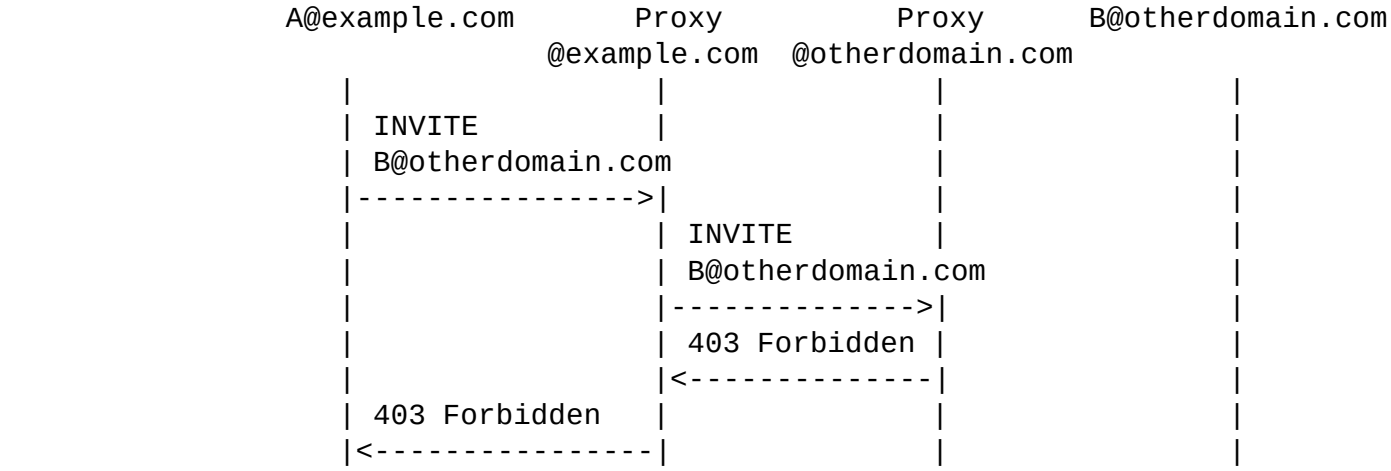


Figure 1: Framework

Authorization policies can be applied at the end host and/or by intermediaries. The rule maker might be an user that owns the end device, a VoIP service provider, a person with a relationship to the end user (e.g., the parents of a child using a mobile phone). The subsequent text lists a few use cases.

The first use case that can be imagined is the case of a user that asks its outbound proxy to offer protection of requests from a particular SIP UA. He can create an authorization policy rule and upload it to the SIP proxy within its own domain. Requests coming from this SIP URI will then be blocked or treated differently.

Supposing "B@otherdomain.com" has added an authorization rule blocking request coming from domain example.com. Here is a potential message sequence:

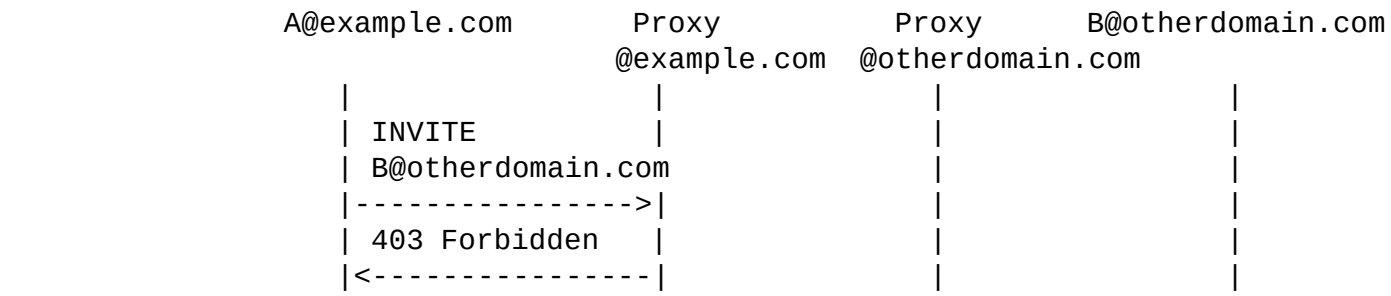


If a solution has to be provided to enable SPIT filtering then the following two sub-problems have to be solved:

- A authorization policy language that allows to expression the conditions and actions. An example is [I-D.ietf-simple-presence-rules].

- A mechanism to create, delete, update, retrieve and upload XML based authorization policy rules. XCAP [I-D.ietf-simple-xcap] is a possible solution.

In a more sophisticated setting one might even consider the following idea. Ideally, it would be good to stop unsolicited traffic as early as possible to avoid consuming bandwidth and processing power. Domains may agree to exchange authorization policies in order to stop SPIT earlier (i.e., closer to the source of the problem). The subsequent text describes this scenario.



This call flow illustrates the bandwidth-saving interest of this use case.

Though, two scenarios could happen:

- In the good case, the sender’s domain is honest and exchanges authorization policies in order to apply rules that avoids forwarding unsollicited requests.

- In the worst case, the sender’s domain is not cooperative. It will refuse to upload such documents. In this case, the presence of rules in the recipient’s domain will suffice to keep the recipient "SPAM free", even if more traffic has been consumed (since the request has been relayed at least until the first proxy of the recipient’s domain, exactly like in the first use case).

These two use cases illustrate the advantages of using a standard mechanisms in this framework.

It might be desirable to use a hierarchy of authorization policy documents that need to be combined when applying them to the SIP signaling traffic. This raises the question of a merging algorithm, particularly when authorization policy rules are conflicting or contain blacklists.

4. Requirements

The design of anti-SPIT authorization policies is guided by the following requirements. Note that this is a first strawman proposal that requires further discussions (since some requirements are potentially controversal).

1. The policies SHOULD allow filtering incoming requests depending on several criteria's:
    * Value of any SIP header attribute (e.g., From, To, Contact)
    * Method invoked by the caller (e.g., INVITE, MESSAGE)
    * Value of parameters specified in [I-D.schwartz-sipping-spit-saml]
        + IdentityStrength
        + CostOfCall
        + AuthenticationMethod
        + IdentityAssertion
        + ConnectionSecurity
        + SPITSuspected
        + CallCenter
        + AssertionStrength
    * Request URI of a request
    * Presence of a given expression in the body (subject for further investigation)
2. The policies SHOULD support wildcards (e.g., sip:*@example.com)
3. The policies SHOULD support logical operations (and, or, not) between individual elements in conditions
4. The policies SHOULD refer to all authenticated and unauthenticated identities.

5. The policies SHOULD allow the following actions to be specified:

   * "block": stop forwarding the request and answer with a ‘‘403 Forbidden’’

   * "polite-block": drop the request without answering anything

   * "mark": forward the request, putting a flag ‘‘SPAM’’

   * "allow": forward this message without conditions (this mechanism is described further)

   * and trigger other mechanism, such as:

     + "puzzle": trigger the "Computational Puzzles" [I-D.jennings-sip-hashcash] mechanism.

     + "consent": trigger the "Consent Framework" [I-D.rosenberg-sipping-consent-framework] mechanism

6. The policies SHOULD allow a default action to be specified.

7. It SHOULD be possible to allow a hierarchy of authorization policies to be used.

5. Discussion and Open Issues

5.1. Extending Geopriv Authorization Policies

To fulfill requirements (1) to (6), it is necessary to decide if [I-D.ietf-geopriv-common-policy] and [I-D.ietf-simple-presence-rules] can be extended.

The following open issues have been identified:

- The authorization policies defined by the Geopriv working group focus on a whitelist approach. This document also raises the question of a backlisting capability that might need to be supported.

- The Geopriv Common Policy mechanism does not allow "deny" actions to be defined. This aspect refers to requirements (4) ("all") where (although "all except one" is supported by Common Policy).

- Requirement 2 (wildcards) is provided by Common Policy in a limited fashion by referring to the domain part of an identity. Regular expressions are not supported.

5.2. Hierarchical Authorization Policy Documents

If requirement (7) is valid then a conflict resolution mechanism needs to be evaluated. Geopriv Common Policy currently defines a very simple mechanism but it is for further investigation whether it is actually applicable to this problem domain. Other policy languages define a more sophisticated set of conflict resolution mechanisms with preceedence and weights for policies. Although this might be an obviously solution for usage in the context of hierarchical authorization policies it causes problems in other places.

6. IANA Considerations

This document does not require actions by IANA.

7. Security Considerations

The security concerns are related to the ability of certain entities to create, update and delete authorization policies. If an unauthorized entity is allowed to modify policies (and to distribute them to other domains) then a denial of service attack is the consequence with impact for more than a single end point.

8. Acknowledgements

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997.

9.2. Informative References

[I-D.ietf-geopriv-common-policy]
Schulzrinne, H., "A Document Format for Expressing Privacy Preferences", Internet-Draft ietf-geopriv-common-policy-07, February 2006.

[I-D.ietf-simple-presence-rules]
Rosenberg, J., "Presence Authorization Rules", Internet-Draft ietf-simple-presence-rules-04, October 2005.

[I-D.ietf-simple-xcap]
Rosenberg, J., "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", draft-ietf-simple-xcap-08 (work in progress), October 2005.

[I-D.ietf-sip-identity]
Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-06 (work in progress), October 2005.

[I-D.ietf-sipping-spam]
Rosenberg, J., "The Session Initiation Protocol (SIP) and Spam", draft-ietf-sipping-spam-01 (work in progress), July 2005.

[I-D.jennings-sip-hashcash]
Jennings, C., "Computational Puzzles for SPAM Reduction in SIP", draft-jennings-sip-hashcash-03 (work in progress), October 2005.

[I-D.rosenberg-sipping-consent-framework]
Rosenberg, J. and J. Rosenberg, "A Framework for Consent-Based Communications in the Session Initiation Protocol (SIP)", draft-rosenberg-sipping-consent-framework-00 (work in progress), July 2004.

[I-D.schwartz-sipping-spit-saml]
Schwartz, D., "SPAM for Internet Telephony (SPIT) Prevention using the Security Assertion Markup Language (SAML)", draft-schwartz-sipping-spit-saml-00 (work in progress), October 2005.

[I-D.tschofenig-sip-saml]
Tschofenig, H., "Using SAML for SIP",
draft-tschofenig-sip-saml-04 (work in progress),
July 2005.

Authors' Addresses

Geoffrey Dawirs
University of Namur
21, rue Grandgagnage
Namur B-5000
Belgique

Email: gdawirs@gdawirs.be

Thomas Froment
Alcatel
1, rue Ampere - BP 80056
Massy, Paris 91302
France

Email: Thomas.Froment@alcatel.fr

Hannes Tschofenig
Siemens
Otto-Hahn-Ring 6
Munich, Bavaria 81739
Germany

Email: Hannes.Tschofenig@siemens.com
URI: http://www.tschofenig.com

Intellectual Property Statement

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.