Groupe de travail Réseau
Request pour Comments : 4656
Catégorie : Sur la voie de la normalisation
Traduction Claude Brière de L'Isle

S. Shalunov, Internet2
B. Teitelbaum, Internet2
A. Karp, Univ. Wisconsin
J. Boote, Internet2
M. Zekauskas, Internet2
septembre 2006

Protocole de mesure active unidirectionnelle (OWAMP)

Résumé
Le protocole de mesure active unidirectionnelle (OWAMP, *One-Way Active Measurement Protocol*) mesure les caractéristiques unidirectionnelles comme le délai dans une direction et les pertes dans une direction. La mesure de haute précision de ces métriques de performances IP dans une direction est devenue possible avec une plus grande disponibilité de bonnes sources d'heure (comme le GPS et CDMA). OWAMP permet l'interopérabilité de ces mesures.

Table des matières

1. Introduction

Le groupe de travail Métriques des performances IP (IPPM, *IP Performance Metrics*) de l'IETF a défini des métriques pour le délai unidirectionnel de paquet [RFC2679] et la perte de paquet [RFC2680] sur les chemins de l'Internet. Bien qu'il y ait maintenant plusieurs plateformes de mesures qui mettent en œuvre une collection de ces métriques [SURVEYOR], [SURVEYOR-INET], [RIPE], [BRIX], il n'y a pas actuellement de norme qui permette l'initiation de flux d'essais ou d'échange de paquets pour collecter les métriques de manière interopérable.

Avec la disponibilité croissante de systèmes de positionnement mondial (GPS, *global positioning system*) abordables et de sources horaires fondées sur CDMA, les hôtes ont de plus en plus de sources horaires très précises qui leur sont disponibles, soit directement, soit à proximité par des serveurs horaires principaux (couche 1) du protocole de l'heure du réseau (NTP, *Network Time Protocol*). En normalisant une technique pour collecter les mesures actives unidirectionnelles IPPM, on espère créer un environnement dans lequel les métriques IPPM peuvent être collectées à travers un maillage beaucoup plus large de chemins Internet qu'il n'est actuellement possible. Une vision particulièrement séduisante serait un large déploiement de serveurs OWAMP ouverts qui rendraient les mesures de délai unidirectionnel aussi courantes que la mesure du temps d'aller-retour en utilisant un outil comme ping fondé sur ICMP.

Les objectifs supplémentaires de la conception de OWAMP incluent d'être difficile à détecter et manipuler, d'être sûr, d'avoir une séparation logique des fonctions de contrôle et d'essai, et la prise en charge de petits paquets d'essai. (La difficulté de détection rend les interférences avec les mesures plus difficiles pour des intermédiaires au milieu du réseau.)

Le trafic d'essai OWAMP est difficile à détecter parce que il est simplement un flux de paquets UDP de et vers des numéros d'accès négociés, avec potentiellement rien de statique dans les paquets (la taille est aussi négociée). OWAMP prend aussi en charge un mode chiffré qui obscurcit encore le trafic et rend impossible d'altérer les horodatages sans être détecté.

Les caractéristiques de sécurité incluent une authentification et/ou chiffrement facultatifs des messages de contrôle et d'essai. Ces caractéristiques peuvent être utiles pour prévenir un accès non autorisé aux résultats ou des attaques par interposition qui tenteraient de donner un traitement particulier aux flux d'essais OWAMP ou qui tenteraient de modifier les horodatages générés par l'envoyeur pour falsifier les résultats d'essais.

Dans ce document, les mots clés "DOIT", "EXIGÉ", "DEVRAIT", "RECOMMANDÉ", et "PEUT" sont à interpréter comme décrit dans la [RFC2119].

1.1 Relations des protocoles d'essai et de contrôle

OWAMP consiste en fait en deux protocoles en relation directe : OWAMP-Control et OWAMP-Test. OWAMP-Control est utilisé pour initier, commencer, et arrêter les sessions d'essais et aller chercher leurs résultats, tandis que OWAMP-Test est utilisé pour échanger les paquets d'essais entre deux nœuds de mesures.

Bien que OWAMP-Test puisse être utilisé en conjonction avec un protocole de contrôle autre que OWAMP-Control, les auteurs ont délibérément choisi d'inclure les deux protocoles dans la même RFC pour encourager la mise en œuvre et le déploiement de OWAMP-Control comme dénominateur commun de protocole de contrôle pour les mesures actives unidirectionnelles. Avoir une solution complète et ouverte de mesures actives unidirectionnelles simple à mettre en œuvre et déployer est crucial pour assurer un futur dans lequel la mesure active unidirectionnelle inter-domaines pourrait devenir aussi courante que le ping. On n'anticipe ni ne recommande que OWAMP-Control forme la base d'un protocle d'utilisation générale extensible de contrôle de mesure et de surveillance.

OWAMP-Control est conçu pour prendre en charge la négociation de sessions de mesures actives unidirectionnelles et la restitution des résultats d'une manière directe. À l'initialisation de la session, il y a une négociation des adresses d'envoyeur et de receveur et des numéros d'accès, de l'heure de début de session, de la longueur de la session, de la taille des paquets d'essai, de l'intervalle d'échantillonnage moyen de Poisson pour le flux d'essai, et des attributs de la notion très générale de type de paquet [RFC 2330], incluant la taille de paquet et le comportement par bond (PIB, *per hop behavior*) [RFC2474], qui pourraient être utilisés pour prendre en charge les caractéristiques de réseau unidirectionnelles à travers des réseaux de services différenciés. De plus, OWAMP-Control prend en charge le chiffrement et l'authentification par session pour le trafic de contrôle et d'essai, des serveurs de mesures qui peuvent agir comme mandataires pour les points d'extrémité des flux d'essais, et l'échange d'une valeur de germe pour le processus pseudo-aléatoire de Poisson qui décrit le flux d'essais généré par l'envoyeur.

On estime que OWAMP-Control peut effectivement prendre en charge la mesure active unidirectionnelle dans divers environnements, des balises de mesure publiquement accessibles fonctionnant sur des hôtes arbitraires à des déploiements de surveillance de réseau au sein de réseaux d'entreprise privés. Si l'intégration avec le protocole simple de gestion de réseau (SNMP, *Simple Network Management Protocol*) ou des protocoles propriétaires de gestion de réseau est requis, des passerelles peuvent être créées.

1.2 Modèle logique

Plusieurs rôles sont logiquement séparés pour permettre une large souplesse d'usage. Spécifiquement, on définit ici :

Envoyeur de session : le point d'extrémité d'envoi d'une session OWAMP-Test ;

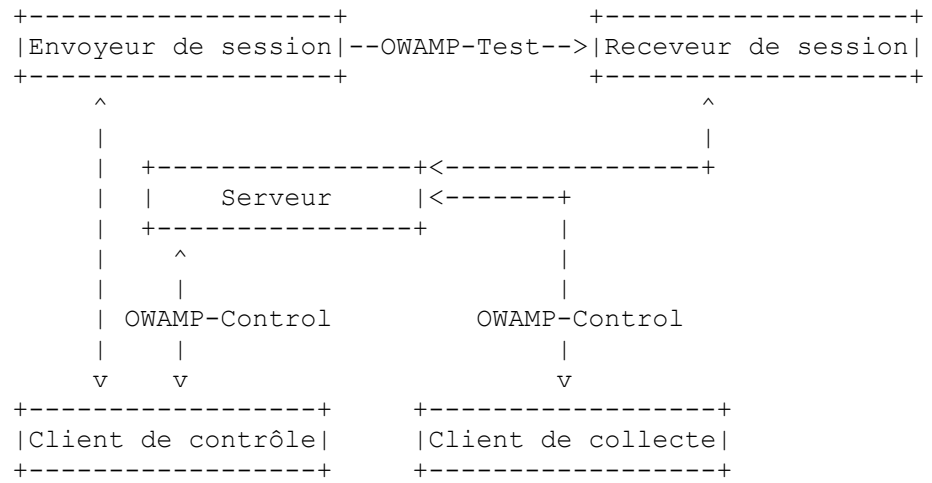
Receveur de session : le point d'extrémité de réception d'une session OWAMP-Test ;

Serveur : un système d'extrémité qui gère une ou plusieurs sessions OWAMP-Test, est capable de configurer un état par session dans les points d'extrémité de session, et est capable de retourner les résultats d'une session d'essais ;

Client de contrôle : un système d'extrémité qui initie les demandes de sessions OWAMP-Test, déclenche le début d'un ensemble de sessions, et peut déclencher leur terminaison ;

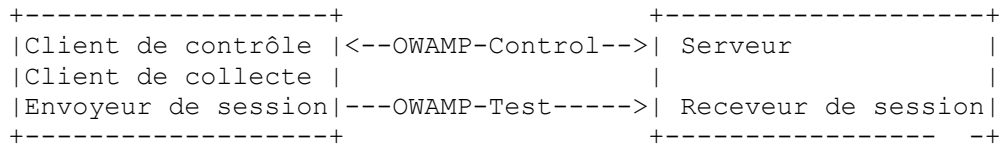
Client de collecte : un système d'extrémité qui initie les demandes d'aller chercher les résultats des sessions OWAMP-Test achevées.

Un scénario possible de relations entre ces rôles est montré ci-dessous.



(Les liaisons non marquées dans la figure ne sont pas spécifiées dans ce document et peuvent être des protocoles propriétaires.)

Différents rôles logiques peut être tenus par le même hôte. Par exemple, dans la figure ci-dessus, il pourrait n'y avoir que deux hôtes : un tenant les rôles de Client de contrôle, de Client de collecte, et d'envoyeur de session, et l'autre tenant les rôles de Serveur et de Receveur de session. C'est ce qui est montré ci-dessous.



Finalement, comme de nombreux chemins de l'Internet incluent des segments qui transportent IP sur ATM, des mesures de délai et de pertes peuvent inclure les effets de la segmentation et réassemblage (SAR) ATM. Par conséquent, OWAMP a été conçu pour permettre que de petits paquets d'essai tiennent dans la charge utile d'une seule cellule ATM (ceci n'est réalisé qu'en mode non authentifié).

2. Vue d'ensemble du protocole

Comme décrit ci-dessus, OWAMP consiste en deux protocoles en inter-relations : OWAMP-Control et OWAMP-Test. Le premier est mis en couche sur TCP et est utilisé pour initier et contrôler les sessions de mesures et aller chercher leurs résultats. Le dernier protocole est mis en couche sur UDP et est utilisé pour envoyer des paquets d'une seule mesure le long des chemins Internet soumis aux essais.

L'initiateur de la session de mesures établit une connexion TCP à un accès bien connu, 861, sur le point cible et cette connexion reste ouverte pour la durée des sessions OWAMP-Test. Un serveur OWAMP DEVRAIT écouter sur cet accès bien connu.

Les messages OWAMP-Control sont seulement transmis avant que les sessions OWAMP-Test commencent réellement et après qu'elles sont achevées (avec l'exception possible d'un message Stop-Sessions précoce).

Les protocoles OWAMP-Control et OWAMP-Test prennent en charge trois modes de fonctionnement : non authentifié, authentifié, et chiffré. Les modes authentifié ou chiffré exigent que les points d'extrémité possèdent un secret partagé.

Toutes les quantités multi-octets définies dans ce document sont représentées comme des entiers non signés dans l'ordre des octets du réseau sauf spécifié autrement.

3. OWAMP-Control

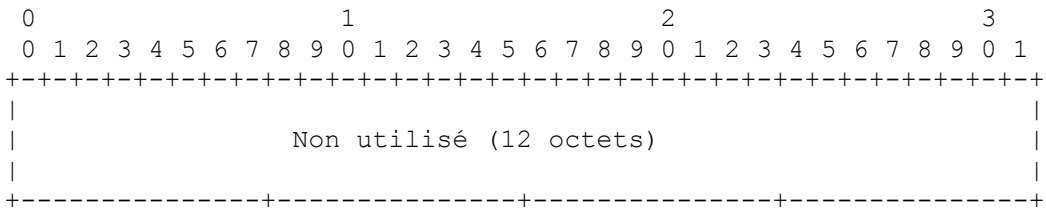
Le type de chaque message OWAMP-Control peut être trouvé après la lecture des 16 premiers octets. La longueur de chaque message OWAMP-Control peut être calculée à la lecture de sa partie de taille fixe. Aucun message ne fait moins de 16 octets.

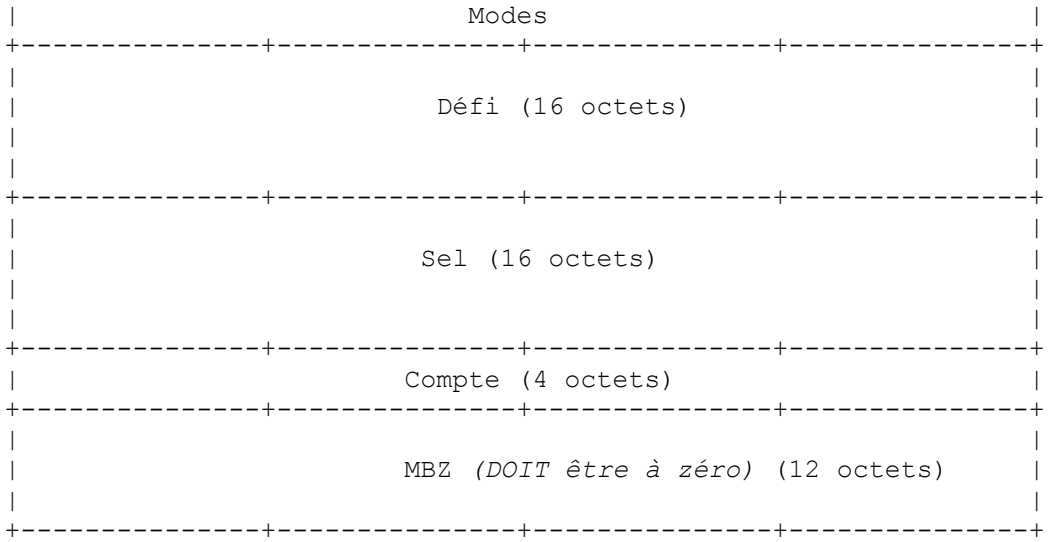
Une mise en œuvre DEVRAIT purger l'état non utilisé pour empêcher les attaques de déni de service, ou un usage illimité de la mémoire, sur le serveur. Par exemple, si le message de contrôle complet n'est pas reçu dans un certain nombre de minutes après qu'il est attendu, la connexion TCP associée à la session OWAMP-Control DEVRAIT être abandonnée. En l'absence d'autres considérations, 30 minutes semble une limite supérieure raisonnable.

3.1 Établissement de connexion

Avant qu'un client de contrôle ou un client de collecte puisse produire des commandes à un serveur, il doit établir une connexion avec le serveur.

D'abord, un client ouvre une connexion TCP avec le serveur sur l'accès bien connu 861. Le serveur répond avec un accueil de serveur :





Les valeurs de mode suivantes ont une signification : 1 pour non authentifié, 2 pour authentifié, et 4 pour chiffré. La valeur du champ Modes envoyée par le serveur est le OU au bit près des valeurs de mode qu'il veut prendre en charge durant cette session. Donc, les trois derniers bits de la valeur Modes de 32 bits sont utilisés. Les 29 premiers bits DOIVENT être à zéro. Un client DOIT ignorer les valeurs dans les 29 premiers bits de la valeur de Modes. (De cette façon, les bits sont disponibles pour de futures extensions du protocole. C'est le seul mécanisme d'extension prévu.)

Défi est une séquence aléatoire d'octets générés par le serveur ; il est utilisé ensuite par le client pour prouver la possession d'un secret partagé de la manière prescrite ci-dessous.

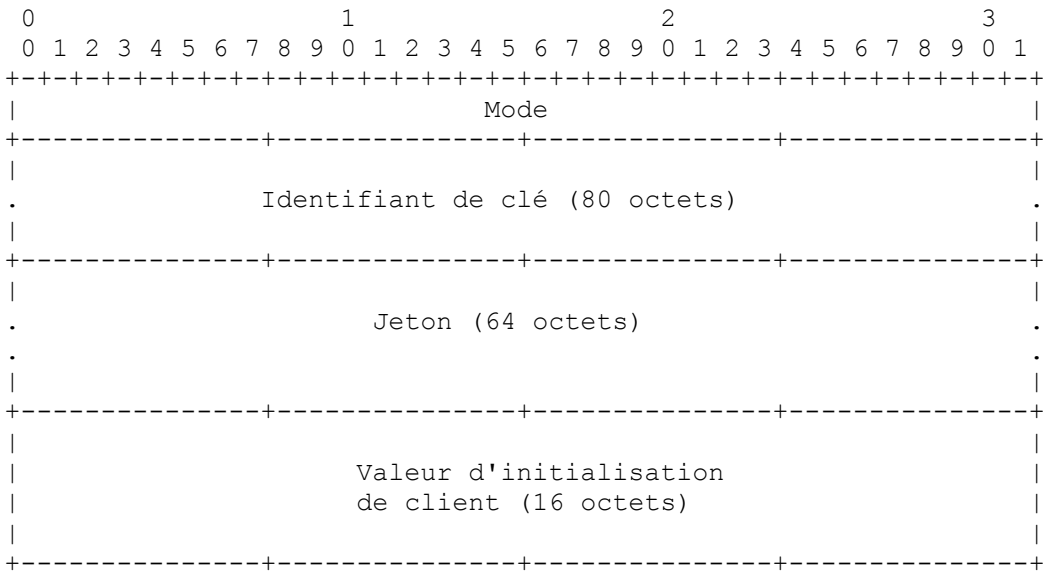
Sel et Compte sont des paramètres utilisés pour déduire une clé d'un secret partagé, comme décrit ci-dessous.

Sel DOIT être généré de façon pseudo-aléatoire (indépendamment de tous les autres processus de ce document).

Compte DOIT être une puissance de 2. Compte DOIT être au moins 1024. Compte DEVRAIT être augmenté lorsque des puissance de calcul supérieures deviendront courantes.

Si la valeur de Modes est zéro, le serveur ne souhaite pas communiquer avec le client et PEUT clore la connexion immédiatement. Le client DEVRAIT clore la connexion si il reçoit un accueil avec Modes égal à zéro. Le client PEUT clore la connexion si le mode désiré du client est indisponible.

Autrement, le client DOIT répondre avec le message Set-Up-Response suivant :



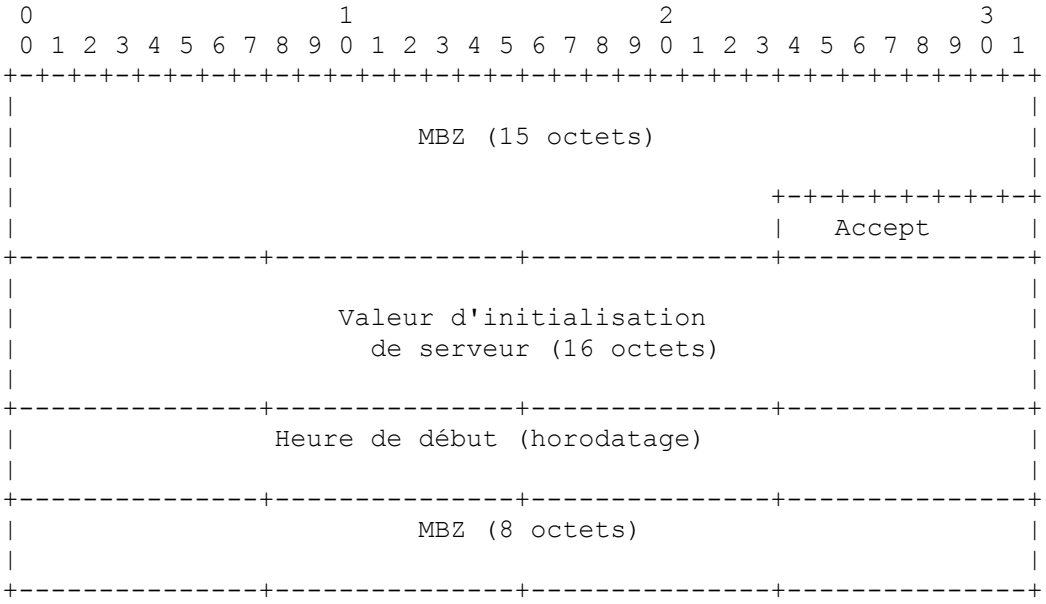
Ici Mode est le mode que le client choisit d'utiliser durant cette session OWAMP-Control. Il va aussi être utilisé pour toutes les sessions OWAMP-Test commencées sous le contrôle de cette session OWAMP-Control. Dans Mode, un ou zéro bit DOIT être établi parmi les trois derniers bits. Si il y a un bit établi parmi les trois derniers bits, ce bit DOIT indiquer un mode que le serveur a accepté d'utiliser (c'est-à-dire, le même bit DOIT avoir été établi par le serveur dans l'accueil du serveur). Les 29 premiers bits de Mode DOIVENT être zéro. Un serveur DOIT ignorer les valeurs des 29 premiers bits. Si zéro bit de Mode n'est établi par le client, ceci indique qu'il ne va pas continuer la session ; dans ce cas, le client et le serveur DEVRAIENT clore la connexion TCP associée à la session OWAMP-Control.

En mode non authentifié, Identifiant de clé, Jeton, et Valeur d'initialisation de client sont inutilisés. Autrement, Identifiant de clé *(KeyID)* est une chaîne UTF-8, longue de jusqu'à 80 octets (si la chaîne est plus courte, elle est bourrée avec des octets à zéro) cela dit au serveur quel secret partagé le client souhaite utiliser pour authentifier ou chiffrer, tandis que Jeton est l'enchaînement d'un défi de 16 octets, une clé de session AES de 16 octets utilisée pour le chiffrement, et une clé de session HMAC-SHA1 de 32 octets utilisée pour l'authentification. Le jeton lui-même est chiffré en utilisant AES *(Advanced Encryption Standard*) [AES] en mode de chaînage de bloc de chiffrement (CBC, *Cipher Block Chaining*). Le chiffrement DOIT être effectué en utilisant une valeur d'initialisation (IV, *Initialization Vector*) de zéro et une clé déduite du secret partagé associé à l'identifiant de clé. (Le serveur et le client utilisent les mêmes transpositions de KeyID en secrets partagés. Le serveur, qui est prêt à conduire des sessions avec plus d'un client, utilise les KeyID pour choisir la clé secrète appropriée ; un client va normalement avoir différentes clés secrètes pour différents serveurs. La situation est analogue à celle avec des mots de passe.)

Le secret partagé est une phrase de passe ; elle NE DOIT PAS contenir de nouvelles lignes. La clé secrète est déduite de la phrase de passe en utilisant une fonction de déduction de clé PBKDF2 (PKCS n° 5) fondée sur un mot de passe [RFC2898]. La fonction PBKDF2 exige plusieurs paramètres : la PRF est HMAC-SHA1 [RFC2104] ; le sel et le compte sont ceux transmis par le serveur.

La clé de session AES, la clé de session HMAC et la valeur d'initialisation de client sont générées au hasard par le client. La clé de session AES et la clé de session HMAC DOIVENT être générées avec une entropie suffisante pour ne pas réduire la sécurité du chiffrement sous-jacent [RFC4086]. La valeur d'initialisation du client a simplement besoin d'être unique (c'est-à-dire, elle NE DOIT jamais être répétée pour des sessions différentes qui utilisent la même clé secrète ; une façon simple de faire cela sans utiliser en état étrange est de générer des valeurs de Client-IV en utilisant une source de nombres pseudo aléatoires cryptographiquement sûre : si c'est fait, la production de la première répétition est improbable avant 2^64 sessions avec la même clé secrète).

Le serveur DOIT répondre avec le message Server-Start suivant :



Les parties MBZ DOIVENT être à zéro. Le client DOIT ignorer leur valeur. Les champs MBZ (DOIT être zéro) ici et après ont la même sémantique : la partie qui envoie le message DOIT régler le champ de façon telle que tous les bits soient égaux à zéro ; la partie qui interprète le message DOIT ignorer la valeur. (De cette façon, le champ pourrait être utilisé pour de futures extensions.)

Valeur d'initialisation de serveur *(Server-IV)* est généré au hasard par le serveur. En mode non authentifié, Server-IV est inutilisé.

Le champ Accept indique la volonté du serveur de continuer la communication. Une valeur de zéro dans le champ Accept signifie que le serveur accepte l'authentification et veut conduire d'autres transactions. Des valeurs non zéro indiquent que le serveur n'accepte pas l'authentification ou, pour quelque autre raison, ne veut pas mener d'autre transaction dans cette session OWAMP-Control. La liste complète des valeurs Accept disponibles est donnée au paragraphe 3.3, "Valeurs du champ Accept".

Si une réponse négative (non zéro) est envoyée, le serveur PEUT (et le client DEVRAIT) clore la connexion après ce message.

Heure de début *(Start-Time)* est un horodatage représentant l'heure à laquelle l'instance actuelle de serveur a commencé à fonctionner. (Par exemple, dans un système d'exploitation multi-utilisateurs général, ce pourrait être l'heure où le processus de serveur a démarré.) Si Accept n'est pas zéro, Heure de début DEVRAIT être réglé de façon à ce que tous ses bits soient des zéros. En modes authentifié et chiffré, Heure de début est chiffré comme décrit au paragraphe 3.4, "Commandes OWAMP-Control", sauf si Accept n'est pas zéro. (Les modes authentifié et chiffré ne peuvent pas être activés si la connexion de contrôle n'est pas initialisée.)

Le format d'horodatage est décrit au paragraphe 4.1.2. La même instance de serveur DEVRAIT rapporter l'exacte même valeur d'Heure de début à chaque client dans chaque session.

Les transactions précédentes constituent l'établissement de connexion.

3.2 Protection d'intégrité (HMAC)

L'authentification de chaque message (aussi appelé une commande dans ce document) dans OWAMP-Control est accomplie en y ajoutant un HMAC. Le HMAC que OWAMP utilise est le HMAC-SHA1 tronqué à 128 bits. Donc, tous les champs HMAC font 16 octets. Un HMAC a besoin d'une clé. La clé de session HMAC est communiquée avec la clé de session AES durant l'établissement de la connexion OWAMP-Control. La clé de session HMAC DEVRAIT être déduite indépendamment de la clé de session AES (une mise en œuvre PEUT, bien sûr, utiliser le même mécanisme pour générer les bits aléatoires pour les deux clés). Chaque HMAC envoyé couvre tout ce qui est envoyé dans une certaine direction entre le HMAC précédent (mais sans l'inclure) et jusqu'au début du nouveau HMAC. De cette façon, une fois que le chiffrement est établi, chaque bit de la connexion OWAMP-Control est authentifié exactement une fois par un HMAC.

Lors du chiffrement, l'authentification se produit avant le chiffrement, de sorte que les blocs HMAC sont chiffrés avec le reste du flux. Au déchiffrement, l'ordre est bien sûr inversé : d'abord on déchiffre, puis on vérifie le HMAC, puis on procède à l'utilisation des données.

Le HMAC DOIT être vérifié aussitôt que possible pour éviter d'utiliser et propager des données corrompues.

En mode ouvert, les champs HMAC sont inutilisés et ont la même sémantique que les champs MBZ.

3.3 Valeurs du champ Accept

Les valeurs Accept sont utilisées dans tout le protocole OWAMP-Control pour communiquer la réponse du serveur aux demandes du client. Le jeu complet des valeurs valides du champ Accept est le suivant :
0 : OK.
1 : échec, raison non spécifiée (fourre-tout).
2 : erreur interne.
3 : un aspect de la demande n'est pas supporté.
4 : la demande ne peut être satisfaite à cause de limitations permanentes de ressources.
5 : la demande ne peut être satisfaite à cause de limitations temporaires de ressources.

Toutes les autres valeurs sont réservées. L'envoyeur du message PEUT utiliser la valeur 1 pour toutes les valeurs Accept non zéro. Un envoyeur de message DEVRAIT utiliser la valeur correcte de Accept si il va utiliser d'autres valeurs. Le receveur de message DOIT interpréter toutes les valeurs de Accept autres que celles réservées comme 1. De cette façon, les autres valeurs sont disponibles pour de futures extensions.

3.4 Commandes OWAMP-Control

En mode authentifié ou chiffré (qui sont identiques pour ce qui concerne OWAMP-Control, et ne diffèrent que dans OWAMP-Test) toutes les communications suivantes sont chiffrées avec la clé de session AES (en mode CBC) et authentifiées avec la clé de session HMAC. Le client chiffre tout ce qu'il envoie sur la connexion OWAMP-Control qui vient d'être établie en utilisant le chiffrement de flux avec la valeur d'initialisation de client comme IV. De la même façon, le serveur chiffre son côté de la connexion en utilisant la valeur d'initialisation de serveur comme IV.

Les IV elles mêmes sont transmises en clair. Le chiffrement commence avec le bloc qui suit immédiatement celui qui contient l'IV. Les deux flux (un allant du client au serveur et un en retour) sont chiffrés indépendamment, chacun avec sa propre IV, mais utilisant la même clé (la clé de session AES).

Les commandes suivantes sont disponibles pour le client : Request-Session *(demande de session)*, Start-Sessions *(commencer la session)*, Stop-Sessions *(arrêter les sessions)*, et Fetch-Session *(aller chercher la session)*. La commande Stop-Sessions est disponible pour le client et le serveur. (Le serveur peut aussi envoyer d'autres messages en réponse aux commandes qu'il reçoit.)

Après que le client a envoyé la commande Start-Sessions et jusqu'à ce qu'il envoie et reçoive (dans un ordre non spécifié) la commande Stop-Sessions, il est dit conduire des mesures actives. De même, le serveur est dit conduire des mesures actives après qu'il a reçu la commande Start-Sessions et jusqu'à ce qu'il envoie et reçoive (dans un ordre non spécifié) la commande Stop-Sessions.

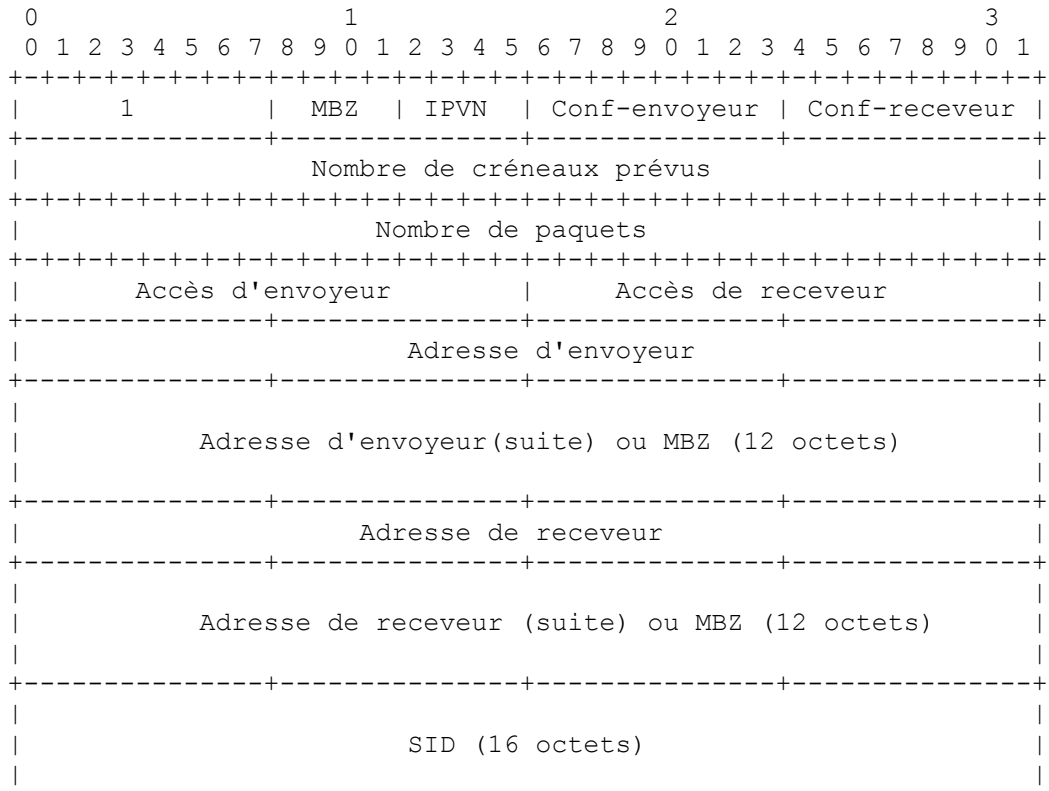
Pendant la conduite des mesures actives, la seule commande disponible est Stop-Sessions.

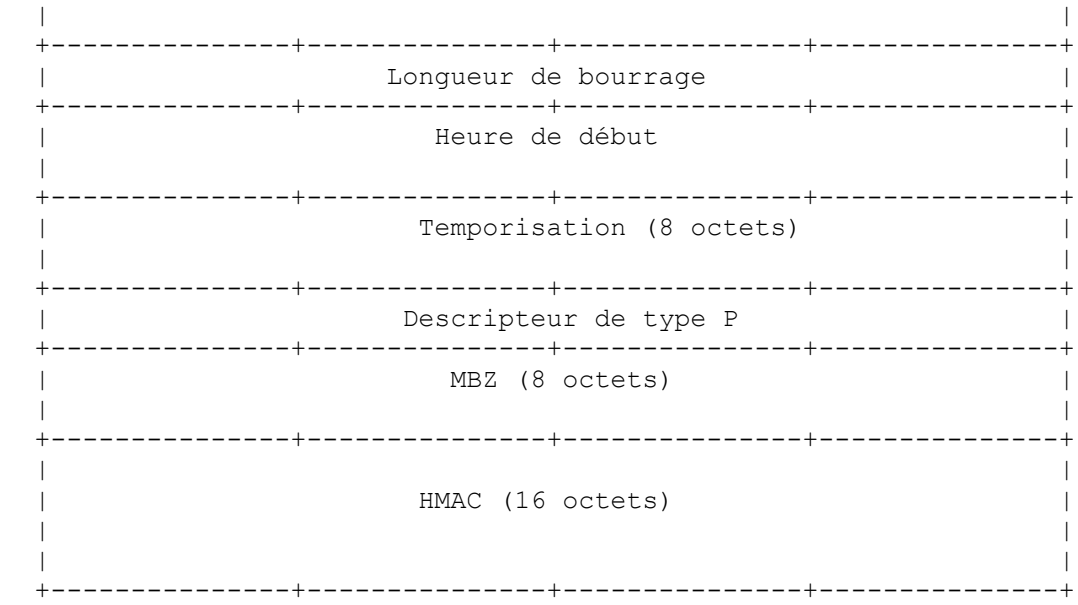
Ces commandes sont décrites en détail ci-dessous.

3.5 Création de sessions d'essais

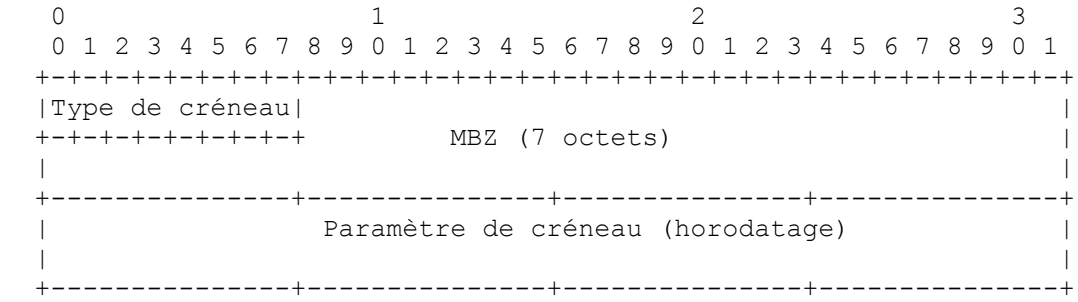
Les sessions individuelles de mesures actives unidirectionnelles sont établies en utilisant un simple protocole de demande/réponse. Dans OWAMP, le client PEUT produire zéro, un ou plusieurs messages Request-Session à un serveur OWAMP, qui DOIT répondre à chacun avec un message Accept-Session. Un message Accept-Session PEUT refuser une demande.

Le format du message Request-Session est le suivant :

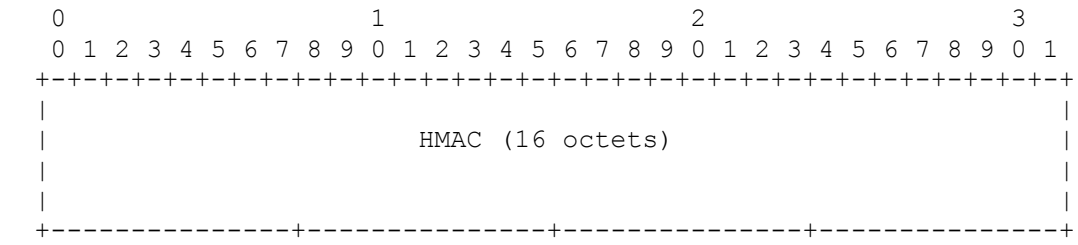




Ceci est immédiatement suivi par une ou plusieurs descriptions de créneaux prévus (le nombre de créneaux prévus est spécifié dans le champ "Nombre de créneaux prévus" ci-dessus) :



Ceci est immédiatement suivi par le HMAC :



Tous ces messages constituent un message logique, la commande Request-Session.

En haut, le premier octet (1) indique que c'est la commande Request-Session.

IPVN est le numéro de version IP pour l'envoyeur et le receveur. Quand le numéro de version IP est 4, 12 octets suivent les 4 octets de l'adresse IPv4 mémorisée dans Adresse d'envoyeur et Adresse de receveur. Ces octets DOIVENT être réglés à zéro par le client et DOIVENT être ignorés par le serveur. Les valeurs actuellement significatives de IPVN sont 4 et 6.

Conf-envoyeur et Conf-receveur DOIVENT être réglés à 0 ou 1 par le client. Le serveur DOIT interpréter toute valeur non zéro comme 1. Si la valeur est 1, il est demandé au serveur de configurer l'agent correspondant (envoyeur ou receveur). Dans ce cas, la valeur Accès correspondante DEVRAIT être ignorée par le serveur. Au moins un des champs Conf-envoyeur et Conf-receveur DOIT être 1. (Les deux peuvent être établis, et dans ce cas il est demandé au serveur d'effectuer une session entre les deux hôtes qu'il peut configurer.)

Nombre de créneaux prévus, comme mentionné précédemment, spécifie le nombre d'enregistrements de créneaux qui vont

entre les deux blocs de HMAC. Il est utilisé par l'envoyeur pour déterminer quand envoyer des paquets d'essai (voir le paragraphe suivant).

Nombre de paquets est le nombre de paquets de mesure active qui vont être envoyés durant cette session OWAMP-Test (noter que le serveur ou le client peut interrompre prématurément la session).

Si Conf-envoyeur n'est pas établi, Accès d'envoyeur est l'accès UDP à partir duquel les paquets OWAMP-Test vont être envoyés. Si Conf-receveur n'est pas établi, Accès de receveur est l'accès UDP OWAMP-Test auquel il est demandé d'envoyer les paquets.

Les champs Adresse d'envoyeur et Adresse de receveur contiennent, respectivement, les adresses d'envoyeur et de receveur aux points d'extrémité du chemin Internet sur lequel une session d'essais OWAMP est demandée.

SID est l'identifiant de session. Il peut être utilisé dans des sessions ultérieures comme argument pour la commande Fetch-Session. Il n'a de signification que si Conf-receveur est 0. De cette façon, le SID est toujours généré par le côté receveur. Des informations sur la façon de générer le SID figurent à la fin de ce paragraphe.

Longueur de bourrage est le nombre d'octets à ajouter au paquet normal OWAMP-Test (en voir plus sur le bourrage dans la discussion de OWAMP-Test).

Heure de début est l'heure où la session va démarrer (mais pas avant que la commande Start-Sessions soit produite). Cet horodatage est dans le même format que les horodatages OWAMP-Test.

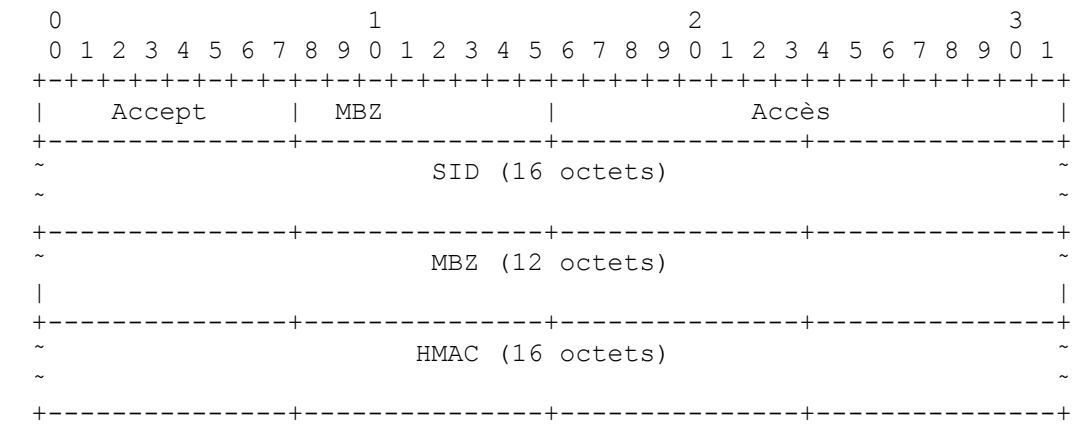
Temporisation (ou un seuil de perte) est un intervalle de temps (exprimé comme un horodatage). Un paquet appartenant à la session d'essai en cours d'établissement par la commande Request-Session actuelle va être considéré comme perdu si il n'est pas reçu durant Temporisation secondes après son envoi.

Descripteur de type P couvre seulement un sous ensemble de l'espace (très grand) de type P. Si les deux premiers bits du descripteur de type P sont 00, alors les six bits suivants spécifient la valeur du codet de service différencié (DSCP, *Differentiated Services Codepoint*) demandée pour les paquets OWAMP-Test envoyés, comme défini dans la [RFC2474]. Si les deux premiers bits du descripteur de type P sont 01, alors les 16 bits suivants spécifient le code d'identification de PHB (PHB ID) demandé, comme défini dans la [RFC2836]. Donc, la valeur toute de zéros spécifie le service au mieux par défaut.

Si Conf-envoyeur est établi, le descripteur de type P est à utiliser pour configurer l'envoyeur à envoyer des paquets conformément à cette valeur. Si Conf-envoyeur n'est pas établi, le descripteur de type P est la déclaration de la configuration de l'envoyeur.

Si Conf-envoyeur est établi et si le serveur ne reconnaît pas le descripteur de type P, ou si il ne peut ou ne souhaite pas établir les attributs correspondants sur les paquets OWAMP-Test, il DEVRAIT rejeter la demande de session. Si Conf-envoyeur n'est pas établi, le serveur DEVRAIT accepter ou rejeter la session, sans prêter attention à la valeur du descripteur de type P.

À chaque message Request-Session, un serveur OWAMP DOIT répondre par un message Accept-Session :



Dans ce message, zéro dans le champ Accept signifie que le serveur veut conduire la session. Une valeur non zéro indique

le rejet de la demande. La liste complète des valeurs disponibles de Accept figure au paragraphe 3.3, "Valeurs du champ Accept".

Si le serveur rejette un message Request-Session, il NE DEVRAIT PAS clore la connexion TCP. Le client PEUT la clore si il reçoit une réponse négative au message Request-Session.

La signification de Accès dans la réponse dépend des valeurs de Conf-envoyeur et Conf-receveur dans l'interrogation qui sollicitait la réponse. Si les deux étaient établis, le champ Accès est inutilisé. Si seul Conf-envoyeur était établi, Accès est l'accès duquel attendre les paquets OWAMP-Test. Si seul Conf-receveur était établi, Accès est l'accès auquel les paquets OWAMP-Test sont envoyés.

Si seul Conf-envoyeur était établi, le champ SID dans la réponse est inutilisé. Autrement, SID est un identifiant de session unique généré par le serveur. Il peut être utilisé plus tard comme bride pour aller chercher les résultats d'une session.

Les SID DEVRAIENT être construits en enchaînant le numéro IP IPv4 de 4 octets appartenant à la machine génératrice, un horodatage de 8 octets, et une valeur aléatoire de 4 octets. Pour réduire la probabilité de collisions, si la machine génératrice a des adresses IPv4 (à l'exception de rebouclage) une d'elles DEVRAIT être utilisée pour la génération de SID, même si toute la communication se fonde sur IPv6. Si il n'y a pas du tout d'adresse IPv4, les quatre derniers octets d'une adresse IPv6 PEUVENT être utilisés à la place. Noter que le SID est toujours choisi par le receveur. Si des valeurs vraiment aléatoires ne sont pas disponibles, il est important que le SID soit rendu imprévisible, car la connaissance du SID pourrait être utilisée pour contrôler l'accès.

3.6 Programmation des envois

L'envoyeur et le receveur ont tous deux besoin de connaître la même programmation d'envoi. De cette façon, quand des paquets sont perdus, le receveur sait quand ils sont supposés être envoyés. Il est souhaitable de compresser les programmations courantes et d'être néanmoins capable d'en utiliser une arbitraire pour les sessions d'essai. Dans de nombreux cas, la programmation va consister en séquences répétées de paquets : de cette façon, la séquence effectue un essai, et l'essai est répété un certain nombre de fois pour collecter des statistiques.

Pour mettre cela en en œuvre, on a un programme avec un nombre donné de créneaux. Chaque créneau a un type et un paramètre. Deux types sont pris en charge : une quantité pseudo aléatoire à distribution exponentielle (notée par un code de 0) et une quantité fixée (notée par un code de 1). Le paramètre est exprimé comme un horodatage et spécifie un intervalle de temps. Pour un créneau de type 0 (quantité pseudo aléatoire à distribution exponentielle) cet intervalle est la valeur moyenne (ou 1/lambda si la fonction de densité de distribution est exprimée par lambda*exp(-lambda*x) pour les valeurs positives de x). Pour un créneau de type 1 (quantité fixée) le paramètre est le délai lui-même. L'envoyeur commence avec le début de la programmation et exécute les instructions dans les créneaux : pour un créneau de type 0, attendre une durée distribuée de façon exponentielle avec une moyenne du paramètre spécifié et ensuite envoyer un paquet d'essai (et passer au créneau suivant) ; pour un créneau de type 1, attendre le délai spécifié et envoyer un paquet d'essai (et passer au créneau suivant). Le programme est circulaire : quand il n'y a plus de créneaux, l'envoyeur retourne au premier créneau.

L'envoyeur et le receveur doivent être capables d'excécuter de façon reproductible le programme entier (afin que, si un paquet est perdu, le receveur puissse quand même y rattacher l'horodatage d'envoi). La reproductibilité d'exécution des créneaux de type 1 est triviale. Pour les créneaux de type 0, on doit être capable de générer des quantités pseudo aléatoires à distribution exponentielle de façon reproductible. La façon de faire cela est discutée à la Section 5, "Calcul des nombres pseudo aléatoires à distribution exponentielle".

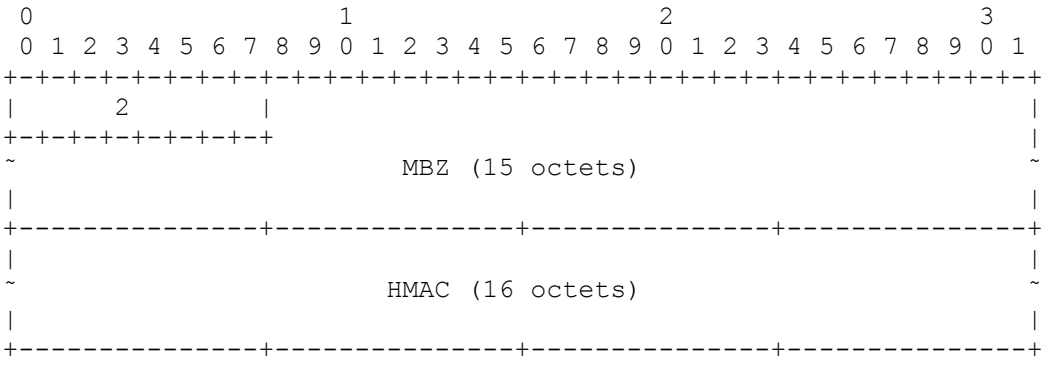
En utilisant ce mécanisme, on peut aisément spécifier les scénarios d'essais courants. En voici quelques exemples :
+ flux de Poisson : un seul créneau de type 0 ;
+ flux périodique : un seul créneau de type 1.
+ flux de Poisson de paires de paquets dos à dos : deux créneaux, de type 0 avec un paramètre non zéro et type 1 avec un paramètre zéro.

De plus, une programmation complètement arbitraire peut être spécifiée (bien qu'inefficace) en rendant le nombre de paquets d'essais égal au nombre de créneaux de programmation. Dans ce cas, le programme complet d'une session OWAMP-Test est transmis à l'avance.

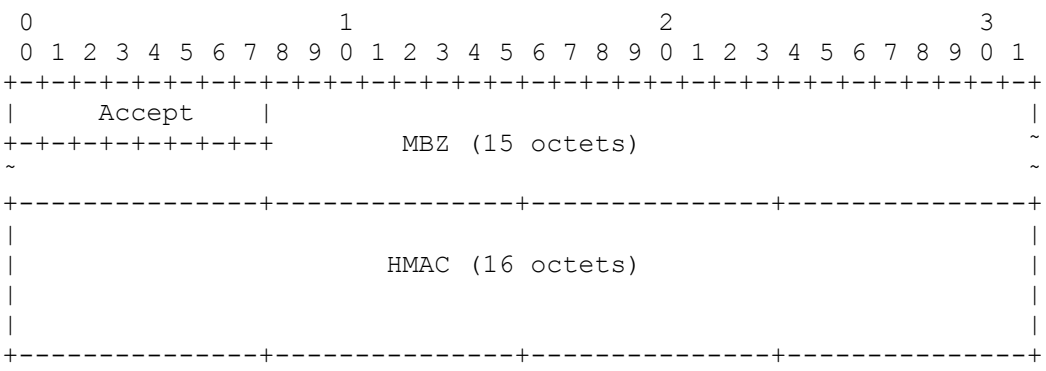
3.7 Début des sessions d'essai

Ayant demandé une ou plusieurs sessions d'essais et reçu des réponses affirmatives Accept-Session, un client OWAMP PEUT commencer l'exécution des sessions d'essais demandées en envoyant un message Start-Sessions au serveur.

Le format de ce message est le suivant :



Le serveur DOIT répondre avec un message Start-Ack (qui DEVRAIT être envoyé aussi rapidement que possible). Les messages Start-Ack ont le format suivant :

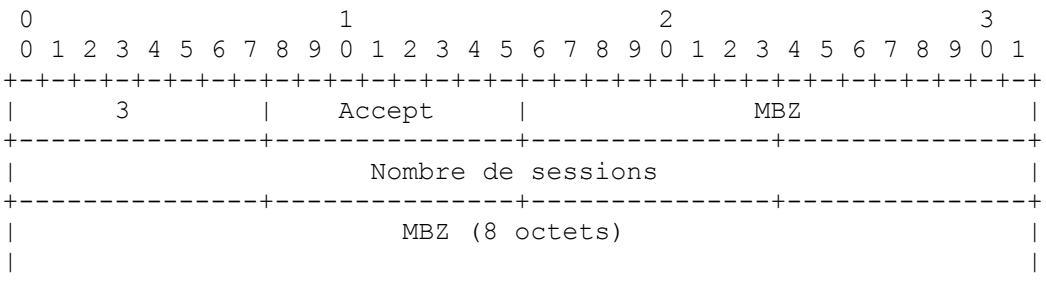


Si Accept est non zéro, la demande Start-Sessions a été rejetée ; zéro signifie que la commande a été acceptée. La liste complète des valeurs Accept disponibles est décrite au paragraphe 3.3, "Valeurs du champ Accept". Le serveur PEUT, et le client DEVRAIT, clore la connexion en cas de rejet.

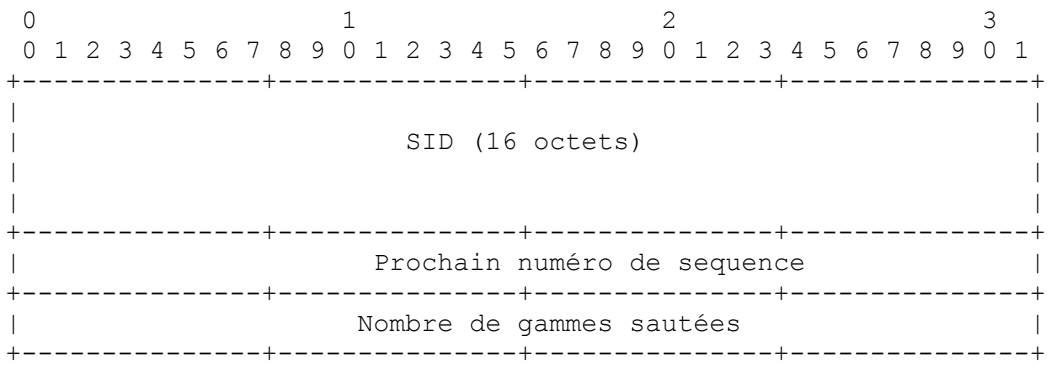
Le serveur DEVRAIT commencer tous les flux OWAMP-Test immédiatement après l'envoi de la réponse ou immédiatement après leur heure de départ spécifiée, selon celle qui est le plus tard. Si le client représente un envoyeur, le client DEVRAIT commencer son flux OWAMP-Test immédiatement après qu'il voit la réponse Start-Ack du serveur (si la commande Start-Sessions a été acceptée) ou immédiatement après son heure de début spécifiée, selon celle qui est le plus tard. En voir plus sur le comportement de l'envoyeur de OWAMP-Test au paragraphe 4.1.

3.8 Stop-Sessions

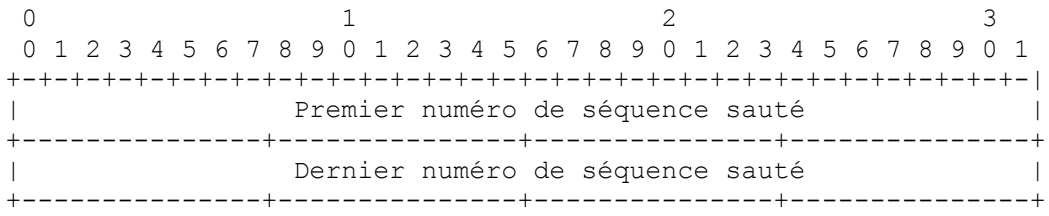
Le message Stop-Sessions peut être produit par le client de contrôle ou par le serveur. Le format de cette commande est le suivant :



Ceci est immédiatement suivi par zéro, un ou plusieurs enregistrements de description de sessions (le nombre d'enregistrements de description de sessions est spécifié dans le champ "Nombre de sessions"). L'enregistrement de description de sessions est utilisé pour indiquer quels paquets ont réellement été envoyés par le processus envoyeur (plutôt que sautés). L'en-tête de l'enregistrement de description de session est le suivant :

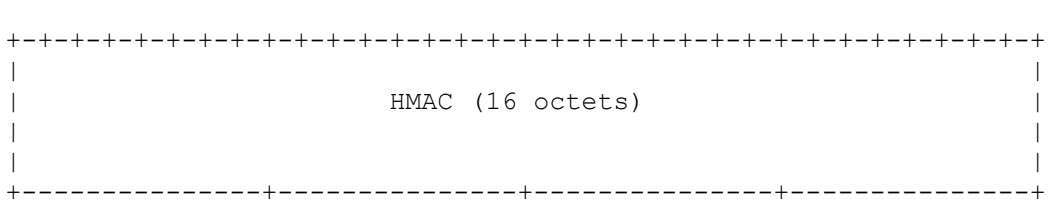


Ceci est immédiatement suivi de zéro, une ou plusieurs descriptions de gammes sautées comme spécifié par le champ "Nombre de gammes sautées" ci-dessus. Les gammes sautées sont simplement deux numéros de séquence qui, ensemble, indiquent une gamme de paquets qui n'ont pas été envoyés :



Les gammes sautées DOIVENT être dans l'ordre. Le dernier bloc de 16 octets (complet ou incomplet) de données DOIT être bourré avec des zéros, si nécessaire. Cela assure que le prochain enregistrement de description de session commence sur une limite de bloc.

Finalement, un seul bloc (16 octets) de HMAC est enchaîné à la fin pour compléter le message Stop-Sessions.



Tous ces enregistrements constituent un message logique : la commande Stop-Sessions.

Ci-dessus, le premier octet (3) indique que c'est la commande Stop-Sessions.

Des valeurs non zéro de Accept indiquent une défaillance d'un certain type. Les valeurs de zéro indiquent l'achèvement normal (mais éventuellement prématuré). La liste complète des valeurs Accept disponibles est au paragraphe 3.3, "Valeurs du champ Accept".

Si Accept a une valeur non zéro (de l'une ou l'autre partie) les résultats de toutes les sessions OWAMP-Test produites par cette session OWAMP-Control DEVRAIENT être considérés comme invalides, même si un Fetch-Session avec le SID provenant de cette session fonctionne pour une session OWAMP-Control différente. Si Accept n'a pas été transmis du tout (pour une raison quelconque, incluant la rupture de la connexion TCP utilisée pour le OWAMP-Control) les résultats de toutes les sessions OWAMP-Test produits par cette session OWAMP-control PEUVENT être considérés comme invalides.

Nombre de sessions indique le nombre d'enregistrements de descriptions de sessions qui suivent immédiatement l'en-tête de Stop-Sessions.

Nombre de sessions DOIT contenir le nombre de sessions envoyées commencées par le côté local de la connexion de contrôle qui n'ont pas été terminées précédemment par une commande Stop-Sessions (c'est-à-dire, le client de contrôle DOIT tenir compte de chaque Request-Session acceptée où Conf-receveur était établi ; le serveur de contrôle DOIT tenir compte de chaque Request-Session acceptée où Conf-envoyeur était établi). Si le message Stop-Sessions ne prend pas exactement en compte les sessions envoyées contrôlées par ce côté, il est alors considéré comme invalide et la connexion DEVRAIT être close et tous les résultats obtenus considérés comme invalides.
Chaque enregistrement de description de session représente une session OWAMP-Test.

SID est l'identifiant de session (SID) utilisé pour indiquer quelle session envoyée est décrite.

Prochain numéro de sequence indique le prochain numéro de séquence qui va être envoyé de cette session d'envoi. Pour les sessions achevées, cela va être égal au nombre de paquets provenant de Request-Session.

Nombre de gammes sautées indique le nombre de trous qui se sont en fait produits dans le processus d'envoi. C'est une gamme de paquets qui n'ont en fait jamais été envoyés par le processus d'envoi. Par exemple, si une session d'envoi est commencée trop tard pour que les dix premiers paquets soient envoyés et si c'est le seul trou dans le programme, alors le "Nombre de gammes sautées" va être 1. La seule description de gamme sautée aura le Premier numéro de séquence sauté égal à 0 et Dernier numéro de séquence sauté égal à 9. Ceci est décrit plus en détail au paragraphe 4.1 "Comportement de l'envoyeur".

Si la connexion OWAMP-Control casse quand la commande Stop-Sessions est envoyée, le receveur PEUT ne pas complètement invalider les résultats de session. Il DOIT éliminer tous les enregistrements de paquets qui suivent (en d'autres termes, qui ont un numéro de séquence supérieur à celui du) le dernier paquet réellement reçu avant un enregistrement de paquet perdu. Cela va aider à différencier entre pertes de paquet dans le réseau et paquets que le processus d'envoi peut n'avoir jamais envoyé.

Si un receveur d'une session OWAMP-Test apprend, par un message Stop-Sessions de OWAMP-Control, que le dernier numéro de séquence de l'envoyeur OWAMP-Test est inférieur à tout numéro de séquence reçu actuellement, les résultats de la session OWAMP-Test complète DOIVENT être invalidés.

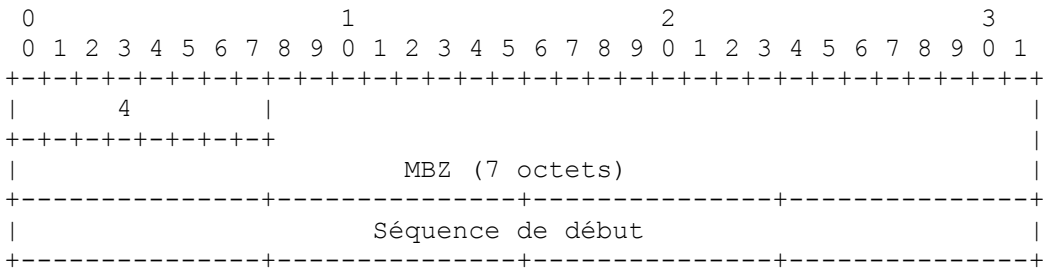
Un receveur d'une session OWAMP-Test, à réception d'une commande Stop-Sessions de OWAMP-Control, DOIT éliminer tous les enregistrements de paquets -- incluant les enregistrements de paquets perdus -- avec une heure d'envoi (calculée) qui tombe entre l'heure actuelle moins Temporisation et l'heure courante. Cela assure la cohérence statistique pour les mesures de pertes et de dupliqués dans le cas où Temporisation est supérieur au temps qu'il faut pour que la commande Stop-Sessions s'exécute.

Pour effectuer des sessions complètes, chaque côté de la connexion de contrôle DEVRAIT attendre que toutes les sessions soient achevées avant d'envoyer le message Stop-Sessions. Le temps complet de chaque session est déterminé comme Temporisation après l'heure programmée pour le dernier numéro de séquence. Les points d'extrémité PEUVENT ajouter un petit incrément à l'heure d'achèvement calculée pour l'envoi aux points d'extrémité pour s'assurer que le message Stop-Sessions atteint le point d'extrémité receveur après Temporisation.

Pour effectuer un arrêt prématuré des sessions, la partie qui initie cette commande DOIT arrêter ses flux d'envoi OWAMP-Test pour envoyer les valeurs de Paquets de session envoyés avant d'envoyer cette commande. Cette partie DEVRAIT attendre d'avoir reçu le message de réponse Stop-Sessions avant d'arrêter les flux de receveur afin qu'elle puisse utiliser les valeurs du message Stop-Sessions reçu pour valider les données.

3.9 Fetch-Session

Le format de cette commande de client est le suivant :



```
|                       Séquence de fin                         |
+---------------+---------------+---------------+---------------+
|                                                               |
|                         SID (16 octets)                       |
|                                                               |
|                                                               |
+---------------+---------------+---------------+---------------+
|                                                               |
|                        HMAC (16 octets)                       |
|                                                               |
|                                                               |
+---------------+---------------+---------------+---------------+
```

Séquence de début est le numéro de séquence du premier paquet demandé. Séquence de fin est le numéro de séquence du dernier paquet demandé. Si Séquence de début est tout de zéros et Séquence de fin tout de uns, cela signifie que la session complète est demandée.

Si une session complète est demandée et qu'elle est encore en cours ou s'est terminée d'une autre façon que normale, la demande d'aller chercher les résultats de session DOIT être refusée. Si une session incomplète est demandée, tous les paquets reçus jusqu'alors qui tombent dans la gamme demandée DEVRAIENT être retournés. Noter que, comme aucune commande ne peut être produite entre Start-Sessions et Stop-Sessions, des demandes incomplètes ne peuvent arriver que sur une connexion OWAMP-Control différente (provenant du même hôte ou d'un hôte différent du client de contrôle).

Le serveur DOIT répondre avec un message Fetch-Ack. Le format de cette réponse de serveur est:le suivant :

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Accept    |   Fini        |          MBZ (2 octets)       |
+---------------+---------------+---------------+---------------+
|                       Prochain numéro de séquence             |
+---------------+---------------+---------------+---------------+
|                       Nombre de gammes sautées                |
+---------------+---------------+---------------+---------------+
|                       Nombre d'enregistrements                |
+---------------+---------------+---------------+---------------+
|                                                               |
|                        HMAC (16 octets)                       |
|                                                               |
|                                                               |
+---------------+---------------+---------------+---------------+
```

Là encore, une valeur différente de zéro dans le champ Accept signifie un rejet de la commande. Le serveur DOIT spécifier zéro pour tous les champs restants si Accept n'est pas zéro. Le client DOIT ignorer tous les champs restants (sauf le HMAC) si Accept n'est pas zéro. La liste complète des valeurs de Accept disponibles figure au paragraphc 3.3.

Fini est différent de zéro si la session OWAMP-Test est terminée.

Prochain numéro de séquence indique le prochain numéro de séquence qui va être vu dans cette session d'envoi. Pour les sessions achevées, cela va être égal au Nombre de paquets provenant de la Request-Session. Cette information n'est disponible que si la session s'est terminée. Si Fini est zéro, alors Prochain numéro de séquence DOIT être mis à zéro par le serveur.

Nombre de gammes sautées indique le nombre de trous qui se sont en fait produits dans le processus d'envoi. Cette information n'est disponible que si la session s'est terminée. Si Fini est zéro, alors Gammes sautées DOIT être mis à zéro par le serveur.

Nombre d'enregistrements est le nombre d'enregistrements de paquets qui entrent dans la gamme demandée. Ce nombre pourrait être inférieur au nombre de paquets dans la reproduction de la commande Request-Session parce qu'une session s'est terminée de façon prématurée, ou il pourrait être supérieur à cause de dupliqués.

Si Accept n'est pas zéro, cela conclut la réponse au message Fetch-Session. Si Accept était 0, le serveur DOIT alors immédiatement envoyer les données de la session OWAMP-Test en question.

Les données de la session OWAMP-Test consistent en l'enchaînement de ce qui suit :

+ une reproduction de la commande Request-Session qui a été utilisée pour commencer la session ; elle est modifiée afin que les numéros d'accès réels d'envoyeur et receveur qui ont été utilisés par la session OWAMP-Test apparaissent toujours dans la reproduction.

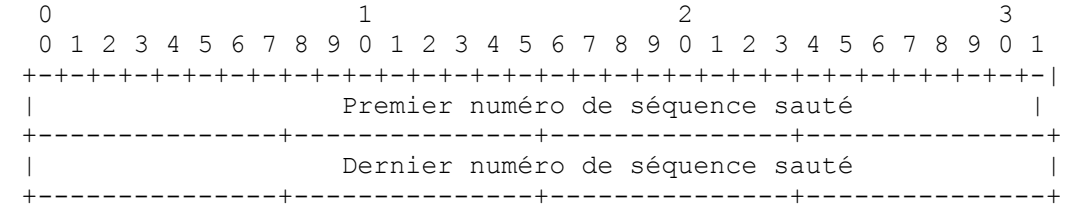
+ zéro, une ou plusieurs (comme spécifié) descriptions de gammes sautées. Le dernier bloc (plein ou éventuellement incomplet) (16 octets) de descriptions de gammes sautées est bourré avec des zéros, si nécessaire.

+ 16 octets de HMAC.

+ zéro, un ou plusieurs (comme spécifié) enregistrements de paquet. Le dernier bloc (plein ou éventuellement incomplet) (16 octets) de données est bourré avec des zéros, si nécessaire.

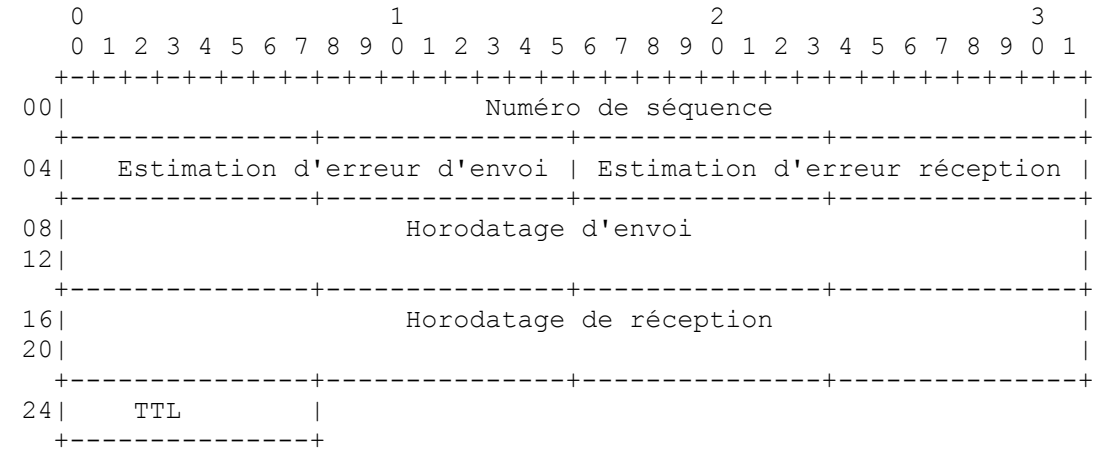
+ 16 octets de HMAC.

Les descriptions de Gammes sautées sont simplement deux numéros de séquence qui, ensemble, indiquent une gamme de paquets qui n'ont pas été envoyés :



Les descriptions de Gammes sautées devraient être envoyés dans l'ordre, comme triées par le premier numéro de séquence. Si des gammes sautées se chevauchent ou sont déclassées, les données de session sont à considérer comme invalides et la connexion DEVRAIT être close et tous les résultats obtenus considérés comme invalides.

Chaque enregistrement de paquet fait 25 octets et inclut 4 octets de numéro de séquence, 8 octets d'horodatage d'envoi, 2 octets d'estimation d'erreur de l'horodatage d'envoi, 8 octets d'horodatage de réception, 2 octets d'estimation d'erreur de l'horodatage de réception, et 1 octet de durée de vie (TTL, *Time To Live*), ou de limite de bonds dans IPv6 :



Les enregistrements de paquet sont envoyés dans le même ordre que celui de réception réelle des paquets. Donc, les données sont dans l'ordre d'arrivée.

Noter que les paquets perdus (si des pertes ont été détectées durant la session OWAMP-Test) DOIVENT apparaître dans la séquence des paquets. Ils peuvent apparaître soit au point où la perte a été détectée, soit à tout point ultérieur. Les enregistrements de paquet perdus sont distingués comme suit :

+ un horodatage d'envoi rempli avec l'heure d'envoi présumée (comme calculées par le programme d'envoi).

+ une estimation d'erreur d'envoi remplie avec Multiplicateur=1, Échelle=64, et S=0 (voir dans la description de OWAMP-Test la définition de ces quantités et l'explication des formats d'horodatage et d'estimation d'erreur).

+ une estimation d'erreur de réception normale comme déterminée par l'erreur de l'horloge utilisée pour déclarer la perte de paquet. (Il est déclaré perdu si il n'est pas reçu Temporisation après l'heure d'envoi présumée, comme déterminé par l'horloge du receveur.)

+ un horodatage de réception comportant tous les bits à zéro.

+ Une valeur de TTL de 255.

4. OWAMP-Test

Cette Section décrit le protocole OWAMP-Test. Il fonctionne sur UDP, utilisant les adresses IP et numéros d'accès d'envoyeur et receveur négociés durant l'échange Request-Session.

Comme avec OWAMP-Control, OWAMP-Test a trois modes : non authentifié, authentifié, et chiffré. Toutes les sessions OWAMP-Test qui sont engendrées par une session OWAMP-Control héritent de son mode.

Le client OWAMP-Control, le serveur OWAMP-Control, l'envoyeur OWAMP-Test, et le receveur OWAMP-Test peuvent tous être des machines différentes. (Dans un cas typique, on s'attend à ce qu'il n'y ait que deux machines.)

4.1 Comportement de l'envoyeur

4.1.1 Rythme des paquets

Les programmes d'envoi fondés sur des créneaux, décrits précédemment, en conjonction avec l'heure de début de session programmée, permettent à l'envoyeur et au receveur de calculer indépendamment l'un de l'autre la même exacte programmation d'envoi de paquet. Ces programmations d'envoi sont indépendantes pour les différentes sessions OWAMP-Test, même si elles sont gouvernées par la même session OWAMP-Control.

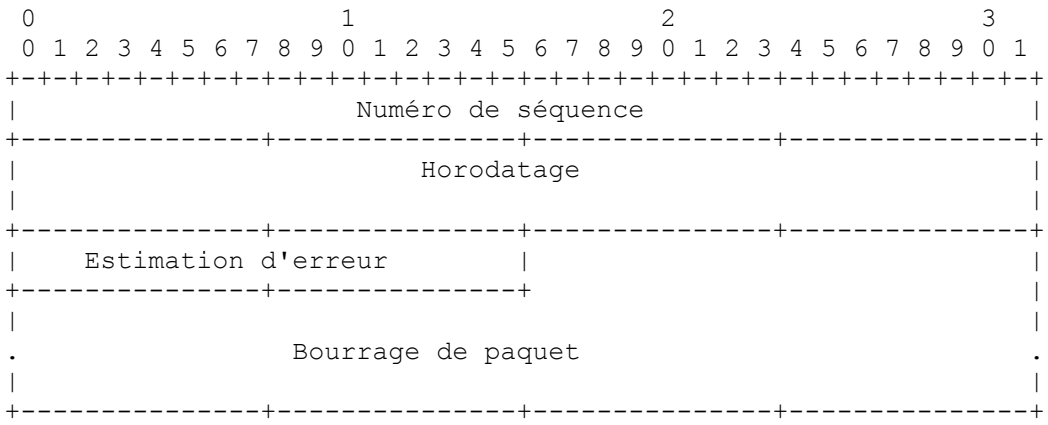
Considérons une session OWAMP-Test. Une fois achevé l'échange Start-Sessions, l'envoyeur est prêt à commencer l'envoi de paquets. Dans les circonstances d'utilisation normales d'OWAMP, l'heure d'envoi du premier paquet est dans le proche futur (peut-être dans une fraction de seconde). L'envoyeur DEVRAIT envoyer des paquets aussi près que possible de leur heure de programmation, avec l'exception suivante : si l'heure d'envoi programmée est passée, et est séparée du temps présent de plus d'une Temporisation, l'envoyeur NE DOIT PAS envoyer le paquet. (Bien sûr, un tel paquet va être considéré comme perdu par le receveur.) L'envoyeur DOIT garder trace des paquets qu'il n'envoie pas. Il va utiliser cela pour dire au receveur quels paquets n'ont pas été envoyés en réglant Gammes sautées dans le message Stop-Sessions de l'envoyeur au receveur à l'achèvement de l'essai. Les Gammes sautées sont aussi envoyées à un client de collecte au titre des résultats des données de session. Ces trous dans le programme d'envoi peuvent se produire si un instant dans le passé a été spécifié dans la commande Request-Session, ou si l'échange Start-Sessions a pris plus de temps que prévu, ou si l'envoyeur n'a pas pu servir à temps la session OWAMP-Test à cause de problèmes internes de programmation du système d'exploitation. Les paquets qui sont dans le passé mais sont séparés du présent de moins de la valeur de Temporisation DEVRAIENT être envoyés aussi rapidenment que possible. Avec les valeurs normales de taux d'essai et de temporisation, le nombre de paquets dans une telle salve est limité. Néanmoins, les hôtes NE DEVRAIENT PAS programmer intentionnellement des sessions pour que de telles salves de paquets se produisent.

Sans considération des délais de programmation, chaque paquet qui est en fait envoyé DOIT avoir la meilleure approximation possible de son heure réelle de départ de son horodatage (dans le paquet).

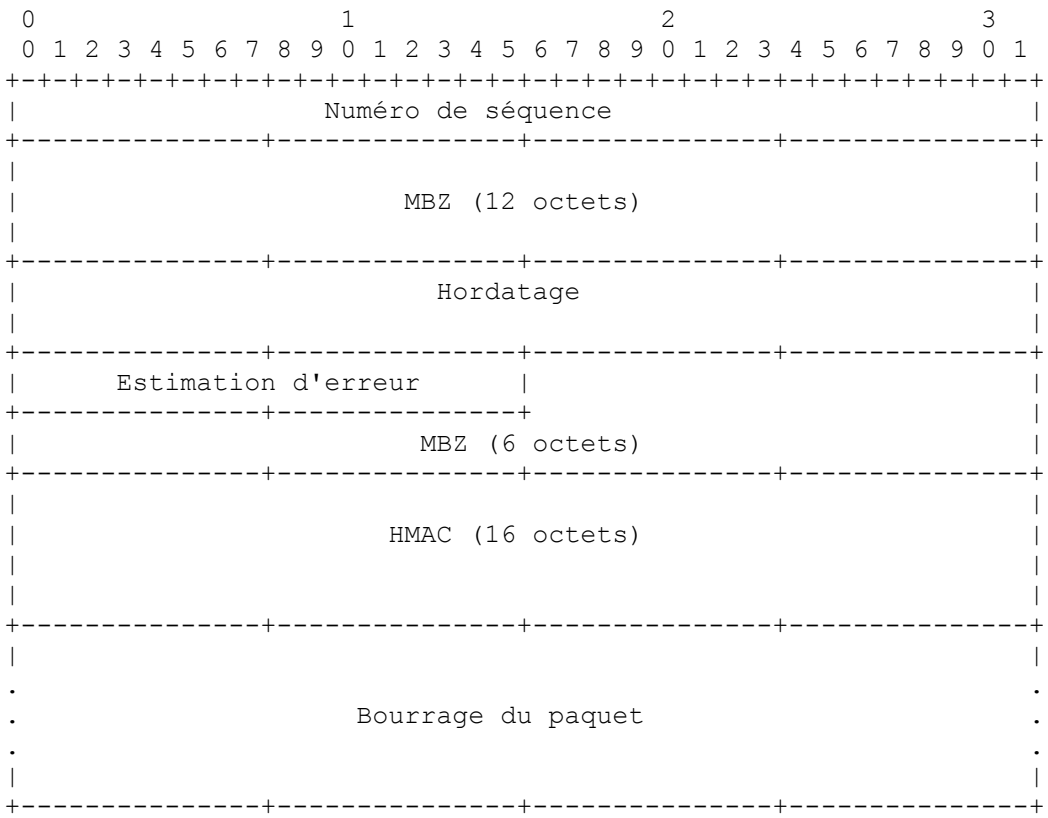
4.1.2 Format et contenu du paquet OWAMP-Test

L'envoyeur envoie au receveur un flux de paquets avec la programmation spécifiée dans la commande Request-Session. L'envoyeur DEVRAIT régler le TTL dans IPv4 (ou la limite de bonds dans IPv6) dans le paquet UDP à l'accès 255. Le format du corps d'un paquet UDP dans le flux dépend du mode utilisé.

Pour le mode non authentifié :

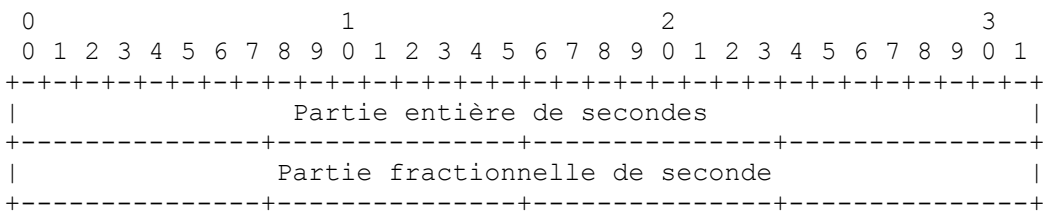


Pour les modes authentifié et chiffré :

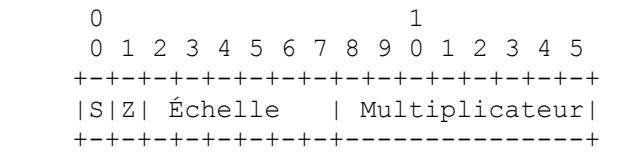


Le format de l'horodatage est le même que dans la [RFC1305] et est comme suit : les 32 premiers bits représentent un entier non signé du nombre de secondes écoulées depuis 0 h le 1er janvier 1900 ; les 32 bits suivants représentent la partie fractionnaire d'une seconde écoulée depuis lors.

Donc, Horodatage est représenté comme suit :



L'estimation d'erreur spécifie l'estimation d'erreur et de synchronisation. Elle a le format suivant :



Le premier bit, S, DEVRAIT être établi si la partie qui génère l'horodatage a une horloge qui est synchronisée à l'UTC en utilisant une source externe (par exemple, le bit devrait être établi si un matériel GPS est utilisé et si il indique qu'il a acquis la position et l'heure courante ou si NTP est utilisé et qu'il indique qu'il s'est synchronisé à une source externe, qui inclut une source de couche 0, etc.). Si il n'y a pas de notion de synchronisation externe pour la source horaire, le bit NE DEVRAIT PAS être établi. Le bit suivant Z a la même sémantique que les champs MBZ ailleurs : il DOIT être mis à zéro par l'envoyeur et ignoré par tout autre. Les six bits suivants, Échelle, forment un entier non signé ; Multiplicateur est aussi un entier non signé. Ils sont interprétés comme suit : l'estimation d'erreur est égale à Multiplicateur*2^(-32)*2^Échelle (en secondes). (Précision de notation : 2^Écnelle est deux à la puissance Échelle.) Multiplicateur NE DOIT PAS être réglé à zéro. Si Multiplicateur est zéro, le paquet DEVRAIT être considéré comme corrompu et éliminé.

Les numéros de séquence commencent à zéro et sont incrémentés de un pour chaque paquet suivant.

La longueur minimum du segment de données est donc de 14 octets en mode non authentifié, et de 48 octets dans les deux modes authentifié et chiffré.

La disposition du paquet OWAMP-Test est la même dans les modes authentifié et chiffré. Les opérations de chiffrement et d'authentification sont cependant différentes. La différence est que en mode chiffré le numéro de séquence et l'horodatage sont tous deux protégés pour fournir une confidentialité et une protection d'intégrité maximum des données, tandis qu'en mode authentifié le numéro de séquence est protégé alors que l'horodatage est envoyé en clair. L'envoi de l'horodatage en clair en mode authentifié permet de réduire le temps entre le moment où un horodatage est obtenu par un envoyeur et celui où le paquet est expédié. En mode chiffré, l'envoyeur doit aller chercher l'horodatage, le chiffrer, et l'envoyer ; en mode authentifié, l'étape du milieu est supprimée, améliorant potentiellement la précision (le numéro de séquence peut être chiffré et authentifié avant d'aller chercher l'horodatage).

En mode authentifié, le premier bloc (16 octets) de chaque paquet est chiffré en utilisant le mode dictionnaire AES (ECB, *Electronic Cookbook*).

De même que pour chaque session OWAMP-Control, chaque session OWAMP-Test a deux clés : une clé de session AES et une clé de session HMAC. Cependant, il y a une différence dans la façon dont les clés sont obtenues : dans le cas de OWAMP-Control, les clés sont générées par le client et communiquées (au titre du jeton) durant l'établissement de connexion dans le message Set-Up-Response ; dans le cas de OWAMP-Test, décrit ici, les clés sont déduites des clés et du SID OWAMP-Control.

La clé de session AES de OWAMP-Test est obtenue comme suit : la clé de session AES de OWAMP-Control (la même clé de session AES qu'utilisée pour la session OWAMP-Control correspondante, où elle est utilisée dans un mode de chaînage différent) est chiffrée, en utilisant AES, avec l'identifiant de session (SID de 16 octets comme clé ; c'est un chiffrement ECB d'un seul bloc ; son résultat est la clé de session AES OWAMP-Test à utiliser pour chiffrer (et déchiffrer) les paquets de la session OWAMP-Test particulière. Noter que toutes les clés de session AES d'OWAMP-Test, les clés de session AES de OWAMP-Control, et le SID comportent 16 octets.

La clé de session HMAC OWAMP-Test est obtenue comme suit : la clé de session HMAC OWAMP-Control (la même clé de session HMAC qu'utilisée pour la session OWAMP-Control correspondante) est chiffrée, en utilisant AES, avec l'identifiant de session de 16 octets que la clé ; c'est un chiffrement de deux blocs CBC, toujours effectué avec IV=0 ; son résultat est la clé de session HMAC OWAMP-Test à utiliser pour authentifier les paquets de la session OWAMP-Test particulière. Noter que toutes les clés de session HMAC OWAMP-Test et les clés de session OWAMP-Control sont de 32 octets, tandis que le SID est de 16 octets.

Le mode ECB utilisé pour chiffrer le premier bloc de paquets OWAMP-Test en mode authentifié n'implique aucun chaînage réel ; de cette façon, les paquets perdus, dupliqués, ou en désordre ne causent pas de problème pour le déchiffrement de paquet dans une session OWAMP-Test.

En mode chiffré, les deux premiers blocs (32 octets) sont chiffrés en utilisant le mode AES CBC. La clé de session AES à

utiliser est obtenue de la même façon que la clé pour le mode authentifié. Chaque paquet OWAMP-Test est chiffré comme un flux distinct, avec juste une opération de chaînage ; le chaînage ne s'étend pas sur plusieurs paquets, de sorte que les paquets perdus, dupliqués, ou en désordre ne causent pas de problème. La valeur d'initialisation pour le chiffrement CBC est une valeur dont tous les bits sont égaux à zéro.

Note de mise en œuvre : naturellement, la programmation de clé pour chaque session OWAMP-Test PEUT être établie seulement une fois par session, pas une fois par paquet.

Le HMAC dans OWAMP-Test couvre seulement la partie du paquet qui est aussi chiffrée. Ainsi, en mode authentifié, HMAC couvre le premier bloc (16 octets) ; en chiffré mode, HMAC couvre les deux premiers blocs (32 octets). Dans OWAMP-Test le HMAC n'est pas chiffré (noter que c'est différent de OWAMP-Control, où le chiffrement en mode flux est utilisé, de sorte que tout ce qui est inclus dans les blocs HMAC finit par être chiffré).

En mode non authentifié, aucun chiffrement ni authentification n'est appliqué.

Le bourrage de paquet dans OWAMP-Test DEVRAIT être pseudo aléatoire (il DOIT être généré indépendamment de tous autres nombres pseudo aléatoires mentionnés dans ce document). Cependant, les mises en œuvre DOIVENT fournir un paramètre de configuration, une option, ou un moyen différent pour faire que le bourrage de paquet consiste en zéros.

Le temps écoulé entre les paquets est calculé en accord avec la programmation de créneaux mentionnée dans la description de la commande Request-Session. À ce point, on saute la question du calcul de nombres pseudo aléatoires à répartition exponentielle d'une façon reproductible. Elle est discutée dans une autre section.

4.2. Comportement du receveur

Le receveur sait quand l'envoyeur va envoyer des paquets. On définit le paramètre suivant : Temporisation (d'après Request-Session). Les paquets qui sont retardés de plus de Temporisation sont considérés perdus (ou "au mieux perdus"). Noter qu'il n'y a jamais d'assurance réelle de perte par le réseau : un paquet "perdu" peut encore être livré à tout moment. La spécification d'origine pour IPv4 exigeait que les paquets soient livrés dans les TTL secondes ou jamais (avec un TTL d'une valeur maximum de 255). À la connaissance des auteurs, cette exigence n'a jamais été réellement mise en œuvre (et, bien sûr, seulement une mise en œuvre complète et universelle assurerait que les paquets ne voyagent pas pendant plus longtemps que TTL secondes). En fait, dans IPv6, le nom de ce champ a été changé en Limite de bonds. De plus, la spécification IPv4 ne propose rien pour le temps que prend au paquet la traversée de la dernière liaison du chemin.

Le choix d'une valeur raisonnable de Temporisation est un problème qui se pose à l'utilisateur du protocole OWAMP, pas au développeur. Une valeur de deux minutes est très sûre. Noter que certaines applications (comme le"ping unidirectionnel" interactif) pourraient souhaiter obtenir les données plus rapidement que cela.

Lorsque les paquets sont reçus :
+ horodatage du paquet reçu ;
+ en mode authentifié ou chiffré, déchiffrement et authentification comme nécessaire (les paquets pour lesquels l'authentification échoue DOIVENT être éliminés) ;
+ mémorisation du numéro de séquence du paquet, heure d'envoi, heure de réception, et TTL pour IPv4 (ou Limite de bonds pour IPv6) provenant de l'en-tête IP du paquet pour que le résultat soit transféré.

Les paquets non reçus dans Temporisation sont considérés perdus. Ils sont enregistrés avec leur vrai numéro de séquence, l'heure d'envoi présumée, la valeur d'heure de réception avec tous les bits à zéro, et un TTL (ou Limite de bonds) de 255.

Les mises en œuvre DEVRAIENT aller chercher la valeur de TTL/Limite de bonds de l'en-tête IP du paquet. Si une mise en œuvre ne va pas chercher la valeur réelle de TTL (la seule bonne raison de ne pas le faire est l'incapacité d'accéder au champ TTL des paquets arrivants) il DOIT enregistrer la valeur de TTL comme 255.

Les paquets qui sont réellement reçus sont enregistrés dans l'ordre d'arrivée. Les enregistrements de paquet perdus servent d'indication de l'heure d'envoi des paquets perdus. Ils DEVRAIENT être placés soit au point où le receveur apprend la perte, soit à tout point ultérieur ; en particulier, on PEUT placer tous les enregistrements qui correspondent aux paquets perdus tout à la fin.

Les paquets qui ont une heure d'envoi au futur DOIVENT être enregistrés normalement, sans changer leur horodatage d'envoi, sauf si ils doivent être éliminés. (Les horodatages d'envoi dans le futur vont normalement indiquer des horloges qui diffèrent de plus que le délai. Certaines données, comme la gigue, peuvent être extraites mais sans connaissance de la

différence d'heure. Pour les autres sortes de données, l'ajustement est mieux traité par le consommateur des données sur la base des informations complètes d'une session de mesures, ainsi que, éventuellement, des données externes.)

Les paquets avec un numéro de séquence déjà observé (paquets dupliqués) DOIVENT être enregistrés normalement. (Les paquets dupliqués sont parfois introduits par les réseaux IP. Le protocole doit être capable de mesurer la duplication.)

Si une des conditions suivantes est vraie, le paquet DOIT être éliminé :
+ l'horodatage d'envoi est de plus que Temporisation dans le passé ou le futur,
+ l'horodatage d'envoi diffère de plus que Temporisation de l'heure où le paquet aurait dû être envoyé d'après son numéro de séquence,
+ en mode authentifié ou chiffré, la vérification du HMAC échoue.

5. Calcul de nombres pseudo-aléatoires à distribution exponentielle

On décrit ici la façon dont sont générées les quantités aléatoires exponentielles utilisées dans le protocole. Bien qu'il y ait un grand nombre d'algorithmes pour générer des variables aléatoires exponentielles, la plupart d'entre elles s'appuient sur une fonction logarithmique comme primitive, résultant en des valeurs potentiellement différentes selon la mise en œuvre de la bibliothèque mathématique. On utilise l'algorithme 3.4.1.S de [KNUTH], qui n'a pas le problème sus-mentionné, et qui garantit le même résultat sur toute mise en œuvre. L'algorithme appartient à la famille ziggurat développée dans les années 1970 par G. Marsaglia, M. Sibuya, et J. H. Ahrens [ZIGG]. Il remplace l'utilisation de la fonction logarithmique par une manipulation habile de bits, qui produit en résultat les dérivées exponentielles.

5.1 Description générale de l'algorithme

Pour faciliter l'exposition, l'algorithme est d'abord décrit avec toutes les opérations arithmétiques interprétées dans leur sens naturel. On donnera ensuite les détails exacts des types de données, de l'arithmétique, et de la génération des dérivées aléatoires uniformes utilisés par l'algorithme. C'est presque une citation mot à mot de la page 133 de [KNUTH].

Algorithme S : soit un nombre réel positif "mu", produire une dérivée aléatoire exponentielle avec "mu" moyen.

D'abord, les constantes Q[k] = (ln2)/(1!) + (ln2)^2/(2!) + ... + (ln2)^k/(k!), $1 \leq k \leq 11$ sont calculées à l'avance. Les valeurs exactes qui DOIVENT être utilisées par toutes les mises en œuvre sont données au paragraphe suivant. C'est nécessaire pour s'assurer qu'exactement les mêmes séquences pseudo aléatoires sont produites par toutes les mises en œuvre.

S1. [Obtenir U et le décalage.] Générer une fraction binaire aléatoire uniforme de 32 bits :
U = (.b0 b1 b2 ... b31) [noter le point binaire]

Localiser le premier bit zéro b_j et décaler les (j+1) bits de tête, en réglant U <- (.b_{j+1} ... b31)

Note : dans le rare cas où le zéro n'a pas été trouvé, il est prescrit que l'algorithme retourne (mu*32*ln2).

S2. [Acceptation immédiate ?] Si U < ln2, régler X <- mu*(j*ln2 + U) et terminer l'algorithme. (Noter que Q[1] = ln2.)

S3. [Minimiser.] Trouver le plus petit $k \geq 2$ tel que U < Q[k]. Générer k nouvelles fractions binaires aléatoires uniformes U1,...,Uk et régler V <- min(U1,...,Uk).

S4. [Livrer la réponse.] Régler X <- mu*(j + V)*ln2.

5.2 Types de données, représentation, et arithmétique

L'algorithme général opère sur des nombres réels, normalement représentés comme des nombres à virgule flottante. Cette spécification prescrit que des entiers non signés de 64 bits soient utilisés à la place.

u_int64_t entiers sont interprétés comme des nombres réels en plaçant la virgule décimale après les 32 premiers bits. En d'autres termes, conceptuellement, l'interprétation est donnée par la transposition suivante :

```
u_int64_t u;
```

u |--> (double)u / (2**32)

L'algorithme produit une séquence de tels entiers u_int64_t qui, pour toute valeur donnée de SID, est garantie d'être la même sur toute mise en œuvre.

On spécifie que les u_int64_t représentations des 11 premières valeurs du dispositif Q dans l'algorithme général DOIVENT être comme suit :

n° 1 0xB17217F8,
n° 2 0xEEF193F7,
n° 3 0xFD271862,
n° 4 0xFF9D6DD0,
n° 5 0xFFF4CFD0,
n° 6 0xFFFEE819,
n° 7 0xFFFFE7FF,
n° 8 0xFFFFFE2B,
n° 9 0xFFFFFFE0,
n° 10 0xFFFFFFFE,
n° 11 0xFFFFFFFF

Par exemple, Q[1] = ln2 est bien approximé par 0xB17217F8/(2**32) = 0,693147180601954 ; pour j > 11, Q[j] est 0xFFFFFFFF.

Le petit entier j dans l'algorithme général est représenté comme u_int64_t valeur j * (2**32).

L'opération d'addition est faite comme usuellement sur des nombres u_int64_t ; cependant, l'opération de multiplication dans l'algorithme général devrait être remplacée par : (u, v) |---> (u * v) >> 32.

Les mises en œuvre DOIVENT calculer le produit (u * v) exact. Par exemple, un fragment de d'arithmétique de 128 bits non signés peut être mis en œuvre à cette fin (voir l'échantillon de mise en œuvre à l'Appendice A).

5.3 Quantités aléatoires uniformes

La procédure pour obtenir une séquence de nombres aléatoires de 32 bits (comme U dans l'algorithme S) s'appuie sur l'utilisation du chiffrement AES en mode compteur. Pour décrire le fonctionnement exact de l'algorithme, on introduit deux primitives provenant de Rijndael. Leurs prototypes et spécifications sont donnés ci-dessous, et sont supposés être fournis par la mise en œuvre Rijndael de prise en charge, telle que [RIJN].

+ Une fonction qui initialise une clé Rijndael avec des octets provenant du germe (le SID va être utilisé comme germe) :

  void KeyInit(unsigned char seed[16]);

+ Une fonction qui chiffre le bloc de 16 octets inblock avec la clé spécifiée, retournant un bloc chiffré de 16 octets. Ici, keyInstance est un type opaque utilisé pour représenter les clés Rijndael :

  void BlockEncrypt(keyInstance key, unsigned char inblock[16]);

Algorithme Unif : étant donné un germe de 16 octets, produire une séquence d'entiers non signés de 32 bits pseudo aléatoires à distribution uniforme. Dans OWAMP, le SID (identifiant de session) provenant du protocole de contrôle joue le rôle de germe.

U1. [Initialiser la clé Rijndael] clé <- CléInit(germe) [Initialiser un compteur de 16 octets (ordre des octets du réseau) non signés] c <- 0

U2. [Besoin de plus d'octets aléatoires ?] Régler i <- c mod 4. Si (i == 0) régler s <- BlockEncrypt(clé, c)

U3. [Incrémenter le compteur comme quantité de 16 octets non signés] c <- c + 1

U4. [Faire le résultat] Sortir le ième quartet des octets de s en commençant par les octets de poids fort, convertis en ordre natif des octets et représenté comme valeur OWPNum64 (comme dans 3.b).

U5. [Boucle] Aller à l'étape U2.

6. Considérations sur la sécurité

6.1 Introduction

Le but du mode authentifié est de protéger par une phrase de passe le service fourni par un serveur OWAMP-Control particulier. On peut imaginer diverses circonstances où ce pourrait être utile. Le mode authentifié est conçu pour interdire le vol de service.

Un objectif supplémentaire de la conception du mode authentifié était de rendre impossible à un attaquanr qui ne peut pas lire le trafic entre envoyeur et receveur de OWAMP-Test d'altérer les résultats d'essais d'une façon qui affecte les mesures, mais pas d'autre trafic.

Le but du mode chiffré est assez différent : rendre difficile à une partie innstallé dans le réseau de faire paraître les résultats "meilleurs" que ce qu'ils devraient être. Ceci est particulièrement vrai si un du client ou du serveur ne coincide pas avec l'envoyeur ou le receveur.

Le chiffrement de OWAMP-Control en utilisant le mode AES CBC avec des blocs de HMAC après chaque message vise à réaliser deux buts : (i) assurer le secret de l'échange, et (ii) assurer l'authentification de chaque message.

6.2 Empêcher le déni de service par un tiers

Les sessions OWAMP-Test dirigées sur une partie qui ne le soupsonne pas pourraient être utilisées pour des attaques de déni de service (DoS). En mode non authentifié, les serveurs DEVRAIENT limiter les receveurs aux hôtes qu'ils contrôlent ou au client OWAMP-Control.

Sauf autrement configuré, le comportement par défaut des serveurs DOIT être de décliner les demandes où le champ Adresse de receveur n'est pas égal à l'adresse d'où la connexion de contrôle a été initiée ou à une adresse du serveur (ou adresse d'un hôte qu'il contrôle). Étant donnée la procédure de prise de contact TCP et les numéros de séquence dans la connexion de contrôle, cela assure que les hôtes qui font ces demandes sont en fait ces hôtes eux-mêmes, ou au moins sur le chemin qui mène à eux. Si cette vérification ou la procédure de prise de contact était omise, il deviendrait possible à des attaquants n'importe où dans l'Internet de demander que de grosses quantités de paquets d'essais soient dirigés sur des nœuds victimes quelque part ailleurs.

Dans tous les cas, les paquets OWAMP-Test avec une certaine adresse de source DOIVENT seulement être envoyés du nœud auquel cette adresse a été allouée (c'est-à-dire que l'usurpation d'adresse n'est pas permise).

6.3 Canaux d'information couverts

Les sessions OWAMP-Test pourraient être utilisées comme canaux couverts d'informations. Les environnements qui se soucient de canaux couverts devraient prendre cela en considération.

6.4 Exigence d'inclure AES dans les mises en œuvre

On remarquera que AES, en mode compteur, est utilisé pour la génération de nombres pseudo aléatoires, de sorte que la mise en œuvre de AES DOIT être incluse même dans un serveur qui ne prend en charge que le mode non authentifié.

6.5 Limitations d'usage de ressources

Un serveur OWAMP peut consommer des ressources de diverses sortes. Les deux plus importantes sortes de ressources sont la capacité du réseau et la mémoire (primaire ou secondaire) pour mémoriser les résultats d'essais.

Toute mise en œuvre de serveur OWAMP DOIT inclure des mécanismes techniques pour limiter l'utilisation de la capacité de réseau et de mémoire. Les mécanismes pour gérer les ressources consommées par des utilisateurs non authentifiés et les utilisateurs authentifiés avec un identifiant de clé et une phrase de passe DEVRAIENT être séparés. La configuration par

défaut d'une mise en œuvre DOIT activer ces mécanismes et régler les limites d'utilisation de ressources à des valeurs prudemment basses.

Une façon de concevoir les mécanismes de limitation de ressource est la suivante : allouer chaque session à une classe d'utilisateur. Les classes d'utilisateur sont partiellement ordonnées avec une relation "inclut", avec une classe ("tous utilisateurs") qui est toujours présente et qui inclut toute autre classe. L'allocation d'une session à une classe d'utilisateur peut se fonder sur la présence de l'authentification de la session, de l'identifiant de clé, de la gamme d'adresses IP, de l'heure, et, peut-être, d'autres facteurs. Chaque classe d'utilisateur va avoir une limite pour l'usage de la capacité réseau (spécifiée en bit/s) et de mémoire pour mémoriser les résultats d'essais (spécifiés en octets). Avec les limites d'utilisation de ressources, l'utilisation courante va être retracée par le serveur. Quand une session est demandée par un utilisateur dans une classe d'utilisateur spécifique, les ressources nécessaires pour cette session sont calculées : l'utilisation moyenne de capacité réseau (sur la base du programme d'envoi) et l'utilisation maximum de mémoire (sur la base du nombre de paquets et du nombre d'octets de chaque paquet) vont devoir être mémorisées en interne – noter que les sessions sortantes ne vont pas exiger d'utilisation de mémoire. Ces nombres d'utilisations de ressource sont ajoutés aux nombres actuels d'utilisation de ressource pour la classe d'utilisateur en cause ; si une telle addition ferait sortir l'utilisation de ressources des limites pour cette classe, la session est rejetée. Quand des ressources sont réclamées, les mesures correspondantes sont soustraites de l'utilisation courante. La capacité de réseau est réclamée aussitôt que la session se termine. La mémoire est réclamée quand les données sont supprimées. Pour les sessions non authentifiées, la mémoire consommée par une session OWAMP-Test DEVRAIT être réclamée après la cloture de la connexion OWAMP-Control qui a initié la session (en douceur ou autrement). Pour les sessions authentifiées, l'administrateur qui configure le service devrait être capable de décider la politique exacte, mais des mécanismes de politique utiles qui PEUVENT être mis en œuvre sont la capacité de réclamer automatiquement la mémoire quand les données sont récupérées et la capacité de réclamer la mémoire après une certaine période configurable (sur la base de la classe d'utilisateur) suivant la terminaison de la session OWAMP-Test.

6.6 Utilisation de primitives de chiffrement dans OWAMP

Au début de la conception du protocole, on a envisagé d'utiliser la sécurité de la couche Transport (TLS) [RFC2246], [RFC3546] et IPsec [RFC2401] comme mécanismes de sécurité cryptographiques pour OWAMP ; plus tard, on a aussi envisagé DTLS. Leurs inconvénients sont les suivants (liste non exhaustive) :

Concernant TLS :

+ TLS pourrait être utilisé pour sécuriser OWAMP-Control fondé sur TCP, mais il serait difficile de l'utiliser pour sécuriser OWAMP-Test fondé sur UDP : les paquets OWAMP-Test, si ils sont perdus, ne sont pas ré envoyés, de sorte que les paquets doivent (facultativement) être chiffrés et authentifiés tout en conservant leur utilité individuelle. TLS fondé sur le flux ne peut pas être facilement utilisé pour cela.

+ Traitant les flux, TLS n'authentifie pas les messages individuels (même dans OWAMP-Control). La façon la plus facile serait d'ajouter un bourrage de format connu à chaque message et de vérifier que le format du bourrage est intact avant d'utiliser le message. La solution perdrait donc un peu de son intérêt ("utiliser simplement TLS"). Il serait aussi beaucoup plus difficile d'évaluer la sécurité de ce schéma avec les divers modes et options de TLS ; ce ne serait très certainement pas sûr du tout. La capacité d'un attaquant à remplacer des parties du messages (à savoir, la fin) avec n'importe quoi pourrait avoir de sérieuses implications de sécurité et devrait être analysé avec soin. Supposons, par exemple, qu'un paramètre utilisé dans une forme de contrôle du débit soit remplacé par n'importe quoi au hasard ; il y a des chances pour que le résultat (un entier non signé) soit assez grand.

+ Selon le mode d'utilisation, on peut finir avec une exigence de certificats pour tous les utilisateurs et une PKI. Même si on accepte qu'une PKI est souhaitable, il n'y en a simplement pas d'utilisable aujourd'hui.

+ TLS exige une très grosse mise en œuvre. OpenSSL, par exemple, est plus gros que notre mise en œuvre de OWAMP total. Cela peut avoir une importance pour les mises en œuvre incorporées.

Concernant DTLS :

+ La duplication et aussi le changement d'ordre sont des phénomène du réseau que OWAMP doit être capable de mesurer ; les mesures anti répétition et la protection contre le changement d'ordre de DTLS empêcheraient les paquets dupliqués et déclassés d'atteindre la partie pertinente du code OWAMP. On pourrait, bien sûr, modifier DTLS afin que ces protections soient affaiblies ou même qu'on spécifie d'examiner les messages dans une séquence soigneusement préparée quelque part entre les vérifications de DTLS ; mais alors, bien sûr, l'avantage d'utiliser un protocole existant ne serait plus réalisé.

+ En mode authentifié, l'horodatage est en clair et n'est absolument pas protégé cryptographiquement, tandis que le reste du message a la même protection qu'en mode chiffré. Ce mode permet un compromis entre la protection cryptographique et la précision de l'horodatage. Par exemple, la mise en œuvre du matériel APAN de OWAMP [APAN] est capable de prendre en charge le mode authentifié. La précision de ces mesures est dans la gamme de la sous micro seconde. Les erreurs de mesures OWAMP de Abilene [Abilene] (faites en utilisant une mise en œuvre de logiciel, dans son mode chiffré) excèdent 10 µs. Les utilisateurs dans des environnements différents ont des soucis différents, et certains peuvent très bien se soucier de chaque dernière micro seconde de précision. En même temps, les utilisateurs dans ces mêmes environnements peuvent se soucier du contrôle d'accès au service. Le mode authentifié leur permet de contrôler l'accès au serveur et d'utiliser des horodatages non protégés, peut-être générés par un matériel.

Concernant IPsec :

+ Ce qu'on appelle maintenant mode authentifié ne serait pas possible (dans IPsec on ne peut pas authentifier une partie d'un paquet).

+ Les chemins de déploiement de IPsec et OWAMP pourrait être séparés si OWAMP ne dépend pas d'IPsec. Après neuf ans d'IPsec, seulement 0,05 % du trafic sur un cœur de réseau évolué, comme Abilene, utilise IPsec (par comparaison avec le chiffrement au dessus de la couche 4, l'utilisation de SSH est de 2 à 4 % et l'utilisation de HTTPS est de 0,2 à 0,6 %). Il est souhaitable d'être capable de déployer OWAMP sur un aussi grand nombre de plateformes différentes que possible.

+ Les problèmes de déploiement d'un protocole dépendant de IPsec vont être particulièrement aigus dans les cas de petits appareils incorporés. Les commutateurs Ethernet, les "modems" DSL, et autres appareils de ce type ne prennent pas IPsec en charge pour la plupart.

+ L'API pour manipuler IPsec à partir d'une application est actuellement mal comprise. Écrire un programme qui doit chiffrer certains paquets, authentifier certains paquets, et en laisser certains ouverts -- pour la même destination – deviendrai plus un exercice de IPsec qu'une mesure IP.

Pour les raisons énumérées, on a décidé d'utiliser un simple protocole de chiffrement (fondé sur un chiffrement de bloc en mode CBC) qui est différent de TLS et d'IPsec.

6.7 Remplacement de la primitive de chiffrement

Il pourrait à l'avenir devenir nécessaire de remplacer AES, ou la façon dont c'est utilisé dans OWAMP, par une nouvelle primitive cryptographique, ou de faire au protocole d'autres changement relatifs à la sécurité. OWAMP fournit un point d'extensibilité bien défini : le mot Modes dans l'accueil du serveur et la réponse Mode dans le message Set-Up-Response. Par exemple, si un simple remplacement de AES par un chiffrement de bloc différent avec un bloc de 128 bits est nécessaire, ceci pourrait être accompli comme suit : on prend deux bits de la partie réservée (MBZ) du mot Modes de l'accueil du serveur ; on utilise un de ces bits pour indiquer le mode chiffré avec le nouveau chiffrement et un autre pour indiquer le mode authentifié avec le nouveau chiffrement. (La consommation de bits pourrait, en fait, être réduite de deux à un, si il est permis au client de retourner un choix de mode avec plus d'un seul bit établi : on pourrait concevoir qu'un seul bit signifie que le nouveau chiffrement est pris en charge (dans le cas du serveur) ou choisi (dans le cas du client) et continuer d'utiliser les bits déjà alloués pour les modes authentifié et chiffré ; cette optimisation est conceptuellement sans importance, mais pourrait être utile en pratique pour faire le meilleur usage des bits.) Alors, si le nouveau chiffrement est négocié, toutes les opérations suivantes l'utilisent simplement à la place de AES. Noter que la séquence normale de transition va être utilisée dans ce cas : les mises en œuvre commenceraient probablement à prendre en charge et préférer le nouveau chiffrement, et ensuite abandonneraient la prise en charge du vieux chiffrement (probablement plus considéré comme sûr).

Si apparaissaît le besoin de faire des changements plus importants (peut-être de remplacer AES par un chiffrement de bloc de 256 bits) ce serait plus difficile et exigerait de changer la disposition des messages. Cependant, le changement peut encore être fait dans le cadre de OWAMP en utilisant l'extensibilité des mots Modes/Mode. La sémantique des nouveaux bits (ou d'un seul bit, si l'optimisation décrite ci-dessus est utilisée) inclurait le changement de la dispositin de message ainsi que le changement de la primitive cryptographique.

Chacun des bits du mot Modes peut être utilisé pour une extension indépendante. Les extensions signalées par les divers bits sont orthogonales ; par exemple, un bit pourrait être alloué au changement de AES-128 à un autre chiffrement, un autre bit pourrait être alloué à l'ajout d'une caractéristique de protocole (comme, par exemple, la prise en charge de mesures en

diffusion groupée) et une autre pourrait être allouée à changer une fonction de déduction de clé, etc. La progression des versions n'est pas d'ordre linéaire, mais plutôt d'ordre partiel. Une mise en œuvre peut utiliser tout sous ensemble de ces caractéristiques (bien sûr, les caractéristiques peuvent être rendues de mise en œuvre obligatoire, par exemple, de nouveaux chiffrements plus sûrs si ils sont nécessaires).

Si un chiffrement avec une taille de clé différente (disons une clé de 256 bits) devenait nécessaire, une nouvelle fonction de déduction de clé pour les clés de OWAMP-Test serait aussi nécessaire. La sémantique du changement de chiffrement DEVRAIT alors être liée à l'avenir à la sémantique du changement de la fonction de déduction de clé (KDF). Une KDF qui pourrait être considérée pour cela pourrait être une fonction pseudo aléatoire (PRF) avec un résultat de taille appropriée, comme 256 bits (peut-être HMAC-SHA256, si il est alors encore considéré être une PRF sûre) qui pourrait alors être utilisé pour déduire les clés de session OWAMP-Test de la clé de session OWAMP-Control en utilisant la clé de session OWAMP-Control comme clé HMAC et le SID comme HMAC de message.

Noter que le schéma de remplacement mentionné ci-dessus est trivialement susceptible d'attaques en dégradation : une partie malveillante sur le chemin peut changer les bits de modes lorsque le mode est négocié afin que le plus ancien et plus faible mode supporté par les deux parties soit utilisé. Si cela est réputé poser problème au moment du remplacement de la primitive cryptographique, le schéma pourrait être augmenté avec une mesure pour empêcher de telles attaques (peut-être en échangeant les modes une fois encore en établissant un canal de communication sûr, en comparant les deux jeux de mots de mode, et en éliminant la connexion si ils ne correspondent pas).

6.8 Clés gérées manuellement à long terme

OWAMP-Control utilise des clés à long terme avec gestion manuelle. Ces clés sont utilisées pour négocier automatiquement les clés de session pour chaque session OWAMP-Control fonctionnant en mode authentifié ou chiffré. Le nombre de ces clés gérées par un serveur s'adapte linéairement avec (et, en fait, est égal au) le nombre d'utilisateurs administrativement différents (peut-être des personnes particulières, des rôles, ou des robots représentant des sites) qui doivent se connecter à ce serveur. De même, le nombre de clés manuelles différentes gérées par chaque client est le nombre des différents serveurs auxquels le client doit se connecter. Cet usage de clés manuelles à long terme est conforme à la [RFC4107].

6.9 (Ne pas) utiliser l'heure comme sel

Une idée naturelle est d'utiliser l'heure courante comme sel quand on déduit les clés de session. Malheureusement, ceci paraît être trop limitatif.

Bien que OWAMP fonctionne souvent sur des hôtes qui ont des horloges bien synchronisées, il est aussi possible de le faire fonctionner avec des horloges complètement décalées. Les délais obtenus ne sont, bien sûr, pas directement utilisables ; cependant, certaines métriques, comme la perte unidirectionnelle, le déclassement, les mesures d'encombrement comme le délai médian moins un minimum, et beacoup d'autres sont utilisables directement et immédiatement (et s'améliorent probablement avec des informations qui auront été fournies par une mesure de délai d'aller-rtour). De plus, même les informations de délai peuvent être utiles avec un post traitement approprié. Bien sûr, on peut même avancer que fonctionner avec des horloges libres et faire subir un traitement ultérieur aux résultats de mesures maillées va donner une meilleure précision, car plus d'informations sont disponibles à postériori et la corrélation de données provenant d'hôtes différents est possible dans un post-traitement, mais pas avec une horloge en ligne.

Ceci étant, l'heure n'est pas utilisée comme sel dans la déduction de clé.

6.10 Utilisation de AES-CBC et HMAC

OWAMP s'appuie sur AES-CBC pour la confidentialité et sur HMAC-SHA1 tronqué à 128 bits pour l'authentification de message. Le choix d'une valeur d'initialisation aléatoire est important pour prévenir une attaque de dictionnaire sur le premier bloc (on devrait aussi noter que, avec sa taille de bloc de 128 bits, AES est plus résistant aux attaques de dictionnaire que ne le sont les chiffrements avec de plus courts blocs ; on utilise de toutes façon une IV aléatoire).

HMAC DOIT se vérifier. Il est crucial de vérifier cela avant d'utiliser le message ; autrement, la falsification existentielle devient possible. Le message complet pour lequel la vérification de HMAC échoue DOIT être éliminé (à la fois pour les courts messages consistant en quelques blocs et pour des messages potentiellement longs, comme une réponse à la commande Fetch-Session). Si un tel message fait partie de OWAMP-Control, la connexion DOIT être éliminée.

Comme les messages OWAMP peuvent avoir des nombres de blocs différents, l'attaque de falsification existentielle décrite dans l'exemple 9.62 de [MENEZES] devient un souci. Pour l'empêcher (et simplifier la mise en œuvre) la longueur de tout message devient connue après le déchiffrement de son premier bloc.

Un cas particulier est celui du premier message (de longueur fixe) envoyé par le client. Là, le jeton est un enchaînement du défi de 128 bits (transmis en clair par le serveur) d'une clé de session AES de 128 bits (générée au hasard par le client, chiffrée avec AES-CBC avec IV=0) et d'une clé de session HMAC-SHA1 de 256 bits utilisée pour l'authentification. Comme IV=0, le défi (un seul bloc de chiffrement) est simplement chiffré avec la clé secrète. Donc, on s'appuie sur la résistance de AES aux attaques de texte en clair choisi (car le defi pourrait avoir été substitué par un attaquant). On devrait noter que le nombre de blocs de texte en clair choisi qu'un attaquant peut avoir chiffré avec la clé secrète est limité par le nombre de sessions que le client veut initier. Un attaquant qui sait le chiffrement du défi d'un serveur peut produire un faux existentiel de la clé de session et donc perturber la session ; cependant, tout attaquant peut perturber une session en corrompant les messages du protocole de façon arbitraire. Donc, aucune nouvelle menace n'est créée ici ; néanmoins, on exige que le serveur ne produise jamais deux fois le même défi. (Si les défis sont générés au hasard, une répétition ne devrait se produire, en moyenne, qu'après 2^64 sessions ; on estime que c'est satisfaisant car c'est assez même pour un improbable serveur actif qui participe à 1 000 000 de sessions par seconde de ne pas avoir de répétitions pour plus de 500 siècles.) À l'égard de la seconde partie du jeton, un attaquant peut produire un faux existentiel de la clé de session en modifiant la seconde moitié du jeton du client tout en laisant la première partie intacte. Ce faux, cependant, va être immédiatement découvert par le client quand le HMAC sur le prochain message du serveur (acceptation ou rejet de la connexion) ne se vérifie pas.

7. Remerciements

Nous tenons à remercier Guy Almes, Mark Allman, Jari Arkko, Hamid Asgari, Steven Van den Berghe, Eric Boyd, Robert Cole, Joan Cucchiara, Stephen Donnelly, Susan Evett, Sam Hartman, Kaynam Hedayat, Petri Helenius, Scott Hollenbeck, Russ Housley, Kitamura Yasuichi, Daniel H. T. R. Lawson, Will E. Leland, Bruce A. Mah, Allison Mankin, Al Morton, Attila Pasztor, Randy Presuhn, Matthew Roughan, Andy Scherrer, Henk Uijterwaal, et Sam Weiler pour leurs commentaires, suggestions, relectures et discussions.

8. Considérations relatives à l'IANA

L'IANA a alloué un numéro d'accès TCP bien connu (861) pour la partie OWAMP-Control du protocole OWAMP.

9. Considérations d'internationalisation

Le protocole ne porte aucune information en langage naturel, à l'exception possible de KeyID dans OWAMP-Control, qui est codé en UTF-8.

10. Références

10.1 Références normatives

[AES] NIST, "Advanced Encryption Standard (AES)", http://csrc.nist.gov/encryption/aes/

[RFC2104] H. Krawczyk, M. Bellare et R. Canetti, "HMAC : Hachage de clés pour l'authentification de message", février 1997.

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997. *(MàJ par* RFC8174)

[RFC2330] V. Paxson, G. Almes, J. Mahdavi, M. Mathis, "Cadre pour la mesure des performances d'IP", mai 1998. *(Information ; MàJ par* RFC8468)

[RFC2474] K. Nichols, S. Blake, F. Baker et D. Black, "Définition du champ Services différenciés (DS) dans les en-têtes IPv4 et IPv6", décembre 1998. *(P.S. ; MàJ par* RFC3168, RFC3260, RFC8436)

[RFC2679] G. Almes, S. Kalidindi, M. Zekauskas, "Métrique de délai unidirectionnel pour IPPM", septembre 1999. *(P.S. ; Remplacée par* RFC7679, STD 81*)*

[RFC2680] G. Almes, S. Kalidindi, M. Zekauskas, "Métrique de perte de paquet unidirectionnelle pour IPPM", septembre 1999. *P.S. ; Remplacée par* RFC7680)

[RFC2836] S. Brim, B. Carpenter, F. Le Faucheur, "Codes d'identification de comportement par bond", mai 2000. (*Obsolète, voir* RFC3140) *(P.S.)*

[RFC2898] B. Kaliski, "PKCS n° 5 : Spécification de la cryptographie fondée sur un mot de passe, version 2.0", septembre 2000. *(Info. ; remplacée par* RFC8018)

[RFC4107] S. Bellovin, R. Housley, "Lignes directrices pour la gestion des clés de chiffrement", juin 2005. (BCP0107)

10.2 Références pour information

[Abilene] "One-way Latency Measurement (OWAMP)", http://e2epi.internet2.edu/owamp/

[APAN] Z. Shu et K. Kobayashi, "HOTS: An OWAMP-Compliant Hardware Packet Timestamper", In Proceedings of PAM 2005, http://www.springerlink.com/index/W4GBD39YWC11GQTN.pdf

[BRIX] Brix Networks, http://www.brixnet.com/

[MENEZES] A. J. Menezes, P. C. van Oorschot, et S. A. Vanstone, "Handbook of Applied Cryptography", CRC Press, réédition révisée avec mises à jour, 1997.

[KNUTH] D. Knuth, "The Art of Computer Programming", vol.2, 3rd edition, 1998.

[RFC1305] D. Mills, "Protocole de l'heure du réseau, version 3, spécification, mise en œuvre et analyse", STD 12, mars992. *(Remplacée par* RFC5905)

[RFC2246] T. Dierks et C. Allen, "Protocole TLS version 1.0", janvier 1999. *(P.S. ; MàJ par* RFC7919)

[RFC2401] S. Kent et R. Atkinson, "Architecture de sécurité pour le protocole Internet", novembre 1998. (*Obsolète, voir* RFC4301)

[RFC3546] S. Blake-Wilson et autres, "Extensions à la sécurité de la couche Transport (TLS) ", juin 2003. *(Obsolète, voir* RFC4366)

[RFC4086] D. Eastlake 3rd, J. Schiller, S. Crocker, "Exigences d'aléa pour la sécurité", juin 2005. (*Remplace* RFC1750) (BCP0106)

[RIJN] "Reference ANSI C Implementation of Rijndael", http://www.esat.kuleuven.ac.be/~rijmen/rijndael/rijndaelref.zip

[RIPE] RIPE NCC Test-Traffic Measurements home, http://www.ripe.net/test-traffic/ .

[SURVEYOR] Surveyor Home Page, http://www.advanced.org/surveyor/ .

[SURVEYOR-INET] S. Kalidindi and M. Zekauskas, "Surveyor: An Infrastructure for Network Performance Measurements", Proceedings of INET'99, juin 1999. http://www.isoc.org/inet99/proceedings/4h/4h_2.htm

[ZIGG] J. H. Ahrens, U. Dieter, "Computer methods for sampling from the exponential and normal distributions", Communications of ACM, volume 15, issue 10, 873-882, 1972. http://doi.acm.org/10.1145/355604.361593

Appendice A. Échantillon de code C pour dérivées exponentielles

Les valeurs dans la matrice Q[] sont les valeurs exactes qui DOIVENT être utilisées par toutes les mises en œuvre (voir les paragraphes 5.1 et 5.2). Cet appendice ne sert qu'à des fins d'illustration.

/* Exemple d'usage : générer un flux de quantités aléatoires exponentielles (moyenne 1) (en ignorant la vérification d'erreur durant l'initialisation). Si une dérivée avec une moyenne mu autre que 1 est désirée, le résultat de cet algorithme peut être multiplié par mu conformément aux règles de l'arithmétique qu'on décrit.

```
** On suppose qu'un "germe" de 16 octets a été initialisé (comme secret partagé dans OWAMP, par exemple)
unsigned char seed[16];

** OWPrand_context next;

** (initialise l'état)
** OWPrand_context_init(&next, seed);

** (génère une séquence de dérivées exponentielles)
** while (1) {
**    u_int64_t num = OWPexp_rand64(&next);
       <faire ici quelque chose avec num>
                ...
** }
*/

  #include <stdlib.h>

  typedef u_int64_t u_int64_t;

  /* (K - 1) est le premier k tel que Q[k] > 1 - 1/(2^32). */
  #define K 12

  #define BIT31   0x80000000UL                  /* Voir si le premier bit dans les 32 bits inférieurs est zéro. */
  #define MASK32(n)       ((n) & 0xFFFFFFFFUL)

  #define EXP2POW32       0x100000000ULL

  typedef struct OWPrand_context {
        unsigned char counter[16];                              /* Compteur (ordre des octets du réseau).*/
        keyInstance key;                                        /* Clé pour chiffrer le compteur.*/
        unsigned char out[16];                                  /* Le bloc chiffré.*/

  } OWPrand_context;

/*
** La matrice a été calculée en accord avec la formule :  Q[k] = (ln2)/(1!) + (ln2)^2/(2!) + ... + (ln2)^k/(k!)
** comme décrit dans l'algorithme S. (Les valeurs ci-dessous ont été multipliées par 2^32 et arrondies à l'entier le plus proche.) Ces valeurs exactes DOIVENT être utilisées afin que des mises en œuvre différentes produisent les mêmes séquences.
*/
  static u_int64_t Q[K] = {
        0,                                              /* Bouche-trou, afin que les indices de la matrice commencent à 1. */
        0xB17217F8,
        0xEEF193F7,
        0xFD271862,
        0xFF9D6DD0,
        0xFFF4CFD0,
        0xFFFEE819,
        0xFFFFE7FF,
        0xFFFFFE2B,
```

```
        0xFFFFFFE0,
        0xFFFFFFFE,
        0xFFFFFFFF
    };

/* cet élément représente ln2 */
    #define LN2 Q[1]

/* Convertir un entier non signé de 32 bits en un nombre u_int64_t. */

    u_int64_t
    OWPulong2num64(u_int32_t a)
    {
        return ((u_int64_t)1 << 32) * a;
    }

/* Fonctions arithmétiques sur nombres u_int64_t. */

/* Addition. */

    u_int64_t
    OWPnum64_add(u_int64_t x, u_int64_t y)

    {
        return x + y;
    }

/* Multiplication. Permet le débordement. Mise en œuvre directe de l'algorithme 4.3.1.M (p.268) de [KNUTH].
    */
    u_int64_t
    OWPnum64_mul(u_int64_t x, u_int64_t y)
    {
        unsigned long w[4];
        u_int64_t xdec[2];
        u_int64_t ydec[2];

        int i, j;
        u_int64_t k, t, ret;

        xdec[0] = MASK32(x);
        xdec[1] = MASK32(x>>32);
        ydec[0] = MASK32(y);
        ydec[1] = MASK32(y>>32);

        pour (j = 0; j < 4; j++)
            w[j] = 0;

        pour (j = 0; j < 2; j++) {
            k = 0;
            pour (i = 0; ; ) {
                t = k + (xdec[i]*ydec[j]) + w[i + j];
                w[i + j] = t%EXP2POW32;
                k = t/EXP2POW32;
                si (++i < 2)
                    continue;
                else {
                    w[j + 2] = k;
                    break;
                }
            }
        }
```

```
        ret = w[2];
        ret <<= 32;
        return w[1] + ret;
  }
```

/* Nourrir le générateur de nombres aléatoires en utilisant une quantié de 16 octets 'seed' (== l'identifiant de session dans OWAMP). Cette fonction met en œuvre l'étape U1 de l'algorithme Unif. */

```
  void
  OWPrand_context_init(OWPrand_context *next, unsigned char *seed)
  {
        int i;

/* Initialise la clé */
        rijndaelKeyInit(next->key, seed);

/* Initialise le compteur avec des zéros */
        memset(next->out, 0, 16);
        pour (i = 0; i < 16; i++)
              next->compteur[i] = 0UL;
  }
```

/* Fonctions de génération de nombres aléatoires. */

/*Générer et retourner une valeur aléatoire uniforme de 32 bits (sauvegardés dans la moitié de moindre poids de u_int64_t). Cette fonction met en œuvre les étapes U2 à U4 de l'algorithme Unif. */

```
  u_int64_t
  OWPunif_rand64(OWPrand_context *next)
  {
        int j;
        u_int8_t  *buf;
        u_int64_t  ret = 0;

/* Étape U2 */
        u_int8_t i = next->compteur[15] & (u_int8_t)3;
        si (!i)
              rijndaelEncrypt(next->key, next->compteur, next->out);

/* Étape U3. Incrémente next.compteur comme une seule quantité de 16 octets dans l'ordre des octets du réseau pour le mode compteur AES. */
        pour (j = 15; j >= 0; j--)
              si (++next->compteur[j])
                    break;

/* Étape U4. Faire le résultat. Les 4 derniers octets de ret contiennent maintenant l'entier aléatoire dans l'ordre des octets du réseau */
        buf = &next->out[4*i];
        pour (j=0; j<4; j++) {
              ret <<= 8;
              ret += *buf++;
        }
        return ret;
  }

/* Générer une dérivée exponentielle avec moyenne 1. */

  u_int64_t
```

```
  OWPexp_rand64(OWPrand_context *next)
  {
        unsigned long i, k;
        u_int32_t j = 0;
        u_int64_t U, V, J, tmp;

/* Étape S1. Obtenir U et décaler */
        U = OWPunif_rand64(next);

        while ((U & BIT31) && (j < 32)) {       /* Décaler jusqu'au premier 0. */
              U <<= 1;
              j++;
        }
/* Supprimer le 0 lui-même. */
        U <<= 1;

        U = MASK32(U);                          /* Garder seulement la partie fractionnelle. */
        J = OWPulong2num64(j);

/* Étape S2. Acceptation immédiate ? */
        si (U < LN2)                            /* retourne (j*ln2 + U) */
              return OWPnum64_add(OWPnum64_mul(J, LN2), U);

/* Étape S3.  Minimiser. */
        pour (k = 2; k < K; k++)
              si (U < Q[k])
                    break;
        V = OWPunif_rand64(next);
        pour (i = 2; i <= k; i++) {
              tmp = OWPunif_rand64(next);
              si (tmp < V)
                    V = tmp;
        }

/* Étape S4. Retourne (j+V)*ln2 */
        return OWPnum64_mul(OWPnum64_add(J, V), LN2);
  }
```

Appendice B. Vecteurs d''essai pour dérivées exponentielles

Il est important que les programmes d'essais générés par les différentes mises en œuvre à partir d'entrées identiques soient identiques. La partie non triviale est la génération de dérivées à répartition exponentielle pseudo aléatoire. Pour aider les mises en œuvre à vérifier l'interopérabilité, plusieurs valeurs d'essais sont fournies. Pour chacune des quatre valeurs de 128 bits de SID données représentées comme des nombres hexadécimaux, 1 000 000 de dérivées à distribution exponentielle de 64 bits sont générées comme décrit ci-dessus. Lorsque elles sont générées, elles sont toutes ajoutées à chaque autre. La somme de toutes les 1 000 000 de dérivées est donnée comme un nombre hexadécimal pour chaque SID. Une mise en œuvre DOIT produire exactement ces nombres hexadécimaux. Pour aider à la vérification de la conversion de ces nombres en valeurs de délai en secondes, des valeurs approximées sont données (en supposant lambda=1). Une mise en œuvre DEVRAIT produire des valeurs de délai en secondes qui soient proches de celles données ci-dessous.

SID = 0x2872979303ab47eeac028dab3829dab2
SUM[1000000] = 0x000f4479bd317381 (1000569,739036 secondes)

SID = 0x0102030405060708090a0b0c0d0e0f00
SUM[1000000] = 0x000f433686466a62 (1000246,524512 secondes)

SID = 0xdeadbeefdeadbeefdeadbeefdeadbeef
SUM[1000000] = 0x000f416c8884d2d3 (999788,533277 secondes)

SID = 0xfeed0feed1feed2feed3feed4feed5ab
SUM[1000000] = 0x000f3f0b4b416ec8 (999179,293967 secondes)

Adresse des auteurs

Stanislav Shalunov
Internet2
1000 Oakbrook Drive, Suite 300
Ann Arbor, MI 48104
mél : shalunov@internet2.edu
WWW: http://www.internet2.edu/~shalunov/

Benjamin Teitelbaum
Internet2
1000 Oakbrook Drive, Suite 300
Ann Arbor, MI 48104
mél : ben@internet2.edu
WWW: http://people.internet2.edu/~ben/

Anatoly Karp
Computer Sciences Department
University of Wisconsin-Madison
Madison, WI 53706
mél : akarp@cs.wisc.edu

Jeff W. Boote
Internet2
1000 Oakbrook Drive, Suite 300
Ann Arbor, MI 48104
mél : boote@internet2.edu

Matthew J. Zekauskas
Internet2
1000 Oakbrook Drive, Suite 300
Ann Arbor, MI 48104
mél : matt@internet2.edu

Déclaration complète de droits de reproduction

Remerciement
Le financement de la fonction d'édition des RFC est fourni par l'activité de soutien administatif (IASA) de l'IETF.