Groupe de travail Réseau
Request for Comments : 4077
Catégorie : Sur la voie de la normalisation

A.B. Roach, Estacado Systems
mai 2005
Traduction Claude Brière de L'Isle

Mécanisme d'accusé de réception négatif pour la compression de signalisation

Résumé
Le présent document décrit un mécanisme qui permet aux mises en œuvre de compression de signalisation(SigComp, *Signaling Compression*) de rapporter des informations d'erreur précises à la réception d'un message qui ne peut pas être décompressé. Ce retour négatif peut être utilisé par le receveur pour faire des ajustements de granularité fine au message compressé avant de le retransmettre, permettant une récupération rapide et efficace des situations d'erreur.

Table des Matières

1. Introduction

La compression de signalisation [RFC3320], souvent appelée "SigComp", définit un protocole pour le transport de messages compressés entre deux éléments de réseau. Une des caractéristiques clés de SigComp est la capacité du nœud envoyeur de demander que le nœud receveur mémorise les objets d'état pour les restituer ultérieurement.

1.1 Problème

Alors que le document "SigComp – Fonctionnement étendu" [RFC3321] définit un mécanisme qui permet la confirmation de la création d'état, l'expérience de fonctionnement du protocole SigComp a montré qu'il y a encore plusieurs circonstances dans lesquelles la vision de l'envoyeur de l'état partagé diffère de celle du receveur. Une liste non exhaustive

détaillant les circonstances dans lesquelles de telles défaillances peuvent se produire figure ci-dessous.

1.1.1 Disposition en compartiments

Dans SigComp, les états mémorisés sont associés à des compartiments. Conceptuellement, les compartiments représentent une instance d'une application distante. Ces compartiments sont utilisés pour limiter la quantité d'état que chaque application distante a la permission de mémoriser. Les compartiments sont créés à réception d'un message SigComp valide provenant d'une application distante. Dans le protocole courant, il est atttendu des applications qu'elles signalent quand elles en ont fini avec un compartiment afin qu'il puisse être supprimé (en utilisant le bit S dans les données de retour demandées).

Malheureusement, attendre des applications qu'elle se comportent bien n'est pas suffisant pour empêcher les états de s'empiler. Des défaillances inattendues de client, des réamorçages, et la perte de connexité peuvent causer le "blocage" de compartiments qui ne sont jamais supprimés. Pour empêcher cette situation, il devient nécessaire de mettre en œuvre un schéma dans lequel les compartiments qui apparaissent comme non utilisés puissent finalement être supprimés.

Alors que les faits qui précèdent rendent une telle pratique nécessaire, éliminer les compartiments sans signalisation explicite peut avoir l'effet collatéral fâcheux que des compartiments actifs soient parfois supprimés. Cela conduit à une vision différente de l'état entre le serveur et le client.

1.1.2 Redémarrage de client

La principale motivation de SigComp était la compression de messages à envoyer sur une interface radio. Par conséquent, la plupart des déploiements de SigComp vont impliquer une unité mobile à l'un des points d'extrémité. Les terminaux mobiles ne sont généralement pas disponibles pendant de longues durées. Les redémarrages de nœuds (dus, par exemple, à l'épuisement de la batterie) vont induire des situations dans lesquelles le serveur réseau pense que le client contient plusieurs états qui ne sont en fait plus disponibles.

1.1.3 Reprise du serveur sur défaillance

De nombreuses applications pour lesquelles SigComp sera utilisé (par exemple, SIP [RFC3361]) utilisent des enregistrements SRV du DNS pour les recherches de serveur. Une caractéristique importante des enregistrements SRV du DNS est leur capacité à spécifier plusieurs serveurs à partir desquels les clients vont choisir au hasard, avec des probabilités déterminées par la pondération de la valeur "q". La raison de la définition de ce comportement pour les enregistrements SRV est de permettre une répartition de charge dans un ensemble de serveurs équivalents, et de permettre aux clients de continuer de fonctionner même si le serveur avec lequel ils communiquent a une défaillance. Lorsque on utilise de protocoles qui se servent des SRV pour une telle répartition, le trafic avec un serveur défaillant est normalement envoyé par le client à un serveur équivalent qui peut servir aux mêmes fins. Du point de vue d'une application, ce nouveau serveur apparaît souvent comme étant le même point d'extrémité que le serveur défaillant, et elle va par conséquent revenir au même compartiment.

Bien que l'état SigComp puisse être dupliqué au sein d'une grappe de serveurs, conserver l'intégrité de tels états exige un processus d'engagement en deux phases qui ajoute beaucoup de complexité au serveur et peut dégrader significativement les performances.

1.2 Solution

Bien que SigComp permette que les paramètres SigComp retournés signalent que tous les états ont été perdus (en réglant "state_memory_size" à 0 pour un message dans la direction inverse) une telle approche donne une solution incomplète au problème. En plus d'effacer un compartiment entier alors qu'un seul état est corrompu ou manquant, cette approche souffre du comportement fâcheux qu'il exige un message dans la direction inverse que l'application distante devra autoriser. Sauf si un mécanisme de sécurité de couche inférieure est employé (par exemple, TLS) cela va normalement signifier qu'un message compressé de niveau application dans la direction inverse doit être envoyé avant que la récupération puisse se faire. Dans de nombreux cas (comme celui des terminaux mobiles fondés sur SIP) ces messages ne seront pas envoyés souvent ; dans d'autres (déploiement de pur client/serveur) ils ne seront jamais envoyés.

La solution proposée à ce problème est un simple mécanisme d'accusé de réception négatif (NACK, *Negative Acknowledgement*) qui permet au receveur de communiquer à l'envoyeur qu'une défaillance s'est produite. Ce NACK contient un code de cause qui communique la nature de la défaillance. Pour certains types de défaillances, le NACK va aussi contenir des détails supplémentaires qui peuvent être utiles dans la récupération de la défaillance.

2. Comportement de nœud

Les paragraphes qui suivent détaillent le comportement des nœuds qui envoient et reçoivent des NACK SigComp. Le format et les valeurs réels sont décrits à la Section 3.

2.1 Transmission normale de message SigComp

Bien que normales sous tous les autres aspects, les mises en œuvre de SigComp qui utilisent le mécanisme de NACK ont besoin de calculer et mémoriser un hachage SHA-1 pour chaque message SigComp qu'elles envoient. Cela doit être mémorisé d'une façon telle que, étant donnée le hachage SHA-1, la mise en œuvre soit capable de localiser le compartiment avec lequel était associé le message envoyé.

En d'autres termes, si quelqu'un renvoie le hachage SHA-1 au compresseur, il doit être capable de trouver le compartiment avec lequel il travaillait lorsque il a envoyé le message avec ce hachage. Cela exige seulement que le compresseur sache avec quel compartiment il travaille lorsque il envoie un message (ce qui est toujours le cas) et que le hachage SHA-1, lorsque il est mémorisé, pointe sur ce compartiment d'une façon ou d'une autre.

2.2 Réception d'un "mauvais" message SigComp

Lorsque la réception d'un message SigComp cause un échec de décompression, le receveur forme et envoie un message NACK SigComp. Ce message NACK contient un hachage SHA-1 du message SigComp reçu qui n'a pas pu être décompressé. Il contient aussi la cause exacte de l'échec de décompression, ainsi que tous les détails supplémentaires qui pourraient aider le receveur du NACK à corriger le problème. Voir à la Section 3 plus d'informations sur le formatage du message NACK et ses champs.

Pour un transport en mode connexion, comme TCP, le message NACK est renvoyé à l'origine du message en échec sur la même connexion.

Pour un transport fondé sur le flux, comme TCP, le délimiteur standard SigComp de 0xFFFF est utilisé pour terminer le message NACK.

Pour un transport sans connexion, comme UDP, le message NACK est renvoyé au générateur du message en échec à l'adresse IP et accès d'où le message a été envoyé. Noter que ceci peut être ou non le même accès sur lequel l'application va normalement recevoir les messages. Pour s'accommoder des mises en œuvre qui utilisent connect() ou des constructions similaires, le NACK sera envoyé de l'adresse et accès IP auquel le message non interprétable a été envoyé. D'un point de vue pratique, il est probablement plus facile de le déterminer en liant des prises d'écoute à une interface spécifique ; cependant, d'autres mécanismes peuvent aussi être employés.

Le comportement spécifié ci-dessus est strictement nécessaire pour toute forme généralement utile de mécanisme de NACK. Dans le cas le plus général, lorsque une mise en œuvre reçoit un message qu'elle ne peut pas décompresser, elle a exactement trois éléments d'information utiles : (1) le contenu du message, (2) une indication de pourquoi le message ne peut pas être décodé, et (3) l'adresse IP et l'accès desquels le message est originaire. Noter qu'aucun des trois ne contient d'indication de où l'application distante écoute les messages, si c'est différent de l'accès d'envoi.

2.3 Réception d'un NACK SigComp

La première action à prendre à réception d'un NACK est de tenter de trouver le message auquel le NACK correspond. Cette recherche est effectuée en utilisant le hachage SHA-1 de 20 octets contenu dans le NACK. Une fois le message correspondant localisé, d'autres opérations sont effectuées sur la base du compartiment qui était associé au message envoyé.

La suite du comportement d'un nœud à réception d'un NACK SigComp dépend de si un transport fiable ou non fiable est utilisé.

2.3.1. Transport non fiable

Lorsque SigComp est utilisé sur un transport non fiable, l'application n'a raisonnablement pas à s'attendre à ce que la couche transport lui livre aucun message particulier. Il est alors de la responsabilité de la couche application de s'assurer que les données sont retransmises comme nécessaire. Dans ces circonstances, le mécanisme NACK s'appuie sur ce comportement pour assurer la livraison du message, et n'effectue jamais de retransmissions au nom de l'application.

Lorsque un NACK est reçu pour un message envoyé sur un transport non fiable, le receveur du NACK utilise les informations contenues pour faire les ajustements appropriés au compresseur associé au compartiment approprié. La nature exacte de ces ajustements est spécifique du schéma de compression utilisé, et va varier d'une mise en œuvre à l'autre. La seule exigence sur ces ajustements est qu'ils doivent avoir pour effet de compenser l'erreur qui a été indiquée (par exemple, en supprimant l'état que le nœud distant indique qu'il ne peut restituer).

En particulier, lorsque un transport non fiable est utilisé, le message original ne doit pas être retransmis par la couche SigComp à réception d'un NACK. La prochaine transmission d'un message initiée par l'application doit plutôt tirer parti des ajustements faits par suite du traitement du NACK.

2.3.2 Transport fiable

Lorsque un transport fiable est employé, l'application fait l'hypothèse de base que tout message qui descend la pile sera retransmis si nécessaire pour assurer que le nœud distant le reçoive, sauf défaillance indiquée par la couche transport. Parce que SigComp agit comme une cale entre la couche transport et l'application, il devient de la responsabilité de la mise en œuvre de SigComp d'assurer que toute défaillance de transmission d'un message soit communiquée à l'application.

Lorsque un NACK est reçu pour un message envoyé sur un transport fiable, la couche SigComp doit indiquer à l'application qu'une erreur s'est produite. En général, l'application devrait réagir de la même façon que pour toute autre erreur de couche transport, comme un rétablissement de connexion TCP. Pour la plupart des applications, cette réaction va initialement être une tentative de réinitialiser et rétablir la connexion, et réinitialiser la transaction qui a échoué. La couche SigComp devrait aussi utiliser les informations contenues dans le NACK pour faire les ajustements appropriés au compresseur associé au compartiment approprié (similaires aux ajustements faits pour un transport non fiable). Donc, si le compartiment n'est pas rétabli par la réinitialisation de la connexion TCP, le prochain message tirera parti des ajustements.

2.4 Détection de la prise en charge du NACK

La détection de la prise en charge du mécanisme NACK peut être bénéfique dans certaines circonstances. Par exemple, avec la définition actuelle de SigComp, l'accusé de réception de la réception d'état est exigé avant qu'un envoyeur puisse faire référence à cet état. Lorsque plusieurs messages sont envoyés avant qu'une réponse soit reçue, le besoin d'attendre de telles réponses peut causer une diminution significative de l'efficacité de la compression de message. Si il sait que le receveur prend en charge le mécanisme NACK, l'envoyeur peut plutôt supposer de façon optimiste que l'état créé par un message envoyé a été créé, et qu'il est permis de le référencer. Si une telle hypothèse se révèle fausse (due, par exemple, à une perte ou décalage de paquet) l'envoyeur peut récupérer à réception d'un NACK.

Afin de faciliter une telle détection, toute mise en œuvre qui va envoyer des messages NACK à l'occasion d'échecs de décompression va indiquer un numéro de version SigComp de 0x02 dans la machine virtuelle de décompresseur universel (UDVM, *Universal Decompressor Virtual Machine*). Les codes d'octet envoyés à un tel point d'extrémité peuvent vérifier le numéro de version, et renvoyer une indication appropriée à leur compresseur comme retour demandé. Excepté pour le mécanisme NACK décrit dans le présent document, les mises en œuvre qui annoncent une version de 0x02 se comportent exactement comme celles qui annoncent un numéro de version de 0x01.

3. Format de message

Les paquets SigComp NACK sont des messages SigComp syntaxiquement valides qui ont été spécifiquement conçus pour être ignorés en toute sécurité par les mises en œuvre qui ne prennent pas en charge le mécanisme NACK .

En particulier, les messages NACK sont formatés comme la seconde variante d'un message SigComp (normalement utilisé pour le chargement du code) avec un champ "longueur de code" de zéro. Les informations de NACK (identifiant de message, cause de l'échec, et détails de l'erreur) sont codées dans la zone "reste du message SigComp", normalement utilisée pour les données d'entrée. De plus, le champ "destination" est utilisé comme identifiant de version pour indiquer quelle version de NACK est employée.

3.1 Champs de message

Le format du message NACK et l'utilisation des champs qu'il contient sont montrés à la Figure 1.

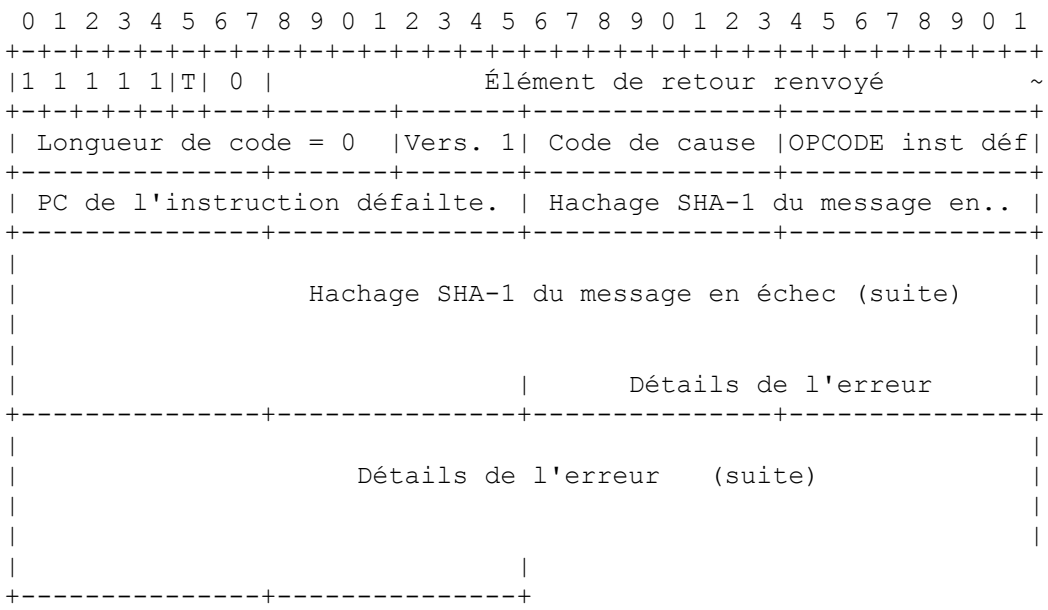

Figure 1 : Format du message NACK SigComp

- "Longueur de code" est le champ "longueur de code" du message SigComp standard. Il est toujours réglé à "0" pour les messages NACK.

- "Version" donne la version du mécanisme NACK employé. Le présent document définit la version 1.

- "Code de cause" est une valeur d'un octet qui indique la nature de l'échec de décompression. Les codes spécifiques sont données au paragraphe 3.2.

- "OPCODE de l'instruction défaillante" est un champ d'un octet qui comporte le opcode sur lequel le PC pointait lorsque la défaillance s'est produite. Si la défaillance s'est produite avant que l'UDVM ait commencé d'exécuter aucun code, ce champ est réglé à 0.

- "PC de l'instruction défaillante" est un champ de deux octets qui contient la valeur du compteur de programme (PC, *program counter*) lorsque la défaillance s'est produite (c'est-à-dire, l'adresse mémoire de l'instruction UDVM défaillante). Le champ est codé avec l'octet de poids fort du PC en premier (c'est-à-dire, dans l'ordre des octets du réseau ou ordre gros boutien). Si la défaillance s'est produite avant que l'UDVM commence l'exécution d'aucun code, ce champ est réglé à 0.

- "Hachage SHA-1 du message en échec" contient le hachage SHA-1 complet de 20 octets du message SigComp qui n'a pas pu être décompressé. Cette information permet au receveur du NACK de localiser le message dont la décompression a échoué afin que des ajustements au compartiment correct puissent être effectués. Quand on effectue ce hachage, le message SigComp entier est utilisé, depuis l'octet d'en-tête (11111xxx en binaire) jusqu'à la fin des entrées. Tous les en-têtes de protocole de couche inférieure (comme UDP ou IP) et les délimiteurs de message (le 0xFFFF qui marque les limites du message dans les protocoles de flux) ne sont pas inclus dans le hachage. Lorsque utilisé sur un protocole fondé sur le flux, toutes les séquences d'échappement 0xFFxx sont "dés-échappées" avant d'effectuer l'opération de hachage.

- "Détails de l'erreur" donne des informations supplémentaires qui pourraient être utiles pour corriger le problème qui a causé l'échec de décompression. Sa signification est spécifique du "Code de cause". Voir au paragraphe 3.2 les informations spécifiques sur ce qui apparaît dans ce champ.

3.2 Codes de cause

Noter que beaucoup des codes d'état sont plus utiles pour déboguer des problèmes d'interopérabilité que pour la correction d'erreurs au vol. L'erreur "STATE_NOT_FOUND" est une exception notable : elle va généralement faire que le receveur du NACK va coder les messages futurs de façon à ne pas utiliser l'état indiqué.

À réception des autres messages d'état, une mise en œuvre va normalement utiliser un ensemble différent de codes d'octet ; si ce n'est pas une option, elle va envoyer ce message spécifique incompressé.

Erreur	**Détails du code**
STATE_NOT_FOUND *(état pas trouvé)*	1 identifiant d'état (6 à 20 octets)
CYCLES_EXHAUSTED *(cycles épuisés)*	2 Cycles par bit (1 octet)
USER_REQUESTED *(utilisateur exigé)*	3
SEGFAULT *(faute de segment)*	4
TOO_MANY_STATE_REQUESTS *(trop de demandes d'état)*	5
INVALID_STATE_ID_LENGTH *(longueur d'identifiant d'état invalide)*	6
INVALID_STATE_PRIORITY *(priorité d'état invalide)*	7
OUTPUT_OVERFLOW *(débordement du résultat)*	8
STACK_UNDERFLOW *(débordement de pile)*	9
BAD_INPUT_BITORDER *(mauvais ordre des bits d'entrée)*	10
DIV_BY_ZERO *(division par zéro)*	11
SWITCH_VALUE_TOO_HIGH *(valeur de commutation trop élevée)*	12
TOO_MANY_BITS_REQUESTED *(trop de bits demandés)*	13
INVALID_OPERAND *(opérande invalide)*	14
HUFFMAN_NO_MATCH *(pas de correspondance Huffman)*	15
MESSAGE_TOO_SHORT *(message trop court)*	16
INVALID_CODE_LOCATION *(code de localisation invalide)*	17
BYTECODES_TOO_LARGE *(codes d'octet trop grands)*	18 Taille de mémoire (2 octets)
INVALID_OPCODE *(opcode invalide)*	19
INVALID_STATE_PROBE *(sonde d'état invalide)*	20
ID_NOT_UNIQUE *(identifiant non unique)*	21 Identifiant d'état (6 à 20 octets)
MULTILOAD_OVERWRITTEN *(écrasement de MULTILOAD)*	22
STATE_TOO_SHORT *(état trop court)*	23 Identifiant d'état (6 à 20 octets)
INTERNAL_ERROR *(erreur interne)*	24
FRAMING_ERROR *(erreur de tramage)*	25

Seules les cinq erreurs "STATE_NOT_FOUND", "CYCLES_EXHAUSTED", "BYTECODES_TOO_LARGE", "ID_NOT_UNIQUE", et "STATE_TOO_SHORT" contiennent des détails ; pour tous les autres codes d'erreur, le champ "Détails d'erreur" a la longueur zéro.

Figure 2 : Codes de cause de NACK SigComp

1. STATE_NOT_FOUND : un état référencé n'est pas trouvé. L'état peut avoir été référencé par l'UDVM en exécutant une instruction STATE-ACCESS ; il peut aussi avoir été référencé par le champ "identifiant d'état partiel" dans un message SigComp. Le champ "détails" contient l'identifiant d'état de l'état qui n'a pu être trouvé. C'est aussi l'erreur qu'il est approprié de retourner dans le cas où un élément d'état unique a été rencontré mais où moins d'octets de l'identifiant d'état qu'exigé par la longueur d'accès minimum ont été envoyés.

2. CYCLES_EXHAUSTED : la décompression du message a pris plus de cycles qu'il ne lui en a été alloué. Le champ "détails" contient une valeur d'un octet qui communique le nombre de cycles par bit. Les cycles par bit sont représentés par un entier non signé (c'est-à-dire, non codé) de 8 bits.

3. USER_REQUESTED : le opcode DECOMPRESSION-FAILURE a été exécuté.

4. SEGFAULT : une tentative de lecture ou d'écriture sur la mémoire en dehors de l'espace mémoire de l'UDVM a eu lieu.

5. TOO_MANY_STATE_REQUESTS : plus de quatre demandes d'objets d'état de mémorisation ou de suppression ont été faites.

6. INVALID_STATE_ID_LENGTH : une longueur d'identifiant d'état de moins de 6 ou de plus de 20 a été spécifiée.

7. INVALID_STATE_PRIORITY : une priorité d'état de 65 535 a été spécifiée pour tenter de mémoriser un état.

8. OUTPUT_OVERFLOW : le message décompressé est trop grand pour être décodé par le nœud receveur.

9. STACK_UNDERFLOW : une tentative de sauter une valeur de la pile UDVM a été faite avec une valeur stack_fill de 0.

10. BAD_INPUT_BITORDER : une instruction INPUT-BITS ou INPUT-HUFFMAN a été rencontrée avec le registre "input_bit_order" réglé à une valeur invalide (c'est-à-dire, un des 13 bits supérieurs est établi).

11. DIV_BY_ZERO : un opcode DIVIDE ou REMAINDER a été trouvé avec un diviseur de 0.

12. SWITCH_VALUE_TOO_HIGH : l'entrée d'un opcode SWITCH excède le nombre de branches défini.

13. TOO_MANY_BITS_REQUESTED : une instruction INPUT-BITS ou INPUT-HUFFMAN qui tente d'entrer plus de 16 bits a été rencontrée.

14. INVALID_OPERAND : un opérande pour une instruction n'a pu se résoudre en une valeur d'entier (par exemple, un opérande littéral ou de référence commençant par 11111111).

15. HUFFMAN_NO_MATCH : la chaîne d'entrée ne correspond à aucun des codes binaires dans le opcode INPUT-HUFFMAN.

16. MESSAGE_TOO_SHORT : en tentant de décoder un message SigComp, le receveur a déterminé qu'il n'y avait pas assez d'octets dans le message pour qu'il soit valide.

17. INVALID_CODE_LOCATION : le champ "localisation de code" dans le message SigComp a été réglé à la valeur invalide de 0.

18. BYTECODES_TOO_LARGE : les codes d'octet qu'un message SigComp a tenté de charger excèdent la quantité de mémoire disponible dans l'UDVM receveuse. Le champ "'détails" est une expression sur deux octets de la DECOMPRESSION_MEMORY_SIZE de la UDVM receveuse. Cette valeur est communiquée avec l'octet de poids fort en premier.

19. INVALID_OPCODE : l'UDVM a tenté d'identifier une valeur d'octet indéfinie comme une instruction.

20. INVALID_STATE_PROBE : en tentant de restituer l'état, l'opérande state_length est réglé à 0 mais l'opérande state_begin n'est pas zéro.

21. ID_NOT_UNIQUE : un identifiant d'état partiel qui a été utilisé pour accéder à l'état correspond à plusieurs éléments d'état. Noter que cette erreur peut être retournée par suite de l'exécution d'une instruction STATE-ACCESS ou par suite de la tentative de localiser un élément d'état unique comme identifié par le "identifiant d'état partiel" dans un message SigComp. Le champ "détails" contient l'identifiant d'état partiel qui était demandé.

22. MULTILOAD_OVERWRITTEN : une instruction MULTILOAD a tenté de s'écraser elle-même.

23. STATE_TOO_SHORT : une instruction STATE-ACCESS a tenté de copier plus d'octets d'un élément d'état que l'élément d'état n'en contient réellement. Le champ "détails" contient l'identifiant d'état partiel qui a été demandé. Les mises en œuvre ne doivent retourner que l'identifiant d'état partiel qui a été demandé ; si le NACK contient des identifiant d'état en plus de ce qui était demandé, un attaquant sera capable d'utiliser ces informations supplémentaires pour accéder à l'état.

24. INTERNAL_ERROR : l'UDVM a rencontré une condition inattendue qui empêche la décompression du message.

25. FRAMING_ERROR : l'UDVM a rencontré une erreur de tramage (un 0xFF 80 .. 0xFF FE non entre guillemets dans un flux d'entrée.) Cette erreur n'est applicable qu'aux messages reçus sur un transport de flux. Dans le cas d'erreur de tramage, un hachage SHA-1 pour un message unique ne peut pas être déterminé. Par conséquent, lorsque un NACK FRAMING_ERROR est envoyé, le champ "hachage SHA-1 de message en échec" devrait être réglé tout à zéro.

4. Considérations sur la sécurité

4.1 Attaques de réflecteur

Comme, par nécessité, les messages SigComp NACK sont envoyés en réponse à d'autres messages, il est possible de les déclancher en envoyant intentionnellement des messages mal formés à une mise en œuvre SigComp avec une adresse IP simulée. Cependant, comme de telles actions peuvent seulement générer un message pour chaque message envoyé, elle ne servent pas d'amplificateur d'attaque. De plus, du fait de la taille raisonnablement petite des paquets NACK, cela ne peut pas augmenter significativement la taille du paquet généré.

On peut noter que presque tous les protocoles déployés affichent le même comportement.

4.2. NACK simulés

Bien qu'il soit possible de faire de faux messages NACK comme si ils étaient générés par un nœud différent, le dommage qui peut en être infligé serait minimal. Le rapport d'une perte d'état ne va normalement résulter en rien de plus que la retransmission de cet état dans le message suivant. D'autres codes de défaillance résulteraient en ce que le prochain message soit envoyé en utilisant un autre mécanisme de compression, ou éventuellement non compressé.

Bien que toutes les conséquences ci-dessus résultent en des messages légèrement plus grands, aucune d'entre elles n'a d'implications particulièrement catastrophiques pour la sécurité.

5. Considérations relatives à l'IANA

Le présent document définit une nouvelle valeur pour l'attribut SigComp_version enregistré par l'IANA.
Valeur (en hex) : 02
Description : SigComp version 2 (prise en charge de NACK)
Référence : [RFC4077]

6. Remerciements

Merci à Carsten Bormann, Zhigang Liu, Pekka Pessi, et Robert Sugar de leurs commentaires et suggestions. Un merci particulier à Abigail Surtees et Richard Price pour plusieurs relectures très détaillées et leurs suggestions.

7. Références

7.1 Références normatives

[RFC3320] R. Price, et autres, "Compression de signalisation (SigComp)", janvier 2003. (*MàJ par* RFC4896) *(P.S.)*

[RFC3321] H. Hannu et autres, "Compression de signalisation (SigComp) - Opérations d'extension", janvier 2003. (*MàJ par* RFC4896) *(P.S.)*

7.2 Références pour information

[RFC3361] H. Schulzrinne, "Option du protocole de configuration dynamique d'hôte (DHCP-pour-IPv4) pour les serveurs du protocole d'initialisation de session (SIP)", août 2002. *(P.S.)*

Adresse de l'auteur

Adam Roach
Estacado Systems
17210 Campbell Road
Suite 250
Dallas, TX 75252
US

mél : adam@estacado.net

Déclaration complète de droits de reproduction

Remerciement
Le financement de la fonction d'édition des RFC est actuellement fourni par la Internet Society.