IDS – Système de Détection d'Intrusion, Partie II

par Klaus Müller
<Socma(at)gmx.net>

L´auteur:
Pour le moment, Klaus Müller (aussi connu sous le nom de 'Socma') est toujours étudiant. Il s'occupe avec de la programmation sous Linux et les sujets liés à la sécurité.

Traduit en Français par:
Jean–Etienne Poirrier (homepage)

Résumé:

Dans la première partie, nous nous sommes concentrés sur les attaques types sur les Systèmes de Détection d'Intrusion (IDS, Intrusion Detection Systems). La seconde partie introduit les méthodes pour les découvrir et nos réponses : l'application de signatures et de filtres sont parmi celles–ci. Finalement, nous introduirons Snort et LIDS.

Les possibilités d'analyse

Précédemment, nous nous sommes étendus sur les attaques pour protéger les IDS et les différents systèmes pour les contrer. Nous avons ensuite couvert les méthodes d'analyse et la manière dont un IDS détermine si c'était une attaque ou pas; respectivement si l'attaque a été couronnée de succès ou non.

À la base, nous faisons la différence entre la Détection d'Abus (Misuse Detection) et la Détection d'Anomalie (Anomaly Detection). La Détection d'Abus utilise des motifs définis spécifiques pour démasquer une attaque. Ces motifs sont appelés des « signatures », elles seront détaillées dans une section dédiée spécifique. Pour le moment, nous avons besoin de savoir que nous pouvons définir des signatures qui cherchent une certaine chaîne (par exemple « /etc/passwd ») dans le trafic réseau, refusent les demandes d'accès à des fichiers spécifiques et envoient une alerte. L'avantage de la Détection d'Abus est la faible probabilité de fausses alarmes puisque les critères de signatures peuvent être définis de manière précise. Les désavantages sont aussi évidents : les nouvelles attaques passent fréquemment sans être détectées faute de définition (voir la section ... sur les signatures).

L'autre méthode est la Détection d'Anomalie. Cela signifie simplement qu'un profil des activités normales de l'utilisateur a été créé. Si le comportement de l'utilisateur dévie trop de ce profil, une alarme est déclenchée. La première étape de cette analyse est la création de profils (base de données) des activités « normales » de l'utilisateur. Un grand nombre d'étapes peuvent être enregistrées : Combien de fois l'utilisateur exécute–t–il

des commandes spécifiques ? Quand exécute−t−il des commandes spécifiques ? Combien de fois ouvre−t−il des fichiers spécifiques ? ... Un petit exemple : l'utilisateur « Exemple » exécute */bin/su* trois fois par jour (cette valeur sera dans le profil). De manière soudaine, un jour, l'utilisateur « Exemple » exécute *su* sept fois dans la journée, plus de deux fois plus souvent que la normale. La Détection d'Anomalie détectera ce comportement « anormal » et avertira l'administrateur au sujet des sept exécutions de *su* de l'utilisateur « Exemple » alors que trois sont la moyenne « normale ». Les désavantages de cette procédure me sont apparus clairement lorsque j'ai commencé son implémentation (voir l'exemple de COLOID, à la fin). La méthode pour installer la base de données des activités de l'utilisateur, requiert des calculs intensifs. Nous surveillons, par exemple, combien de fois l'utilisateur a ouvert dix fichiers spécifiques. Avec chaque commande *open()*, on doit vérifier si cela concerne un des dix fichiers spécifiques et si le résultat est positif, le compteur correspondant est incrémenté. Néanmoins, c'est une belle opportunité de découvrir de nouvelles techniques d'attaques puisqu'elles apparaîtront très probablement comme « anormales ». En outre, l'administrateur lui−même peut définir quelle déviation doit être considérée comme « anormale », par exemple une déviation de 10% ou même 75%... En utilisant cette méthode, nous devons faire attention à générer les profils d'utilisateur dans un réseau « sain », sinon le comportement de l'attaquant sera considéré comme normal et la conduite d'un utilisateur légitime comme anormale.

En général, la Détection d'Anomalie inclut les procédures suivantes :

- Détection de Seuil : dans cette zone, les compteurs sont utilisés, ils comptent combien de fois ce qui est exécuté, ouvert, démarré... Cette analyse statique peut être augmentée par la Détection « Heuristique » de Seuil.
- Détection basée sur des Règles : basée sur des règles définies, que l'usage dévie de ces règles et une alarme est déclenchée.
- Mesure Statique : le comportement de l'utilisateur et du système se conforme à une signature qui a été soit pré−définie, soit établie par d'autres moyens. Un programme notant l'activité normale de l'utilisateur pour la définition de signatures est souvent inclus.

La Détection « Heuristique » de Seuil, dans ce cas, signifie que le compteur (combien de fois a été exécuté ce qui peut l'être) n'est pas initialement défini à une valeur statique mais dynamique. Qu'un utilisateur exécute normalement */bin/login/* quatre fois, le compteur pourrait être défini alors à cinq ...
La Détection d'Anomalie de Protocole représente un sous−groupe de la Détection d'Anomalie. C'est une technique relativement récente ; elle fonctionne, à la base, comme la Détection d'Anomalie. Chaque protocole a une signature « pré−définie » (voir les RFCs correspondantes). Le but de la Détection d'Anomalie de Protocole est de trouver si le comportement du protocole est comme celui pré−défini ou non. Plus d'attaques sont basées sur l'abus de protocole qu'on pourrait le croire ; ce sous−groupe est donc assez important pour les IDS. En regardant de nouveau la section sur le balayage (scanning), nous pouvons trouver quelques indicateurs pour la Détection d'Anomalie de Protocole.

- Vérifier si aucun drapeau n'est défini (NULL scan)
- Vérifier si tous les drapeaux sont définis (XMas scan)
- Vérifier si des combinaisons dépourvues de sens sont définies, comme SF
- ...

Dans les RFC correspondantes, nous trouverons les spécificités correctes ainsi que les types de comportements qui ne devrait pas se produire et quel type de comportement devrait avoir lieu en réponse à un événement spécifique. En outre, il y a la Détection d'Anomalie d'Application (qui fonctionne presque comme les IDS basés sur Application). Dans la littérature, j'ai trouvé plus d'indications sur le sujet et je l'ai approfondi. Bien sûr, un programme a un comportement « normal », par exemple : comment réagit−t−il à l'événement X, Y ou si un utilisateur note quelque chose d'erroné. Souvent, les binaires existant (par exemple, *ps, netstat*, etc.) sont remplacés par l'entrée de l'utilisateur, pour cacher des processus spécifiques dans le cas de *ps*. Avec la Détection d'Anomalie d'Application, il serait possible de détecter le comportement « anormal »

d'un programme. Quelques IDS basés sur application fonctionnent ainsi, mais j'ai peu d'expérience avec eux.

Finalement, voici une autre nouvelle méthode parmi les IDS : la Prévention d'Intrusion. Elle est appliquée par quelques nouveaux IDS et diffère des précédentes méthodes de détection d'intrusion décrites précédemment. A la place d'analyser les *logfiles/* du trafic c'est−à−dire découvrir les attaques après qu'elles se soient déroulées, la Prévention d'Intrusion essaie de prévenir ces attaques.

Contrairement aux IDS classiques, aucune signature n'est utilisée pour détecter les attaques. Ce qui suit est une courte explication sur la manière dont travaille ces IDS. Leurs fonctionnalités devraient devenir claires avec les exemples suivants :

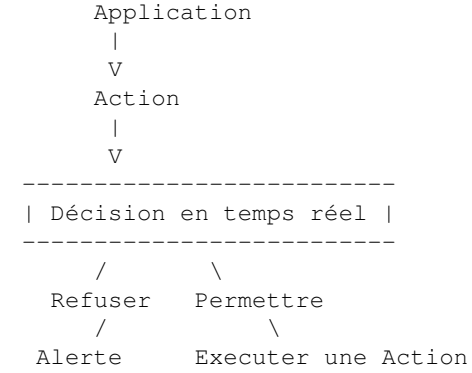
- Surveillance du comportement d'application
- Création de règles d'application
- Alerte suite aux violations
- Corrélation avec d'autres événements
- Interception d'appels au système
- ...

La « surveillance du comportement d'application » se rapproche des IDS basés Application c'est−à−dire que le comportement de l'application est analysé et noté (quelles données sont normalement demandées, avec quels programmes elle interagit, quelles ressources sont requises...). Comme la Détection d'Anomalie, elle essaie de trouver la manière dont un programme opère normalement, ce qu'il est permis de faire.

La troisième fonctionnalité (« Alerte suite aux violations ») ne nécessite aucune explication : cela signifie qu'une alerte est envoyée seulement en cas de déviation (c'est−à−dire quand une attaque est détectée). Cela peut résulter simplement en une entrée dans un journal ou un blocage de ressources.

Avec la seconde fonctionnalité (« Création de règles d'application »), ce qui est appelé une règle d'application est établi en se basant sur l'information résultant de l'analyse en première partie (« Surveillance du comportement d'application »). Cet ensemble de règles donne des informations sur ce que peut faire une application (quelles ressources peut−elle demander ?) et ce qu'il n'est pas permis de faire à une application.

La « Corrélation avec d'autres événements » signifie un partage d'informations entre des senseurs coopératifs, cela donnant une meilleure protection contre les attaques.



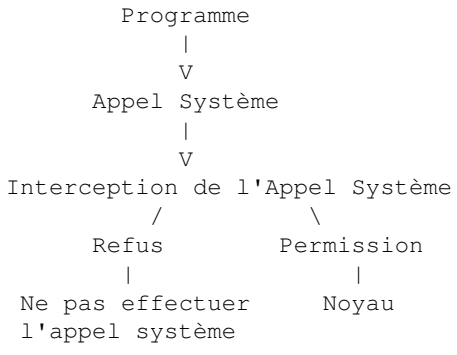
Ce diagramme simplifié devrait clarifer une fois de plus le processus. Avant toute activité, une « décision en

temps réel » est exécutée (c'est–à–dire l'activité est comparée à l'ensemble de règles). Si l'activité est illégale (c'est–à–dire si le programme demande des données ou veut les changer même s'il ne lui est pas permis d'accéder aux données du système), une alarme est donnée. Dans la plupart des cas, les autres senseurs (ou une console centrale) en sera aussi informée. Cela devrait empêcher les autres ordinateurs du réseau d'ouvrir ou d'exécuter des fichiers spécifiques. Si l'activité se conforme à l'ensemble de règles, la permission sera accordée et l'activité aura finalement lieu.

Finalement, voici le dernier item sur notre liste : « Interception d'appels au système ». Manipuler les appels au système (c'est–à–dire ce qu'on appelle les rootkits) sont fréquennement détectés. L'approche de l'interception des appels au système est assez simple : avant qu'un appel au système soit « accepté », on le vérifie complètement. Le vérifier signifie, par exemple, se poser les questions suivantes (voir aussi [5]) :

- qui a demandé l'appel au système (quel programme) ?
- sous quelles autorisations d'utilisateur tourne le processus (root...) ?
- à quoi l'appel système essaie–t–il d'accéder ?

Cela permet la surveillance des essais de changements d'importants fichiers du système ou de la configuration. Nous n'avons qu'à vérifier si l'appel système satisfait l'ensemble de règles prédéfini ou non.



Puisque la Prévention d'Intrusion est relativement nouvelle (par comparaison avec les autres méthodes), plus d'informations sur le sujet seront disponibles.

En conclusion, un conseil pour OKENA, un IPS à fort potentiel. Dans la section des articles (white paper) sur www.okena.com, vous pourrez trouver plus d'informations sur Storm Watch. Pour avoir des informations au sujet des imperfections de Storm Watch, voyez [6].

Signatures

Maintenant, nous allons voir l'utilisation de signatures dans les IDS. La seconde partie va couvrir leurs faiblesses.

Concept

A l'aide des signatures, les attaques connues peuvent être reconnues, une signature correspondant à un certain motif dans les données du traffic. Ce motif peut être une variété d'éléments comme des chaînes, un en−tête remarquable (avec des combinaisons de drapeaux inhabituelles), des ports connus pour être utilisés par des trojans. La plupart des attaques ont certaines caractéristiques, par exemple des drapeaux spécifiques sont définis ou des chaînes dans le corps. À travers les signatures, on essaie de découvrir les attaques à partir de ces caractéristiques.

Je souhaiterais commencer avec ce qu'on appelle les Signatures de Charge Utile (« Payload Signatures »). Voici une partie d'un paquet de charge utile :

```
00 00 00 00 E9 FE FE FF FF E8 E9 FF FF FF E8 4F    ................
0 FE FF FF 2F 62 69 6E 2F 73  68 00 2D 63 00 FF FF .../bin/sh.-c...
```

Comme nous le verrons plus tard dans la description des règles de SNORT, il y a plusieurs options pour effectuer cela. On cherche généralement des chaînes spécifiques (dans Snort, avec « content » ou « content−list ») dans le contenu de la charge utile. Admettons que quelqu'un veuille accéder à, par exemple, un fichier de mot de passe (par exemple */etc/passwd*), on peut chercher dans la charge utile (pour */etc/passwd*) et, si le paquet contient cette chaîne, des contre−mesures peuvent être entreprises comme :

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80        \
(msg:"WEB-MISC/etc/passwd";flags: A+; content:"/etc/passwd"; \
nocase;classtype:attempted-recon; sid:1122; rev:1;)
```

Une autre possibilité est la détection de paquets qui ne comportent pas une chaîne spécifique.

Une autre approche pour se protéger d'un possible débordement de tampon est de contrôler la taille des paquets sur un port spécifique. Il est en général possible de définir le port source, le port de destination et les requêtes émanant de ports spécifiques ou destinés à des ports spécifiques peuvent être interdits. Les chaînes signature sont en général des signatures de corps, par exemple, la recherche d'une chaîne spécifique dans la signature d'une chaîne dans le corps.
Quoi d'autre peut être détecté à partir des signatures ? Chercher des chaînes spécifiques dans la charge utile n'est pas toujours la meilleure solution. Une autre possibilité est de laisser la recherche de signature pour une combinaison de drapeau dans l'en−tête TCP. Si, dans un paquet, le bit SYN tout comme le bit FIN sont définis, cela représente une anomalie qu'un attaquant pourrait utiliser pour trouver des fonctionnalités spécifiques du système d'exploitation (ou le système d'exploitation lui−même). Comme nous l'indiquions au début, il y a des ports spécifiques qui sont connus pour être préférés par les trojans, par exemple les ports 31337 ou 27374.

Peut−être cela deviendra−t−il plus clair si j'explique le processus avec un exemple. Jetons un oeil aux indicateurs typiques d'une attaque synscan :

- IP sources différentes
- Le port TCP Src et Dest est 21

- Type De service 0
- Identifiant d'IP 39426
- SF défini (SYN et FIN)
- Définition de différentes séquences de nombres
- Définition de différents nombres de remerciement (acknowledgment)
- Taille de fenêtre TCP est 1028

Dans des cas comme celui−ci, la fonction de la signature devrait être de pouvoir différencier entre les fonctionnalités « normales » et « anormales » d'une connection. Quelques IDS tiennent des bases de données spéciales avec l'information, comme montré ci−dessus, il vont y chercher les correspondances.
En principe, les anomalies peuvent être détectée dans l'exemple synscan avec la vérification de signature :

- Les ports Source et Destination Port sont 21 (File Transfer Protocol − ftp);
- Un même numéro de port source et de port destination n'indique pas inévitablement l'imminence d'une attaque, seulement une probablilité;
- SF défini, comme mentionné plus haut, cela ne devrait pas se produire puisqu'on ne devrait pas demander une connection et la terminer au même moment;
- Numéro de validation (acknowledgment) n'est pas égal à 0, même si seul SF et pas ACK est défini. Si ACK n'est pas défini, le numéro de validation devrait être 0;
- L'ID d'IP est toujours 39426, même si ce nombre ne devrait pas rester constant (selon la RFC) mais cela n'indique pas inévitablement une attaque synscan : la même chose se produit lors d'une taille constante des fenêtres...

Le développement de la signature pour la détection d'une attaque synscan doit prendre plus de critères que ceux mentionnés ci−dessus. Le but de la signature devrait être la détection d'attaques existantes tout comme celle de nouvelles attaques. Pour cette raison, des caractéristiques générales et spéciales devraient être combinées pour augmenter la probabilité de détecter une attaque. Même s'il serait possible d'écrire une nouvelle signature pour chaque version d'une attaque, cela pourrait nous occuper pour longtemps, nous empêchant de faire des choses plus importantes. Ainsi, nous devons faire attention aux signatures détectant le plus possible d'attaques (et leurs versions), sans nécessité de les éditer.

Les signatures devraient être écrites pour détecter les attaques spécifiques mais on a aussi besoin de signatures générales pour trouver les anomalies. Un exemple de signature pour la détection d'une attaque spécifique est montré ci−dessus (l'attaque de synscan). Une signature plus générale, par exemple, pourrait être la vérification pour les critères suivantes :

- numéro de validation (acknowledgment number) différent de 0 même si ACK n'est pas défini
- combinaisons de drapeaux anormales dans l'en−tête TCP (SYN et FIN) ou d'autres (voir la description des scans)
- ...

Les signatures qui, en général, cherchent des anomalies dans les protocoles sont appelées des « Signatures basées sur l'Analyse de Protocole » alors qu'un autre groupe est connu comme « Packet Grepping ».

Faiblesses

Même si la méthode des signatures de corps (y compris les signatures de chaîne) semble être assez sûre, il y a moyen de les contourner. J'écrirai probablement un papier sur la manière de contourner les règles de Snort mais je vais me limiter ici à l'essentiel. Une signature, comme montré plus haut, recherche « */etc/passwd* », avec la mise en majuscule *nocase* ignorée. Cependant, si nous approchons */etc/passwd* indirectement et pas directement (par exemple si l'attaquant utilise « */etc/blablabla/.../...^passwd* »), est−ce que la signature tirera quand même le signal d'alarme ? Non, parce qu'elle cherche pour */etc/passwd* (et les autres versions en minuscule/majuscule). Une autre limite de ces signatures est la détection des attaques les plus connues et leur recherche des vulnérabilités connues. Les nouvelles versions d'attaques spécifiques ne sont souvent pas découvertes ... D'autres signatures (qui sont spécialisées pour des attaques spécifiques) ou les signatures généralisées ont l'avantages de découvrir ces nouvelles attaques. Cependant, on doit faire attention lors de la création de règles de signature. Une signature qui est spécialisée dans une attaque spécifique ne trouvera pas une variante légèrement modifiée (par exemple : à la place d'une valeur d'ID d'IP de 39426, la nouvelle attaque a une valeur variable ...). En traçant les signatures générales (signatures basées sur l'analyse de protocole), nous devons être sûr que les règles sont réellement définies globalement, c'est−à−dire qu'elles doivent être autorisées à montrer des anomalies qui ne peuvent ou ne devraient pas arriver.

Un autre écueil apparaît à l'examen plus serré de l'attaque Unicode (voir [4]). Voici une description typique d'un trou de sécurité dans MS IIS qui a été permis par l'attaque Unicode :

« Synthèse :
Une faille existe dans Internet Information Server (IIS) de Microsoft ; elle permet aux utilisateurs distants de lister le contenu de répertoires, voir les fichiers, les effacer et d'exécuter des commandes arbitraires. Les attaquants pourraient utiliser l'ensemble de caractères Unicode pour créer des URL afin d'accéder, via IIS, aux ressources qui devraient être normalement inaccessibles. Toutes les versions récentes d'IIS sont affectées par cette vulnérabilité. L'exploitation de celle−ci est triviale. ISS X−Force est au courant de l'exploitation répandue de cette vulnérabilité. »

Un problème pour les IDS réside dans le fait que les caractères en UTF−8 ont plusieurs codes avec le même résultat, par exemple « A » : U+0041, U+0100, U+0102, U+0104, U+01CD, U+01DE, U+8721. Puisque MS IIS n'est pas sensible à la casse, il y a de nouveau de nombreuses possibilités d'afficher plusieurs caractères (par exemple, il y a 83 060 640 possibilités différentes d'afficher AEIOU).

Par exemple, si un attaquant appelle « http://victim/../../winnt/system32/cmd.exe », IIS va générer une erreur. Cependant, en remplaçant "../.." par son équivalent en UTF−8, aucune erreur n'est générée : « http://victim/..%C1%9C../winnt/system32/cmd.exe ». De plus, avec les codes UTF−8 appelés « UN−escape », il est possible d'ouvrir des pages auxquelles nous ne devrions pas avoir accès. Un NIDS a des difficultés pour détecter ces attaques car ces événements se passent au niveau applicatif (dans ce cas, l'application d'un HIDS serait plus approprié). Le cryptage en général représente un problème pour les senseurs − ce fait est fréquemment exploité pendant ce temps. Les démarches de quelques « producteurs d'IDS » montrent clairement les limites des signatures : les signatures détectées disponibles connaissent quelques attaques Unicode mais, après de petites modifications de l'attaque, les signatures deviennent de nouveau inutiles. Seul NetworkICE a développé avec succès des signatures qui découvrent ces types d'attaques (Snort et ISS Real Secure ont cependant développé des signatures qui ont détecté seulement les attaques Unicode connues).

Réponses

Comme expliqué précédemment, les IDS analysent les activités sur le PC et sur le net. Mais à quel point serait utile un IDS s'il ne réagirait pas et ne nous alerterait pas ? Un IDS serait sans valeur, juste un gaspillage de performance machine. À la base, la « réponse » peut être différenciée entre Réponse Active et Réponse Passive. Nous allons expliquer les différences dans une section spécifique.
Le lecteur intéressé peut se référer à l' OPSEC.

```
Sécurisés par Check Point, les appareils sont des solutions de sécurité qui
intègrent la technologie Check Point VPN-1/FireWall-1 sur les plates-formes
matérielles de nos partenaires.
```

Ce système permet l'intégration de systèmes existants avec FireWall−1. Un avantage supplémentaire est sa reconnaissance mondiale (il a environ 300 partenaires). Si vous découvrez une attaque, vous pouvez bloquer l'adresse IP de l'attaquant (seulement comme une option de réponse possible)... Si vous êtes intéressés dans OPSEC, lisez la section « Deployment Platforms », vous y trouverez aussi les conditions pour « joindre » le système (pour devenir un partenaire).

Réponse Active

Une Réponse Active signifie une réaction automatique si l'IDS détecte une attaque (ou un essai d'attaque). En fonction de la sévérité de l'attaque, la plupart des IDS offrent plusieurs options de réponse.
1) Agir contre l'intrus potentiel. 2) « Simplement » collecter des données complémentaires (sur l'intrus et son attaque, ses implications). 3) Changer la configuration
La première réaction possible serait le déclenchement des étapes contre l'intrus. Cela peut inclure une variété d'étapes comme bloquer l'accès de la personne ou déclencher des attaques contre l'intrus. Comme nous l'expliquions en relation avec les pots−de−miel, il est souvent facile de diriger les attaques contre l'intrus mais c'est aussi illégal. Dans ce contexte, l'effet « Tierce Partie » se montre souvent. Mais qu'est−ce que cet effet ? Expliqué graphiquement, l'effet Tierce Partie ressemble à ceci :

```
----------           ------------           --------
| Intrus | --------> | Innocent | --------> | VOUS |
----------           ------------           --------
```

L'effet de Tierce Partie signifie simplement qu'une personne innocente (ou un réseau innocent) est attaqué avec succès par l'intrus et, plus tard, l'intrus utilise ce réseau pour attaquer le nôtre (et probablement d'autres réseaux également). Mais quel est le problème ? Le problème est que notre réseau va catégoriser le réseau « innocent » comme l'attaquant et contre−attaquer en conséquence « innocent » en négligeant « l'intrus » ayant envahi ce réseau pour lancer une attaque contre nous. Résultant de notre attaque (sous la fausse supposition que nous n'attaquerons que les intrus), des dommages indéterminés peuvent être causés à « l'innocent ». Et si « l'intrus » a été assez intelligent pour cacher ses traces, nous serons considérés comme responsables pour ces dommages, pas « l'intrus ».

La seconde option (collecter des informations complémentaires) est moins problématique. Qu'une attaque ou une intrusion potentielle soit détectée et « seulement » des informations complémentaires sur l'utilisateur et son attaque seront collectées. Si un IDS détermine qu'un utilisateur spécifique a étendu avec succès ses privilèges (ou qu'un autre type d'attaque s'est passé), l'observation de cet utilisateur peut être étendue, par

exemple, en enregistrant ses commandes (si cela n'était pas activé auparavant), où cet utilisateur s'introduit, combien de temps il reste, quand s'introduit−il, quand et à quelle fréquence s'introduit−il dans les jours qui suivent, essaie−t−il de transférer (FTP) des binaires spécifiques... De cette manière, un profil de l'attaquant est créé. Avec cela, nous avons l'avantage de pouvoir analyser les journaux (log) en détail, fermer les échappatoires (loopholes) possibles et cela nous permettra aussi de prendre des mesures légales.
Comme troisième option, je vois la modification du système, le pare−feu, etc. Si l'attaquant utilise des adresses IP spécifiques, nous pouvons empêcher cet utilisateur de se connecter au réseau avec ces adresses IP. Bien sûr, nous pouvons bloquer et noter les autres demandes d'accès d'origines suspectes. Dans certains cas spécifiques, nous pouvons simplement bloquer tout accès au réseau lui−même (ou toute requête à des ports spécifiques, à partir d'interfaces réseau spécifiques...). Une autre possibilité de Réponse Active est d'arrêter une connection TCP (connu aussi sous le nom de « TCP kill »). Pour déconnecter un autre ordinateur, nous envoyons un RST (drapeau Reset) : cela « tue » la session. Normalement, RST est envoyé lorsqu'une erreur s'est produite dans la connection... dans ce cas, il peut être utilisé comme IDS (comme dans ISS RealSecure) pour terminer la session avec un autre ordinateur (pour MS−Windows NT, il existe un outil avec le même nom).

```
" tcpkill − tue les connections TCP sur un réseau LAN
  .......
  tcpkill tue les connections TCP en cours spécifiées (utile pour les
  applications basées sur libnids qui requièrent un TCP complet 3−whs pour la
  création de TCB). "
```

C'était un extrait de 'man tcpkill' ...
Comme vous pouvez le voir, il y a de réelles possibilités complètes de réagir aux attaques. Lancer une contre−attaque parait attractif mais ne devrait pas être effectué.

Réponse Passive

En comparaison à la Réponse Active, la plupart du temps, seuls les avertissements (warnings) sont notés et ils doivent être surveillés par l'administrateur / l'utilisateur. Voici les options de réaction :

1) avertissements, conseils... notation
2) génération de rapports qui surveillent les systèmes pour un temps défini et produisent un compte−rendu.

Presque tout IDS est capable de générer des avertissements ou d'envoyer des indications à l'utilisateur / au navigateur. Si, par exemple, on essaie d'effacer un fichier système important, de démarrer des services spécifiques (ceux dont l'usage devrait être interdit)... un avertissement annonçant l'incident peut être généré, indiquant également qui y a pris part et à quel moment. Entretemps, de plus en plus d'IDS ont l'option de générer ces rapports. Le statut d'un système peut être surveillé sur une grande période de temps, les activités peuvent être notées et un rapport de statut peut être généré. Presque tous les IDS fournissent l'option de réponse passive.

Filtrer

Les filtres sont utilisés pour identifier les attaques par leur signature. Cette signature est indirectement reliée aux signatures mentionnées précédemment ; ici, nous cherchons les caractéristiques typiques d'une attaque (comme *ports destination/source*, *IPs destination/source*...). Dans une autre partie de cette section, je vais

introduire et expliquer, au moyen de N−code, quelques exemples de filtres pour des attaques connues. À la fin de cet article, vous trouverez une page (le guide des utilisateurs avancés) qui offre les caractéristiques de N−code, si vous n'êtes pas familier avec lui.
land :

```
# Ceci est un exemple sur la manière de détecter
# une attaque land en N-code
 filter pptp ip () {
        if(ip.src == ip.dest)
        {
                record system.time,
                        eth.src, ip.src, eth.dst, ip.dest
                to land_recdr;
        }
 }
```

Puisque des variables inconnues ont été utilisées, voici une courte explication :

- ip.src = l'adresse IP source
- ip.dest = l'adresse IP destination
- eth.src = l'adresse MAC de la « machine cible »
- eth.dst = l'adresse MAC du « PC source »
- record system.time = note le point dans le temps lorsque la condition ip.src == ip.dest a été remplie

Comme vous pouvez le voir, N−code connait aussi l'opérateur == ; si vous lisez le Guide de l'Utilisateur Avancé, vous trouverez d'autres similarités existantes (avec d'autres langages de haut niveau) comme, par exemple, le fait que N−code connaisse les opérateurs +, −, * ou des opérateurs d'assemblage comme >=, != ou ==, comme montré ci−dessus
Scan Xmas :
Comme vous pourriez vous en souvenir de la section « Types d'Attaques », dans un Scan Xmas, tous les drapeaux sont définis. Ainsi, il devient plausible de vérifier s'ils sont tous définis ou non. Pour cela, nous avons besoin des valeurs des bits définis individuellement :

```
 Bit                    Valeur
 ---------------------------------------
 F-FIN                  1
 S-SYN                  2
 R-RST                  4
 P-PSH                  8
 A-ACK                  16
 U-URG                  32
----------------------------------------
```

filter xmas ip() { if(tcp.hdr) { $dabyte = byte(ip.blob,13); if(!($dabyte ^ 63)) { record system.time, ip.src,tcp.sport,ip.dest, \ tcp.dport, "UAPRSF" to xmas_recorder; return; } } }
Ici, de nouveau, il y a des variables inconnues. Laissez−moi vous les expliquer :

- tcp.hdr = : si tcp.hdr == 0, le paquet ne contient aucun en−tête TCP valide ; si tcp.hdr == 1, il en contient
- tcp.dport = port TCP de destination
- tcp.sport = port TCP source
- ip.blob = contenu de charge d'un paquet (sans en−tête)
- "UAPRSF" signifie que les drapeaux URG,ACK,PSH,RST,SYN et FIN sont définis

$dabyte est une variable locale assignée à l'octet (ip.blob,13). Pour expliquer « l'expression byte », voici une petite démonstration de bits de code TCP :

```
| Src Port | Dest Port | Seq Number | ACK Number | HDR Length | Flags |\
URG | ACK | PSH | RST | SYN | FIN | Win Size | Chksum | Urg Off | Opt |
```

Nous réalisons pourquoi 13 octets dans bytes() ont été spécifiés : parce que 13 octets sont suffisant pour contenir les drapeaux. Avant d'être capable de comprendre comment l'octet fonctionne, voici d'abord quelques remarques sur blob. Si vous vous référez au Chapitre 3 du Guide de l'Utilisateur Avancé, blob est une « séquence de taille arbitraire d'octets ». Un 'octet' retourne un octet de l'offset spécifié d'un blob ; la syntaxe courante ressemble à ceci : byte (str blob_to_search, int offset). Le premier argument spécifie le blob dans lequel il faut chercher (ci−dessus : ip.blob), le second argument défini l'offset (dans 'blob_to_search') de l'octet que l'on recherche. Avec 'if(!($dabyte ^ 63))', on vérifie si tous les drapeaux sont définis ; cela doit donner 63 si on additionne les valeurs de tous les drapeaux (32+16+8+4+2+1). Si quelqu'un souhaite savoir, avec ^, un XOR bit à bit est exécuté

Outre les options mentionnées, N−code offre beaucoup de possibilités complètes. Il est par exemple possible de trouver :

- si le paquet est un paquet IP (ip.is)
- la longueur du paquet IP (ip.len)
- le protocole appliqué au paquet IP (ICMP,TCP or UDP) (ip.protocol)
- la somme de contrôle d'un paquet ICMP (icmp.cksum)
- le contenu de la charge des paquets ICMP (dans un blob) (avec icmp.blob)
- si le paquet contient un en−tête ICMP valide (icmp.hdr)
- si le paquet est un paquet ICMP (icmp.is)
- le type de paquet ICMP comme Echo Reply, Destination inacessible...
- etc...

Vous pourrez trouver des informations additionnelles sur N−code dans le Guide de l'Utilisateur Avancé à : http://www.cs.columbia.edu/ids/HAUNT/doc/nfr−4.0/advanced/advanced− htmlTOC.html

Dans les version futures de ces articles, un bon nombre de filtres seront décrits ; vérifiez s'il n'y a pas de nouvelles versions à intervalles réguliers ;−)

Standards

Dans cette section, je vais vous présenter divers « standards » tout comme des listes / accords qui sont partagés par de nombreux outils « experts ».

CVE

CVE est l'abbréviation de « Common Vulnerabilities and Exposures » (Exposition et Vulnérabilités courantes) qui n'est rien de plus qu'une liste d'expositions et de vulnérabilités. A première vue, cela semble rigolo mais il peut devenir assez pratique plus tard. Différents outils utilisent différents termes pour les vulnérabilités détectées ; en utilisant CVE, une description uniforme des différentes vulnérabilités/expositions peut être

utilisée. Ainsi, il n'est plus nécessaire d'utiliser le même outil que d'autres utilisateurs.

CVE fournit un nom pour une vulnérabilité/exposition spécifique et une description uniforme (et standardisée), empêchant les incompréhensions entre les utilisateurs de différents systèmes. CVE définit une vulnérabilité comme « des problèmes qui sont normalement regardés comme des vulnérabilités dans le contexte de toutes les règles de sécurités raisonnables » et une exposition comme « des problèmes qui sont seulement des violations de quelques règles de sécurité raisonnables ». Dans CVE, la différenciation entre vulnérabilité et exposition est fondamentale. Voici des exemples de vulnérabilités : phf, les fichiers de mots de passe où tout le monde peut écrire... Voici des exemples d'expositions : utilisation de programme pouvant être attaqués par force brute ou l'utilisation de services qui sont attaqués, en général. Par définition et avec ces exemples, il devrait être possible de différencier les vulnérabilités et les expositions (dans CVE). La différence fondamentale pourrait être la capacité de l'attaquant d'appliquer des commandes comme un utilisateur différent ou pouvoir lire/écrire dans des fichiers alors que cela ne devrait pas être possible (à cause des permissions du fichier). Par contre, les exposition permettent à un utilisateur d'extraire des informations supplémentaires à propos du système (et son statut), ces activités tournant en tâche de fond... Les expositions apparaîssent suite à des réglages de sécurité fautifs qui peuvent être « réparés ». Les vulnérabilités peuvent être comprises comme des trous de sécurité dans le système de sécurité « normal » (qui devrait inclure la possibilité de minimiser la menace d'attaquants potentiels grâce à la vérification de permission). Cependant, cette liste doit rester à jour et chacune des vulnérabilités ou expositions n'est pas immédiatement « acceptée ». Après qu'une vulnérabilité / exposition est détectée, elle reçoit un « numéro de candidat » tout d'abord (cela se passe à travers le CNA : Candidate Numbering Authority). De plus, il sera posté sur le Forum (par l'Editeur du CVE Editor) et discuté (est−ce que cette vulnérabilité/exposition doit être acceptée). Si le Forum décide de ne pas accepter le candidat (pour le moment), la raison de ce refus sera postée sur le site web. Si le candidat est accepté, il sera ajouté à la liste (et devient une partie officielle de CVE). Maintenant, il devrait être clair que chaque vulnérabilité (potentielle) reçoit initialement un « numéro de candidat » puisqu'elle doit être discutée pour voir si elle est acceptée ou pas. La vulnérabilité reçoit le 'numéro de candidat' pour la différencier des entrées officielles dans la liste. Chaque candidat possède 3 champs de base (ils « l'identifient ») :

- Numéro
- Description
- Références

Dans ce contexte, le numéro est le nom réel du candidat, composé de « l'année d'apparition », un nombre supplémentaire révèlant quel candidat il représente dans la séquence d'une année :

```
CAN−Année−séquence candidat de l'année
```

Comme indiqué précédemment, un candidat accepté est ajouté à la liste. Il en résulte que la séquence 'CAN−ANNEE−Numéro de candidat' devient 'CVE−Année−Numéro de candidat'. Un exemple : 'CAN−2001−0078' devient 'CVE−2001−0078' dans la liste.

C'est tout ; pour plus d'information, consultez la page web officielle de CVE.

Exemples

Dans cette dernière partie, quelques IDS vont être présentés.

Snort

Puisque Snort est très bien connu et qu'il offre beaucoup d'options, je vais le décrire avec plus de détails que les autres IDS pris en exemple. Fondamentalement, Snort peut être dans un de ces trois modes : Sniffer (Renifleur), Packet Logger (Log de paquets) et Network Intrusion Detection System (Système de Détection d'Intrusion de Réseau). Dans le mode Sniffer, Snort génère des paquets sur la console. Dans le mode Packet Logger, il les note sur le disque dur. Et le mode Network Intrusion Detection permet d'analyser les paquets. Je vais me concentrer principalement sur le dernier mode mais voici une courte introduction aux modes Sniffer et Packer Logger :
En mode Sniffer, une variété d'information sur le paquet peut être lue, comme l'en−tête TCP/IP :

```
[Socma]$ ./snort −v
```

En sortie, nous aurons seulement l'en−tête IP/TCP/ICMP/UDP. Il y a de nombreuses options, seules quelques unes seront mentionnées ici.

```
−d = va livrer le paquet de données
−e = montre le Data Link Layer
```

Mode Packet Logger :
A la différence du mode Sniffer, le mode Packet Logger peut écrire les paquets sur le disque dur. Nous devons seulement assigner un répertoire dans lequel Snort peut les écrire et il va automatiquement passer en mode Packet Logger :

```
#le repertoiredelogging doit exister :
[Socma]$ ./snort −dev −l ./repertoiredelogging
```

Lorsqu'on entre « −l », Snort récupère parfois l'adresse de l'ordinateur distant et l'utilise comme nom de répertoire (dans lequel se trouvent les logs), d'autres fois, il prend l'adresse de l'hôte local. Pour noter le réseau maison (home network), nous devons spécifier ce réseau dans la ligne de commande :

```
[Socma]$ ./snort −dev −l ./repertoiredelogging −h 192.168.1.0./24
```

Une autre possibilité est de noter le log en format TCP−DUMP :

```
[Socma]$ ./snort −l ./repertoiredelogging −b
```

Maintenant, le paquet tout entier sera écrit, pas seulement des sections spécifiques ; cela élimine la nécessité de spécifier des commandes additionnelles. Il est possible d'utiliser des programmes comme tcpdump pour traduire les fichiers en texte ASCII mais Snort peut faire cela aussi :

```
[Socma]$ ./snort −dv −r paquetaverifier.log
```

Mode Network Intrusion Detection : Pour basculer en mode NIDS, nous pouvons utiliser une commande comme celle−ci :

```
[Socma]$ ./snort −dev −l ./log −h 192.168.1.0/24 −c snort.conf
```

Dans ce cas, snort.conf est le fichier de configuration. Il est utilisé pour faire savoir à Snort où il peut trouver ses « règles » pour déterminer s'il y a une attaque ou pas, si la requête doit être permise... Les règles, telles que définies dans snort.conf, seront alors appliquées au paquet pour l'analyser. Si aucun répertoire de sortie spécifique n'a été établi, le répertoire par défaut /var/log/snort est utilisé. Le résultat de sortie de Snort dépend

du mode d'alerte – basé sur ceci, l'information est disponible légèrement plus tôt ou plus tard.

Mode	Comment/Ce qui est affiché
-A fast	Temps, IPs/ports Source et Destination, le Message d'Alerte
-A full	Paramétrage par Défaut
-A unsock	Envoi l'avertissement vers un socket UNIX
-A none	Arrête les alertes

Comme nous l'avons vu, avec l'option –b, nous pouvons noter ce qui se passe en mode binaire et avec l'option –N, la notation de paquets est complètement cessée. Mais ce n'est pas la limite ; par exemple, Snort est capable d'envoyer des messages à syslog. Le règlage par défaut pour cela est LOG_AUTHPRIV et LOG_ALERT. Pour envoyer des messages à syslog, nous devons seulement entrer « –s » (des exemples suivent). En outre, nous avons la possibilité d'envoyer des messages au client smb ou un avertissement pop–up Windows à un ordinateur sous MS–Windows. Pour utiliser cette « fonctionnalité », nous devons entrer « –enable–smbalerts » lors de la configuration de Snort.

```
[Socma]$ ./snort -c snort.conf -b -M MASTATIONWINDOWS
```

Voici un exemple d'application des modes d'alerte :

```
[Socma]$ ./snort -b -A fast -c snort.conf
```

Outre les options décrites, il en existe d'autres comme celles–ci :

```
-D = démarre Snort en mode daemon
-u utilisateursnort = démarre Snort avec l'UID « utilisateursnort »
-g groupesnort = démarre Snort avec le GID « groupesnort »
-d = note aussi les données de la couche d'applications
```

Snort offre de nombreuses options ; si vous rencontrez un problème, entrez juste « snort –h » ou regardez dans les mailing lists si votre problème était apparu autre part. La section suivante couvre les règles de Snort ; si vous n'êtes pas intéressés par la compréhension des règles existantes ou par l'écriture de vos propres règles, vous pouvez passer cette section. Comme je l'indiquais à la fin de cette partie (sur Snort), vous pouvez télécharger le Manuel des Utilisateurs de Snort sur www.snort.org : c'est notre vraie source.

Les règles de Snort :
Pour une meilleure compréhension de Snort, il est obligatoire de connaître les Règles de Snort. Snort utilise parfois des variables spécifiques qui peuvent être définies avec l'utilisation de « var » :

```
var: <name> <wert>      var

MON_REZO [192.168.1.0/24,10.1.1.0/24]
alert tcp any any -> $MON_REZO any (flags:S;msg: "paquet SYN";)
```

Il y a d'autres manières d'entrer le nom d'une variable :

```
$variable = définit la variable Méta
$(variable) = ici, la valeur de la variable « variable » est entré
$(variable:-default) = si « variable » est défini, sa valeur est entrée ici ; si
"variable" n'est pas défini, la valeur « default » est entrée
$(variable:?msg) = entre la valeur de la variable « variable » ou, si indéfinie,
affiche le message « msg »
```

Si vous avez été confronté à la programmation shell auparavant, ce qui suit ne devrait pas vous être étranger :

```
[Socma]$ shelltest=we
[Socma]$ echo hello $shelltestlt
hello
[Socma]$ echo hello ${shelltest}lt
hello world
```

L'application de $(variable) dans Snort et de ${variable} dans le shell est identique. Il y a d'autres équivalents (ou termes semblables) dans la programmation du shell :

```
[Socma]$ shelltest = bash
[Socma]$ echo ${shelltest:-nobash}
bash
[Socma]$ echo ${notdefined:-nobash}
nobash
```

L'application du terme « $(variable:–default) » diffère seulement dans le fait que le shell utilise { et } à la place de (et). Le dernier terme existe aussi dans le shell :

```
[Socma]$ shelltest = bash
[Socma]$ echo ${shelltest:?"then csh"}
bash
[Socma]$ echo ${variablenondefinie:?"pas definie ou nulle"}
pas definie ou nulle
```

Le but de cette courte excursion était « d'associer les connaissances » : j'ai été capable de mémoriser la syntaxe de Snort plus vite en me référant mentalement aux termes du shell dont je me rappellais.
De nombreuses options de la ligne de commande peuvent être définies dans les fichiers de configuration. Pour cela, « config » est utilisé :

```
config <directive> [: <valeur> ]
```

Les « directives » les plus importantes sont :
alertfile = change le fichier dans lequel les alertes sont enregistrées
daemon = démarre le processus comme daemon (–D)
reference_net = défini le réseau maison (–h)
logdir = définit le répertoire pour les logs (–l)
nolog = ferme la création de logs
set_gid = change le GID (–g)
chroot = chrooté dans le répertoire spécifié (–t)
set_uid = défini l'UID (–U)

Si, par exemple, vous voulez changer le fichier d'alerte en, par exemple, « meslogs », vous devez procéder comme suit :

```
config alertfile : meslogs
```

Retour aux règles actuelles (ici un extrait de règles pour le ftp) :

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP EXPLOIT overflow";\
flags: A+; content:"|5057 440A 2F69|";\
classtype:attempted-admin; sid:340;rev:1;)
```

A la base, les règles de Snort consistent en deux parties : l'en−tête de la règle et ses options. L'en−tête de la règle nous informe de trois choses :

- les adresses IP source et destination
- le protocole
- les actions qui doivent être déclenchées par la règle

Dans la règle ftp ci−dessus, l'en−tête est la partie suivante :

```
 Action       ip source          ip destination
    |             |                  |
 alert tcp $EXTERNAL_NET any -> $HOME_NET 21
         |                |                |
        Protocole     de tous ports     Port
```

Comme vous pouvez le voir, l'en−tête de la règle se termine à la première « (» et, après çà, les options de la règles commencent. Il y a plusieurs actions possibles (« alert » dans ce cas) qui peuvent être lancées lorsque les règles de Snort découvrent quelque chose de suspect :

- alert = en fonction de la méthode d'alerte utilisée (par défaut, c'est 'alert full'), une alerte est lancée et le paquet impliqué est tracé
- log = seul le paquet est tracé
- pass = les résultats dans le paquet sont ignorés
- activate = génère une alerte et retourne vers une autre règle dynamique de Snort (bientôt plus sur ce sujet)
- dynamic = la règle reste inactive tant qu'elle est activée par une autre règle ; après, elle fonctionne comme « log » (voir ci−dessus)

Le second champ (protocole ; tcp ici) spécifie quel protocole nécessite une analyse. Les possibilités sont : tcp, udp, icmp et ip (dans le futur, il pourrait en avoir d'autres ... comme ARP, GRE).
En relation avec le champ suivant (ip source), nous trouvons fréquemment l'opérateur « ! » (opérateur de négation).

```
alert tcp !$REZO_EXTERNE any -> $REZO_MAISON 21
```

Il résulte de cet opérateur de négation que chaque paquet qui n'arrive pas du $REZO_EXTERNE est tracé. Il y a des possibilité additionnelles pour entrer un nombre d'adresses IP (c'est−à−dire une liste d'adresses IP). Les adresses doivent être séparées par une virgule et entourées par [ ].

```
alert tcp ![addresse ip,addresse ip] any -> ....
```

Une autre alternative est l'utilisation de « any » qui inclut toutes les adresses IP.

```
log tcp any any -> ...
```

La dernière partie de l'en–tête de la règle est la spécification des ports, ftp dans notre exemple. Il n'est pas seulement possible de surveiller un port spécifique mais également une gamme de ports (plusieurs ports). Voici les options :

```
:numeroport                    -> tous les ports plus petits ou égaux au numeroport
numeroport:                    -> tous les ports plus grands ou égaux au numeroport
debutnumeroport:finnumeroport  -> tous les ports entre debutnumeroport à
                                  finnumeroport (ceux-ci inclus)
```

Bien sûr, il est possible d'utiliser les opérateurs de négations, avec la conséquence que tous les ports seront surveillés à l'exception de celui entré, par exemple.

```
!:21          -> tous les ports qui ne sont pas plus petits ou égal à 21
```

Quelque chose qui n'a pas été expliqué mais qui a été utilisé tout le temps est l'opérateur de direction « –> ».

```
"source" -> "destination"
```

Cependant, il existe une autre variante <> :

```
"source" <> "destination"
```

Cela signifie que Snort va rechercher la source aussi bien que la destination, pour l'adresse.
Comme je l'ai mentionné, il y a l'étape « activate » : elle génère une alerte et retourne vers une autre règle dynamique de Snort.
Si une règle spécifique a complété ses actions, elle peut activer une autre règle. A la base, la différence entre les règles normales et les « règles activées » est le fait que seulement un champ spécifique doit être spécifié : "activates". Les règles dynamiques, d'autre part, fonctionnent comme des logs (voir ci–dessus) ; seule différence : « activate_by » doit être entré. Un champ supplémentaire doit encore être entré : « count ». Après que la « règle activée » ait fait son boulot, la règle dynamique est évoquée mais seulement pour « count » paquets (ce qui signifie « pour 40 paquets » lorsque count = 40).

```
activate tcp !$REZO_MAISON any -> $REZO_MAISON 143 (flags : PA;  \
  content : "|E8C0FFFFFF|\bin|;activates : 1; msg : "IMAP buf!";)
dynamic tcp !$REZO_MAISON any -> $REZO_MAISON 143 (activated_by : 1; \
  count : 50;)
```

Quelques unes des options, comme les options à une règle, n'ont pas encore été présentées. Je vais les expliquer maintenant mais cela prendra plus de sens pour vous plus tard. Veuillez noter que les champs *activates* et *activated_by* (règle dynamique) dans notre exemple plus haut. La première règle invoque une règle dynamique après qu'elle ait fini son travail ; cela est également indiqué par l'assertion « activated_by = 1 » de la règle dynamique.

Maintenant, voici la seconde partie des règles de Snort : les options à la règle. Utilisons encore une fois la première règle (ftp) :

```
alert tcp $REZO_EXTERNE any -> $REZO_MAISON 21 (msg:"FTP EXPLOIT
```

```
overflow"; flags: A+;\
content:"|5057 440A 2F69|";classtype:attempted-admin;\
sid:340; rev:1;)
```

L'option de règle, dans ce cas (l'en–tête de règle s'arrête à la première «) ») est :

```
 (msg:"FTP EXPLOIT overflow";\
flags: A+; content:"|5057 440A 2F69|";\
classtype:attempted-admin; sid:340; rev:1;)
```

Il y a 34 mots clés, je vais limiter mes explications aux plus importantes et/ou aux plus usitées. Pour ceux qui souhaitent avoir un aperçu de tous les mots clés possibles, veuillez jeter un oeil dans le Manuel des Utilisateurs de Snort.

msg – affiche les messages d'alerte et les trace dans le Packet Logger Mode
logto – trace les paquets dans un fichier spécifique
dsize – compare la taille du paquet avec une valeur différente v
flags – vérifie les drapeaux TCP pour des valeurs spécifiques
content – recherche un motif/une chaîne spécifique dans un paquet
content–list – recherche un motif/une chaîne spécifique dans un paquet
nocase – la casse (majuscule et minuscule) est négligée dans la chaîne recherchée
react – réponse active (bloque les sites web)
sid – ID de règle Snort
classtype – trie les attaques potentielles dans des groupes
priority – défini la sensibilité
Jusqu'ici, tout va bien mais comment fonctionne les règles individuelles ?
msg:
Nous trouvons souvent « msg » lorsque nous parcourons les règles. Cette option est responsable de la génération des alertes et leur traçage.

```
msg:"<texte>";
```

Le "<texte>" est le message qui est écrit/affiché dans le fichier d'alerte (alertfile)
logto:
Chaque paquet pour lequel la règle est d'application est tracé dans un fichier spécifique

```
logto: "<nomdefichier>";
```

Dans ce cas, "<nomdefichier>" est le fichier dans lequel les fichiers applicables sont tracés.
dsize:
Ceci est utilisé pour spécifier la taille d'un paquet. Si nous connaissons la taille du tampon d'un service spécifique, cette option peut être utilisée pour se défendre contre les débordement de tampon possible. Comparé à « content », elle est un peu plus rapide ; c'est pourquoi elle est utilisée plus fréquemment pour tester les débordements de tampon.

```
 dsize: [>|<] <taille>;
```

Les deux opérateur optionnels > et < indiquent que la taille du paquet devrait être respectivement plus grande et plus petite que la valeur spécifiée

```
    value flags:
```

Cela vérifie quels sont les drapeaux définis. À présent, 9 drapeaux sont disponibles dans Snort :

```
F       FIN
S       SYN
R       RST
P       PSH
A       ACK
U       URG
2       Bit 2 assigné
1       Bit 1 assigné
0       pas de drapeau TCP défini
```

Il existe des opérateurs logiques supplémentaires pour spécifer les critères de test des drapeaux :

```
+ ALL flag     = Marche pour tous les drapeaux (flag) spécifiés (aussi bien
que les autres).
* ANY flag     = Marche pour tous les drapeaux spécifiés.
! NOT flag     = Si les drapeaux spécifiés ne sont pas définis.
```

En général, le mot clé « flags » est utilisé comme ceci :

```
        flags: <valeur_drapeau>;
```

Les bits réservés peuvent être utilisés pour détecter un comportement inhabituel ; par exemple, des essais d'IP stack fingerprinting.
content:
Un des mots clés les plus utilisés (excepté « msg ») est « content ». Il peut être utilisé pour chercher dans la masse de paquets pour un contenu particulier. Si le contenu spécifié est détecté, des étapes prédéfinies sont lancées contre l'utilisateur. Après la détection du contenu dans la charge d'un paquet, le reste des règles de Snort va être exécuté. Sans appliquer « nocase » (voir ci−dessous), la casse sera considérée. Le contenu de la charge devrait être fouillée pour des binaires comme pour du texte. Les données binaires sont placées entre || et montées comme du byte code. Le byte code montre l'information binaire sous forme de nombres hexadécimaux. En rapport avec ce mot clé, l'opérateur de négation (!) peut être appliqué, par exemple, nous pouvons émettre une alerte si un paquet contient un texte spécifique.

```
        content: [!] "<contenu>";
```

La déclaration de ! n'est pas obligatoire.

```
 alert tcp any any -> 192.168.1.0/24 143 \
(content: "|90C8 C0FF FFFF|/bin/sh";\
msg: "IMAP buffer overflow";)
```

Comme démontré dans cette règle, les données binaires sont entourées dans || ; à la suite de quoi, du texte normal peut être appliqué. Regardez la description de « offset » et « depth » dans le Manuel des Utilisateurs de Snort : ils sont fréquemment utilisé dans ce contexte avec « context ».

content−list:

Ce mot clé fonctionne de manière similaire à « content » avec la différence qu'un nombre de chaînes (utilisées pour chercher dans les paquets) peut être entré. Nous entrons les nombres hexadécimaux, les chaînes, etc. dans un fichier. Ce fichier, contenant les chaînes à trouver, sera spécifié dans l'application de « content list ». Nous ne devons pas oublier d'écrire les lignes espacées verticalement (chaque chaîne sur une ligne). Par exemple :

```
"kinderporno"
"warez"
.....
```

A la suite de quoi, nous pouvons chercher pour le contenu de ce fichier (par application de "content−list: [!] "<filename>" "). Bien sûr, ! est optionnel ; il a le même effet que dans « content ».

nocase:
Cette règle joue un rôle important dans un contexte avec les mots clés « content ». D'habitude, la casse est respectée ; en utilisant « nocase », elle est ignorée :

```
alert tcp any any -> 192.168.1.0/24 21 (content: "USER root";\
 nocase; msg: "FTP root user access attempt";)
```

Sans l'utilisation de « nocase », seule la chaîne « USER root » va être répertoriée ; ici, puisque « nocase » a été spécifié, la casse ne sera pas appliquée.
Si la recherche d'un contenu spécifique (application de « content » ou « content_list ») dans un paquet donne un résultat, « react » peut être utilisé pour y répondre. Comme réponse générale, les pages spécifiques demandées par l'utilisateur seront bloquées (pages pornographiques...). En appliquant « Flex Resp », les connections peuvent être coupées ou des avertissements envoyés au navigateur. Les options suivantes sont possibles/légales :

block − déconnecte et envoie une notification.
warn − envoie un avertissement visible (bientôt disponible)
Ces « arguments de base » peuvent être complétés par des arguments additionnels (aussi appelés « modificateurs additionels ») :

msg − le texte à envoyer (par le mot clé « msg ») est inclus dans la notification allant être envoyée à l'utilisateur
proxy : <nnumerodeport> − le proxy est utilisé pour envoyer la notification (bientôt disponible)

```
react : <argument de base [, modificateur additionel ]>;
```

Le mot clé « react » est ajouté à la fin des options d'une règle. Il peut être utilisé comme ceci :

```
alert tcp any any <> 192.168.1.0/24 80 (content-list: "adults"; \
  msg: "Cette page n'est pas pour les enfants !"; react: block, msg;)
```

sid:
'sid' ou règles d'IDentification de Snort (Snort rules IDentification) est utilisé pour identifier des règles « spéciales » de Snort. Cela rend légal/possible l'utilisation de « plugins de sortie » pour identifier chaque règle. Il y a de nombreux groupes sid :

```
< 100 = réservé pour un usage futur
100-1 000 000 = règles qui viennent avec Snort
> 1 000 000 = utilisé pour les règles locales
```

sid−msg.map contient une carte des tags msg pour les sid. C'est utilisé par le post−traitement d'étranger pour assigner un avertissement à une identité
sid: <snort rules id>;

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80  \
 (msg: "WEB-IIS file permission canonicalization"; uricontent:\
 "/scripts"..%c1%9c../"; flags: A+;nocase;sid: 983;rev:1;)
```

classtype:
En utilisant « classtype », nous pouvons trier les attaques dans divers groupes. Dans les règles, nous pouvons déterminer la priorité d'une attaque potentielle et dans quel groupe elle appartient. Les règles qui sont catégorisées dans le fichier de configuration seront automatiquement attribuée à une priorité standard.

```
classtype: <class name>;
```

Les catégories de règles doivent être définies dans classification.config. La syntaxe suivante est utilisée :

```
config classification: <class name>,<class description>,\
      <standard priority>
```

Dans le prochain paragraphe (la description de « priority »), nous apprendrons quel type de groupes d'attaque existent.
priority:
Ce mot clé est utilisé pour assigner des « priorités de sécurité » à nos règles, signifiant combien endommageante pourrait être une attaque potentielle. Plus la priorité est élevée, plus le risque de sécurité potentiel est dommageable. En rapport avec les « class types » expliqué plus tôt, les priorités sont plus faciles à comprendre :

Class type	Description	Priorité
not-suspicious	Tout traffic "non suspect"	0
unknown	Traffic inconnu	1
bad-unknown	Traffic potentiellement "mauvais"	2
attempted-recon	Essai de fuite d'information	3
successful-recon-limited	Fuite d'information	4
successful-recon-largescale	Fuite d'information à grande échelle	5
attempted-dos	Essai d'attaque de déni de service	6
successful-dos	Attaque de déni de service réussie	7
attempted-user	Essai d'obtention des privilèges utilisateur	8
unsuccessful-user	Essai d'obtention des privilèges utilisateur non réussi	
successful-user	Essai d'obtention des privilèges utilisateur réussi	
attempted-admin	Essai d'obtention des privilèges administrateur	10
successful-admin	Essai d'obtention des privilèges administrateur réussi	

Comme mentionné précédemment, les priorité plus élevées représentent de plus grands risques de sécurité. Un

utilisateur reçevant des privilèges d'administrateur serait l'attaque la plus sérieuse.

```
alert tcp any any -> any 80 (msg: "WEB-MISC phf Versuch";\
       flags: A+;content: "/cgi-bin/bash";priority:10;)
```

Le sujet des « règles » est certainement assez complexe mais pas si difficile. Etudiez quelques règles, cherchez dans le manuel ce qu'elles sont et, après quelques temps, le « monstre des règles » prendra sens ;–) Des ressources pour Snort et la documentation peuvent être trouvés sur http://www.snort.org. Vous y trouverez des PDF de valeur (par exemple, le Manuel des Utilisateurs de Snort) qui ont été la source principale pour cette description.

LIDS

Depuis le papier (et les sources) de Stealth et le LIDS–Hacking–HOWTO, nous savons que LIDS n'augmente pas réellement la sécurité de l'ordinateur mais il tient plus du « root kit » dans certains cas ;–)

Penchons–nous sur le concept et, ensuite, à la force (imaginaire) de LIDS et ses faiblesses. Il a été développé, à l'origine, pour protéger, par exemple, des systèmes de fichier importants et pour cacher des processus spécifiques à l'utilisateur. De plus, il n'est pas autorisé à simplement lier des modules : les modules nécessaires seront liés avec le démarrage du système. Précédemment, j'ai mentionné le LIDS–Hacking–HOWTO et les papiers de Stealth, tous deux expliquant le travail et les faiblesses de LIDS. Je vais me limiter aux fonctionnalités les plus importantes. Vous trouverez un lien vers ces deux textes à la fin de cette section.

Le travail principal de LIDS est la protection du système de fichier. Pour être capable de les protéger, les fichiers/répertoires importants sont triés en groupes :

- Read Only = Ce fichier/répertoire n'a qu'un accès en lecture, les changements ne sont pas permis
- Append Only = Il est seulement permis « d'attacher » du contenu au fichier
- Exception = Ces fichiers ne sont pas protégés
- Protect (un)mounting = permet ou interdit de (dé)monter un système de fichiers

Pour une protection « réelle », quelques appels système sont manipulés pour être sûr que les protections sont maintenues (par exemple : sys_open(), sys_mknod(), ...).
De plus, LIDS empêche des processus spécifiques d'être tués (ou de devenir invisibles). Le but de cette procédure est d'empêcher l'attaquant de voir des processus spécifiques qui le tracerait. Un appel de « ps –ax » ne devrait pas non plus révéler nos processus. Pour véritablement cacher le processus, il est marqué comme « PF_HIDDEN ». Dans le cas où ps travaille pour générer l'information sur les processus, il va être empêché de le faire sur les processus qui sont marqués « PF_HIDDEN ». Cela, en soi, n'est pas suffisant pour cacher de manière sûre un processus parce qu'il a toujours temporairement une entrée dans le système de fichier proc (/proc). En conséquence, LIDS manipule également cette fonction pour empêcher le processus de se montrer dans le répertoire /proc. A part cela, il y a la possibilité de restreindre les privilèges d'un processus avec capacités. Si, par exemple, CAP_ROOT est défini à 0, le processus est empêché d'utiliser chroot (voir /usr/src/linux/include/linux/capatibilites.h).
De plus, LIDS a la possibilité de faire tourner une des deux options de sécurité : « security » ou « none_security ». Pour définir la différence entre « security » et « none_security », la variable globale « lids_load » est utilisée. Sa valeur par défaut est « 1 », ce qui signifie que LIDS tourne en mode « sécurité » (c'est–à–dire que les restrictions sont d'application). Si « security=0 » est défini au démarrage (invite LILO), « none_security » est activé. Il en résulte que toutes les vérifications de sécurité, toutes les restrictions... sont

désactivées. Avec « lids_load=0 », l'ordinateur fonctionne comme si LIDS n'était pas installé. Une autre possibilité de changer les options de sécurité est l'application en ligne de « lidsadm −S » (pour cela, un mot de passe doit être fourni).

LIDS offre également la possibilité de protéger les règles de pare−feu en activant CONFIG_LIDS_ALLOW_CHANGE_ROUTES et désactivant CAP_NET_ADMIN. Si quelqu'un veut modifier les règles du pare−feu, CAP_NET_ADMIN doit être activé pour empêcher n'importe de changer les règles. En plus de cela, Sniffer peut être désactivé et un scanner de port peut être intégré dans le noyau.

LIDS offre aussi plusieurs « options de réponse » (voyez la section sur la réponse) par exemple, pour envoyer des notifications avec un pager, par SMS (texto), à l'administrateur.

En général, il y a de nombreuses options ; Stealth explique, dans son papier, comment abuser LIDS : http://www.securitybugware.org/Linux/4997.html. Le LIDS Howto peut être trouvé sur : http://www.lids.org/lids−howto/lids−hacking−howto.html.

COLOID

COLOID est l'acronyme de « Collection of LKMs for Intrusion Detection » (collection de LKMs pour la détection d'intrusion) et il a été créé par moi−même, il y a quelques temps.
Depuis qu'on a appris que des parties du projet ne fonctionnaient pas bien, le projet a été temporairement arrêté. Cependant, je voudrais m'étendre un peu sur les fonctionnalités prévues initialement : avec le premier module (prev_exec), nous voulions − entre autres − empêcher temporairement l'exécution de binaires spécifiques à des moments définis (dans mes sources, j'utilisait le compilateur GNU GCC). Il est défini dans les sources quand une exécution doit être empêchée (le temps peut être spécifié en minutes). Si un utilisateur veut exécuter GCC à ce moment, il sera bloqué : GCC ne va pas s'exécuter. Si un utilisateur veut exécuter GCC à un moment « autorisé », les arguments vont être vérifiés et une recherche pour les fichiers .c prend place. La raison de cette procédure est la recherche, dans les sources (ici, quelqu'un veut compiler), des « fonctions dangereuses ». Par défaut, il y avait « scanf » et « strcpy », etc. ; il est possible d'ajouter plus de fonctions. Comme je l'ai indiqué plus tôt, le LKM cherche dans le code source ; s'il détecte une des fonctions, l'exécution de GCC est interdite. De plus, un fichier de log est écrit et un « beep » est produit.

Jusqu'ici, le module fonctionnait mais le concept n'était pas assez général et pouvait facilement être contourné.

Le second module était 'anom_detection' qui utilisait la Détection d'Anomalie mentionnée auparavant. En fait, deux LKM appartenaient à cette section :
1) Anomaly_Detection.c qui génère la base de données des activités normales de l'utilisateur
et
2) Misuse_Detect.c qui vérifie si le comportement d'un utilisateur s'écarte de la normale (décrite dans la base de données) en utilisant cette base de données comme repère.

Le plan était de laisser le LKM vérifier les critères suivants :

A quelle fréquence l'utilisateur exécute−t−il les commandes suivantes :

- su
- login

- chmod
- chown
- insmod
- ps
- lsmod
- rm
- last
- lastlog
- ftp

Quand l'utilisateur a−t−il exécuté les commandes suivantes :

- login
- su

Ou quand l'utilisateur a démarré normalement le PC et quand a−t−il lancé un shutdown.
Autre :
Combien de fois a−t−il ouvert (essayé d'ouvrir) les fichiers suivants :

- /etc/passwd
- /etc/group
- /etc/shadow
- /etc/ftpusers
- /etc/ftpgroups
- /etc/ftpaccess
- /etc/hosts.allow
- /etc/hosts.deny
- /etc/inetd.conf
- ...

Combien de fois exécute−t−il des programmes avec le bit SUID défini ?
Nous devons faire attention aux fichiers à surveiller (combien de fois l'utilisateur l'ouvre−t−il). Si nous choisissons trop de fichiers, la performance du PC en prendra un coup et un travail raisonnable et décent ne sera plus possible.

Il existait d'autres LKM plus petits sans toutes les sources ; les sources des deux modules mentionnés ci−dessus sont disponibles sur ma page web. Peut−être que quelqu'un fera quelque chose de cela, un jour, ...

Le mot de la fin

Si quelqu'un a plus d'idées pour le contenu de cet article, s'il vous plaît, envoyez−moi une note à Socma(Q)gmx.net. Pour les stimulations, louanges, critiques, ... supplémentaire, envoyez−moi également un e−mail
Références (outre celles mentionnées dans le texte) :

1. http://online.securityfocus.com/infocus/1524
2. http://online.securityfocus.com/infocus/1534
3. http://online.securityfocus.com/infocus/1544
4. http://online.securityfocus.com/infocus/1232
5. http://www.entercept.com/products/entercept/ whitepapers/downloads/systemcall.pdf
6. http://www.computec.ch/dokumente/intrusion_detection/ angriffsmoeglichkeiten_auf_okenas_stormwatch/ angriffsmoeglichkeiten_auf_okenas_stormwatch.doc

---

Site Web maintenu par l´équipe d´édition LinuxFocus © Klaus Müller "some rights reserved" see linuxfocus.org/license/ http://www.LinuxFocus.org	Translation information: de −−> −− : Klaus Müller <Socma(at)gmx.net> de −−> en: Jürgen Pohl <sept.sapins(Q)verizon.net> en −−> fr: Jean−Etienne Poirrier (homepage)

2005−02−07, generated by lfparser_pdf version 2.51