

Détection des intrusions et supervision de sécurité

M2-SSI

Guillaume HIET
guillaume.hiet@supelec.fr

Equipe CIDRE, SUPELEC

Décembre 2014

1 Supervision de sécurité

2 Sonde NIDS

Snort

Bro

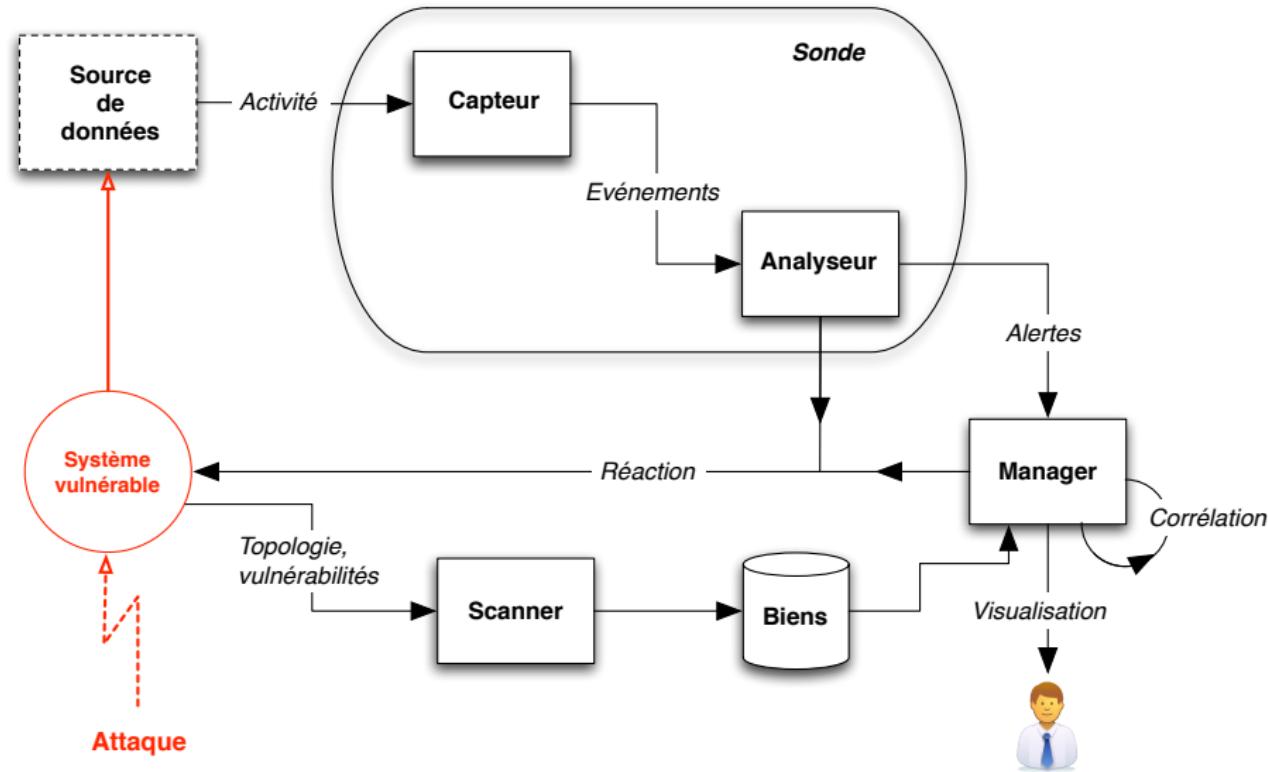
3 Architecture de supervision

4 Outils de détection d'intrusions

Sondes NIDS

Sondes HIDS

Supervision de sécurité : architecture fonctionnelle



1 Supervision de sécurité

2 Sonde NIDS

Snort

Bro

3 Architecture de supervision

4 Outils de détection d'intrusions

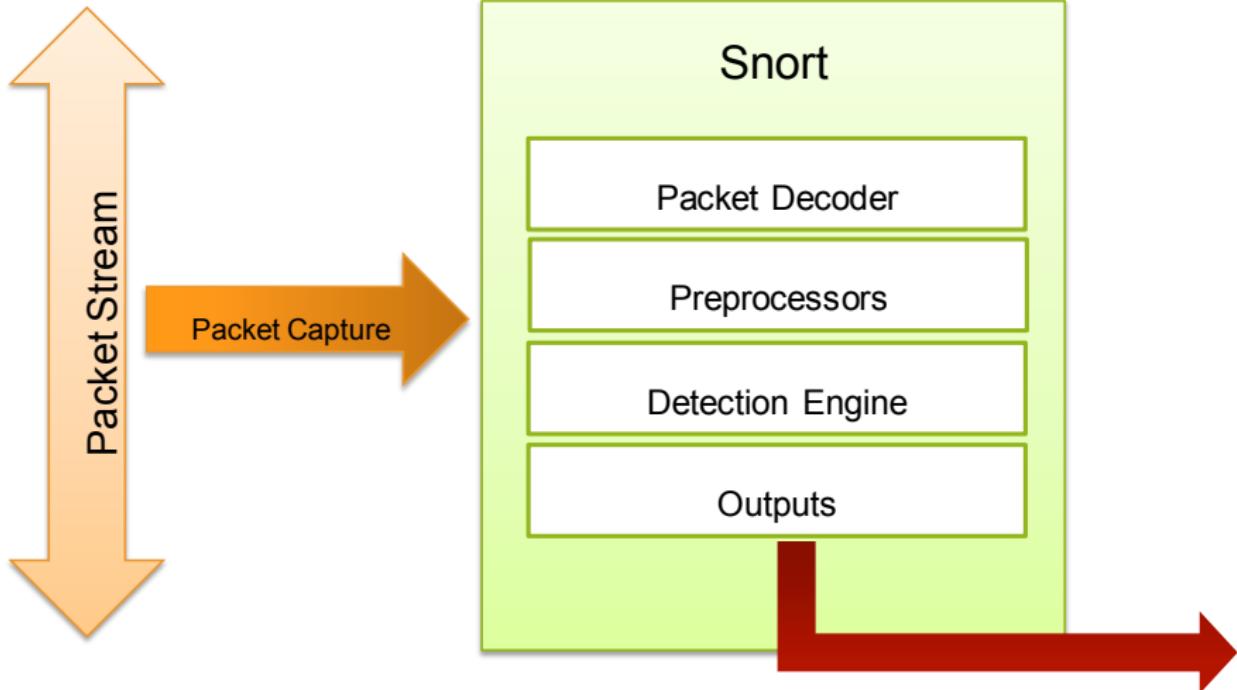
Sondes NIDS

Sondes HIDS



- Sonde NIDS/NIPS libre créée par Marty Roesch
- Développée par Sourcefire (rachetée par Cisco en 2013)
- Première version en 98 (pour UNIX)
 - *"Lightweight" intrusion detection*
 - Simple *pattern matching* sur les paquets
 - Pas de défragmentation, pas de reconstruction de flux
- Aujourd'hui, logiciel complet utilisé dans des *appliances* (dont celles de Sourcefire)
 - Frag3, Stream5 → défragmentation, suivi flux TCP/UDP
 - Décodage applicatif (HTTP, FTP, etc.)
 - Différents modules de capture (inline, passif) et de sortie (fichier, BDD, etc.)
- URL : <http://www.snort.org/>

Architecture de Snort V2

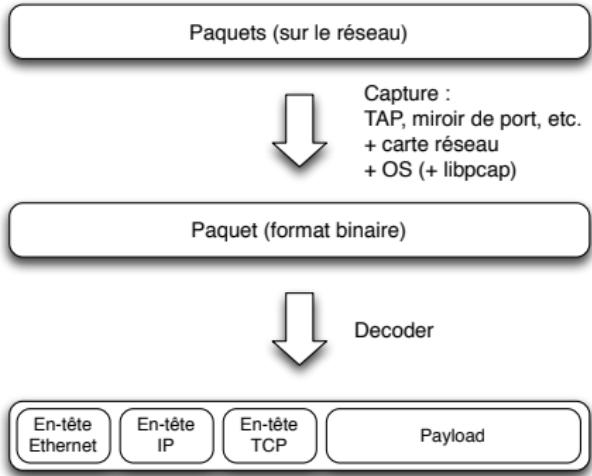


Modules de capture

- Gérés par LibDAQ (V2.9)

Decoder

- Analyse protocolaire sommaire
- Décodage entêtes Ethernet, IP et TCP
- Quelques vérification simples (taille champs en-têtes, etc.)



Préprocesseurs

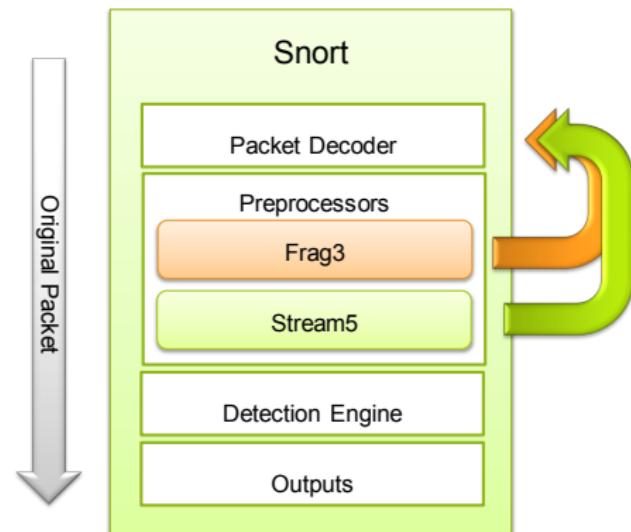
- Défragmentation, reconstruction de flux
- Analyse des protocoles applicatifs
- Détection d'anomalies, normalisation, etc.
- Attention, ils sont exécutés dans l'ordre où il sont déclarés dans le fichier de configuration !



- Moteur de détection + règles (signatures)
 - Moteur de reconnaissances de signatures
 - Les signatures (dont dépendent beaucoup les résultats)
- Modules dynamiques
 - Pré-processeurs dynamiques
 - Règles dynamiques (*so-rules*)
- Modules de sortie :
 - Syslog
 - Ecriture directe en BDD
 - Unified1 et 2 (format binaire, rapide, permet de faire tampon avant traitement en BDD)
 - Export Prelude
 - Réaction (IPS), etc.

Les préprocesseurs classiques : Frag3 et Stream5

- Frag3 :
 - Défragmentation IP
 - Détection d'attaques liées à la fragmentation
 - Génération d'un pseudo paquet qui est réinjecté (les paquets fragmentés suivent leur chemin)
- Stream5 :
 - Réassemblage flux TCP, suivi états
 - Détection attaques liées à la « fragmentation » TCP
 - Génération d'un pseudo paquet
 - Configuration par port



Les préprocesseurs classiques : http_inspect

- Http_inspect :
 - Normalisation URI
 - Création tampon URI (qui peut être inspecté dans les règles → uricontent)
 - Détection des tentatives d'évasion et des anomalies HTTP
 - Attention au paramétrage de flow_depth : faux négatifs vs. consommation de ressources importantes

```
GET /downloads/..../cgi-bin/..../pics\..../downloads/.snort.tar.gz HTTP/1.0
```



```
/downloads/snort.tar.gz
```

- Signatures « officielles » VRT (Sourcefire)
 - Non-libres
 - Gratuites pour une utilisation non-commerciale (disponibles 1 mois après parution)
 - Abonnement payant : utilisation commerciale et disponibilité immédiate
- Signatures fournies par des tiers : par exemple Emerging Threats¹
- Règle (signature) = en-tête + options
- Options :
 - **general** : msg, reference, sid, classtype, priority, etc.
 - **payload** : content, uricontent, isdataat, pcre, byte_x, etc.
 - **non-payload** : flow, ttl, tos, id, fragbites, dsizes, flags, flowbits, etc.
 - **post-detection** : resp, react, tag, activate, activate_by, count, replace, etc.

1. <http://www.emergingthreats.net/>

Exemple de règles (1/2)

```
alert tcp $EXTERNAL_NET any -> $HOME_NET
 4000 (msg:"EXPLOIT Alt-N SecurityGateway
  username buffer overflow attempt"; flow:established,
  to_server; content:"username>"; nocase;
  isdataat:450,relative; content:!"&"; within:450;
  content:!"\0A"; within:450; metadata:policy balanced-
  ips drop, policy connectivity-ips drop, policy security-
  ips drop; reference:url,secunia.com/advisories/30497/;
  classtype:attempted-admin; sid:13916; rev:2;)
```

Exemple de règles (2/2)

Exemple de vulnérabilité : *SQL injection*

code PHP :

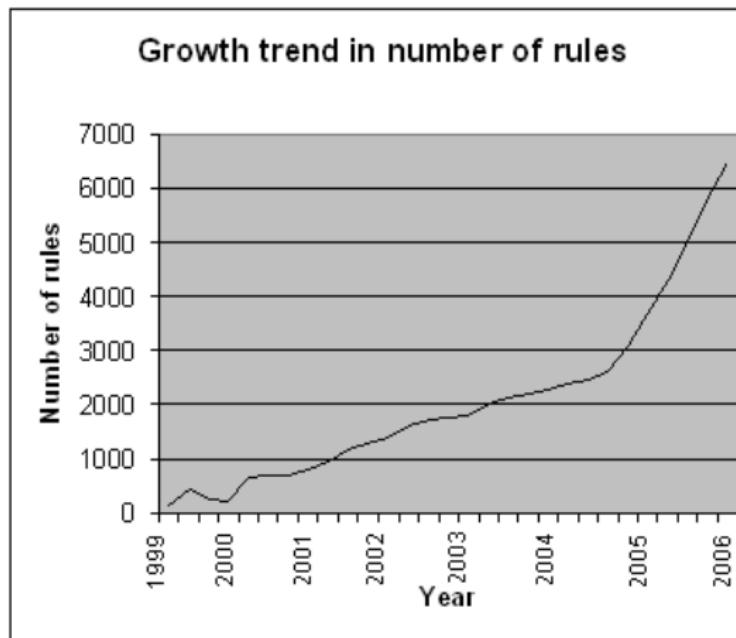
```
req = "SELECT * FROM users WHERE name = '" + user + "';"
```

Une signature très (trop) générique

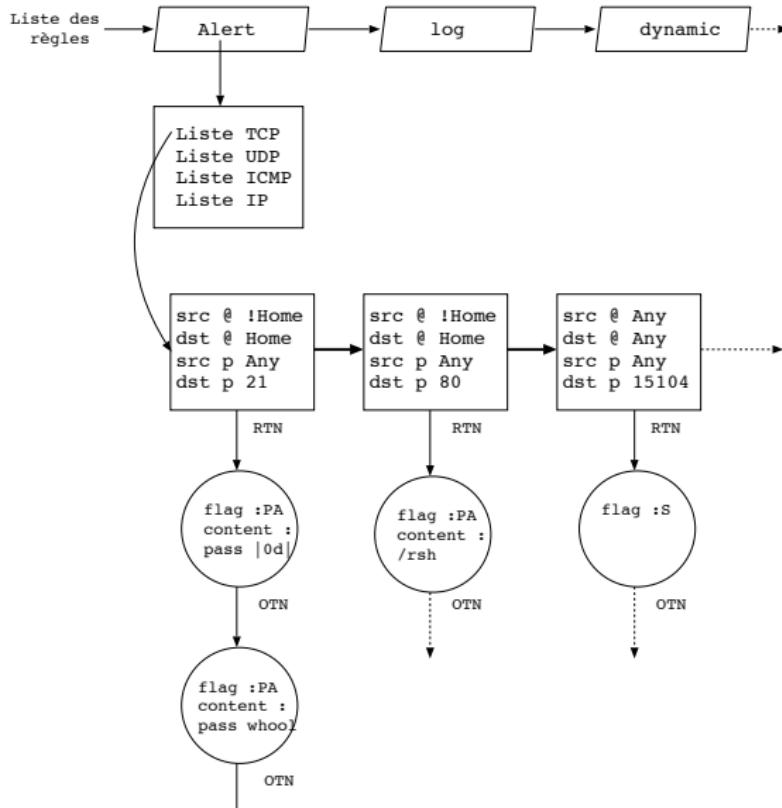
```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"SQL Injection"; flow:to_server,established;
uricontent:".php"; pcre:"/(%27)|(')|(-\-)|(%23)|(#)/i";
classtype:Web-application-attack; sid:9099; rev:5;)
```

Problématique du nombre de règles

- Problème : nombre de règles ↗ \Rightarrow nombre motifs ↗
 \Rightarrow Optimiser la gestion des règles pour limiter la vérification des motifs



Organisation des règles en mémoire



Pour chaque (pseudo) paquet :

- Identifier les RTN suivant en-tête TPC/IP du paquet
- Vérifier chaque OTN :
 - motifs fixes (ex : `content`) : Boyer-Moore (recherche séquentielle) ou Aho-Corasick (recherche en parallèle)
 - `regexp (pcre)` : moteur NFA/DFA
 - algorithmes *ad hoc* pour les autres types de motifs (non-payload, `isdataat`, `byte_x`, etc.)
- Les OTN sont évalués séquentiellement ou en parallèle suivant une stratégie complexe (compromis temps/mémoire)
- Les algorithmes doivent être robustes pour éviter les attaques en complexité algorithmique
- Identification des règles vérifiées en fonction des motifs vérifiés et réalisation de l'action correspondante (génération alerte, etc.)

Améliorer la qualité et la précision du décodage protocolaire

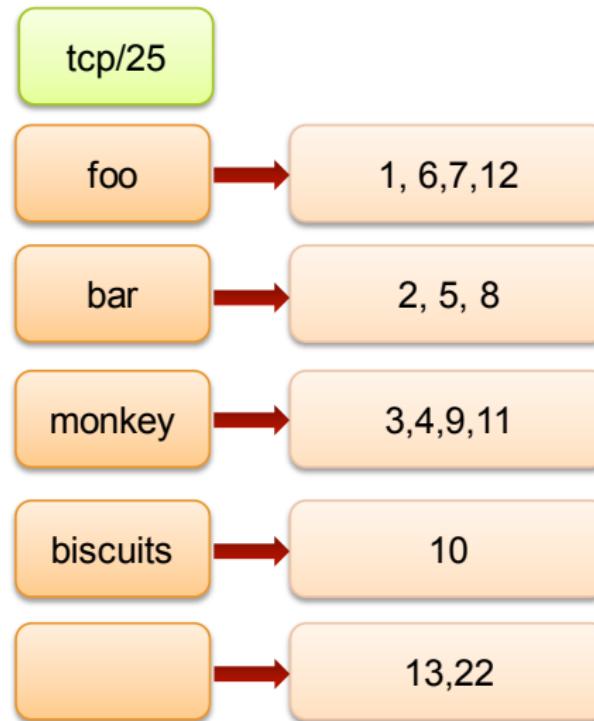
- Fournir des événements de « haut-niveau » (établissement de connexion, téléchargement d'un fichier, etc.)
- Permettre de restreindre la recherche sur des champs protocolaires précis

Améliorer les performances du moteur de détection

- Analyser les règles pour proposer des stratégies de regroupement (compromis temps/mémoire)
- Utilisation de matériel dédié (FPGA, ASIC, GPU)
- Parallélisation du processus de détection (cluster)

- Un automate AC pour chaque RTN → Consommation mémoire importante (beaucoup de règles par RTN)
- Depuis 2.8.2, *fast-pattern matcher* : chercher uniquement les motifs si la règles a « des chances » d'aboutir
- Grouper les règles partageant un même motif :
 - le premier motif commun le plus long
 - motif identifié par l'option `fast_pattern`
 - listes de règles (= liste d'options) associées à chaque motif discriminant
- Détection en trois étapes :
 - ① Filtrage suivant en-tête (@, ports)
 - ② Multi pattern matching (MPSE) sur les motifs discriminants (Aho-Corasick), empile les état de reconnaissance atteints
 - ③ Pour chaque motif discriminant reconnus, évaluation successives des options à l'aide du module de détection adéquat (Boyer-Moore pour les contenus de type chaînes de caractères), évaluation des PCRE à part (NFA/DFA)

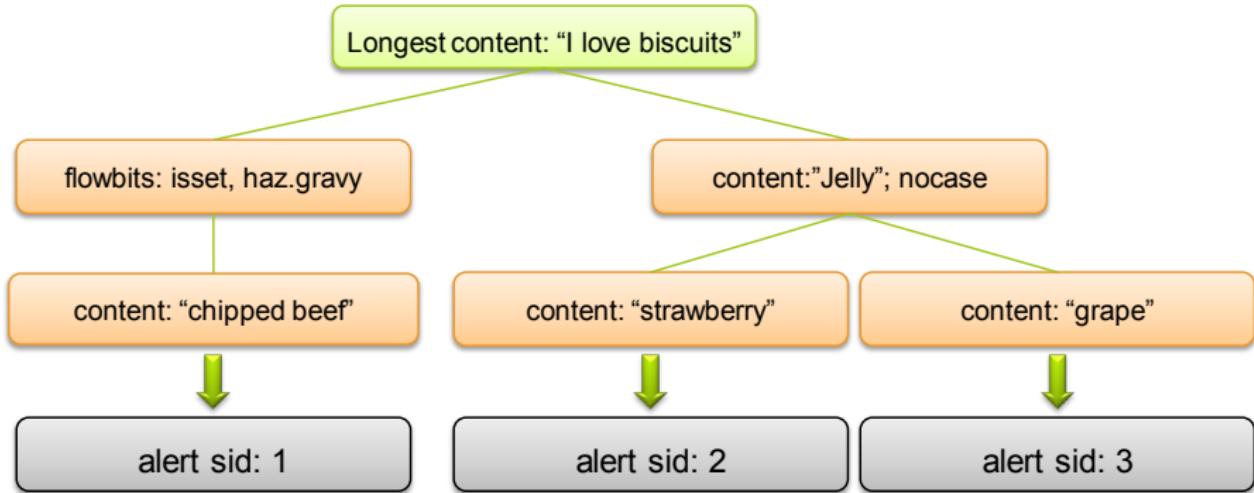
Fast pattern matcher



Arbre de motifs

- Ecriture automatisée de règle à l'aide de scripts → beaucoup de règles se ressemblent
- Idée : optimiser le traitement de règles similaires (qui partagent beaucoup d'options)
- Maintenir le PM dans le cache du processeur (évaluation de la liste de règles → chargement/déchargement) : mettre en cache les motifs détectés durant phase 1 (A-C)
- Généralisation et construction d'un arbre de motifs pour chaque motif le plus long
- Construction en fonction de l'ordre de spécification
 - Attention lors de l'écriture de règles, implication du concepteur de règles dans l'efficacité
 - Spécifier les motifs facilement discriminants en premier
- Lors de la détection, parcourt en profondeur d'abord
- Fait l'objet d'un brevet Sourcefire : US20090262659

Arbre d'options



(flowbits: isset, haz.gravy; content:"I love biscuits"; content:"chipped beef"; sid: 1)
 (content:"I love biscuits"; content:"Jelly"; nocase; content: "strawberry"; sid: 2)
 (content:"I love biscuits"; content:"Jelly"; nocase; content: "grape"; sid: 3)

- Ecrire de règles efficaces
 - Spécifier un motif de type `content` discriminant (le plus long possible)
 - Utiliser éventuellement l'option `fast_pattern` si un motif plus petit est discriminant
 - Placer en premier les options rapides à vérifier (`flow`, `flowbits`)
 - Placer en dernier les vérifications coûteuses (PCRE)
 - Attention aux ressources consommées
 - Attention à l'utilisation de PCRE (backtracking, attaques en complexité algorithmique, etc.)
- Vérifier le contexte (décodage protocolaire) à l'aide des options de test d'octet.
- Restreindre la recherche aux champs adéquats
- Eviter les règles spécifiques à un *exploit* → *vulnerability rules*

Les algorithmes utilisés : recherche mono motif

- Force brute : $O(L \cdot K)$, pas utilisée
- Boyer-Moore

- Vérification par la fin du motif (droite à gauche)
- En cas d'échec, déplacement de la fenêtre de comparaison (de gauche à droite)

- Bad character heuristics

0	1	2	3	4	5	6	7	8	9	...
a	b	b	a	b	a	b	a	c	b	a
b	a	b	a	c						
b	a	b	a	c						

- Good suffix heuristics

0	1	2	3	4	5	6	7	8	9	...
a	b	a	a	b	a	b	a	c	b	a
c	a	b	a	b						
c	a	b	a	b						

- Meilleur cas : $O(L/K)$
- Pire cas : $O(L \cdot K)$, $O(L + K)$ avec règle de Galil

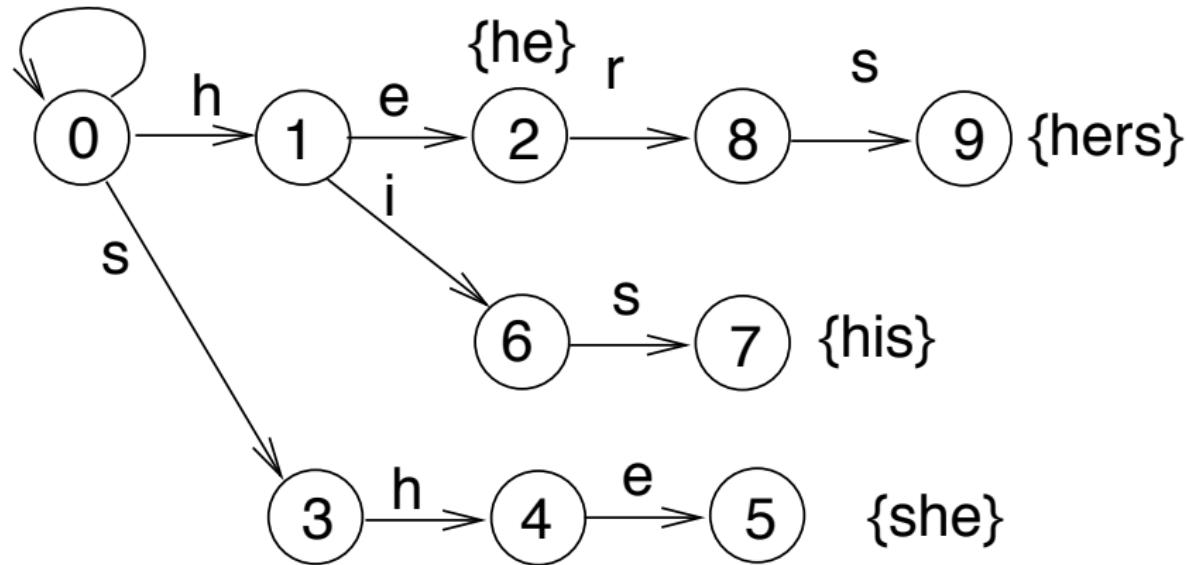
Les algorithmes utilisés : recherche multi-motif

- Aho-Corasick :
 - Construction d'un arbre de motifs
 - Utilisation en NFA
 - Amélioration : fonction d'échec (évite le *backtracking*)
 - Génération du DFA équivalent (transitions supplémentaires)
 - Complexité de la recherche : $O(n+m+z)$, z nombre d'occurrences du motif
- Problème : taille de l'automate (donc de la matrice de transition)
 - Occupation mémoire : $O(n.|A|)$, n nombre états, A alphabet
 - Problématique de la gestion du cache → dégradation des performances
 - Utilisation de structures compactes (sparse, band, etc.)
 - Occupation mémoire : DFA Full > NFA full > DFA banded > NFA banded
 - Induit une baisse des performances (problématique de l'accès aléatoire)
 - Choix par défaut de Snort : AC-Split (variante de AC-Full)

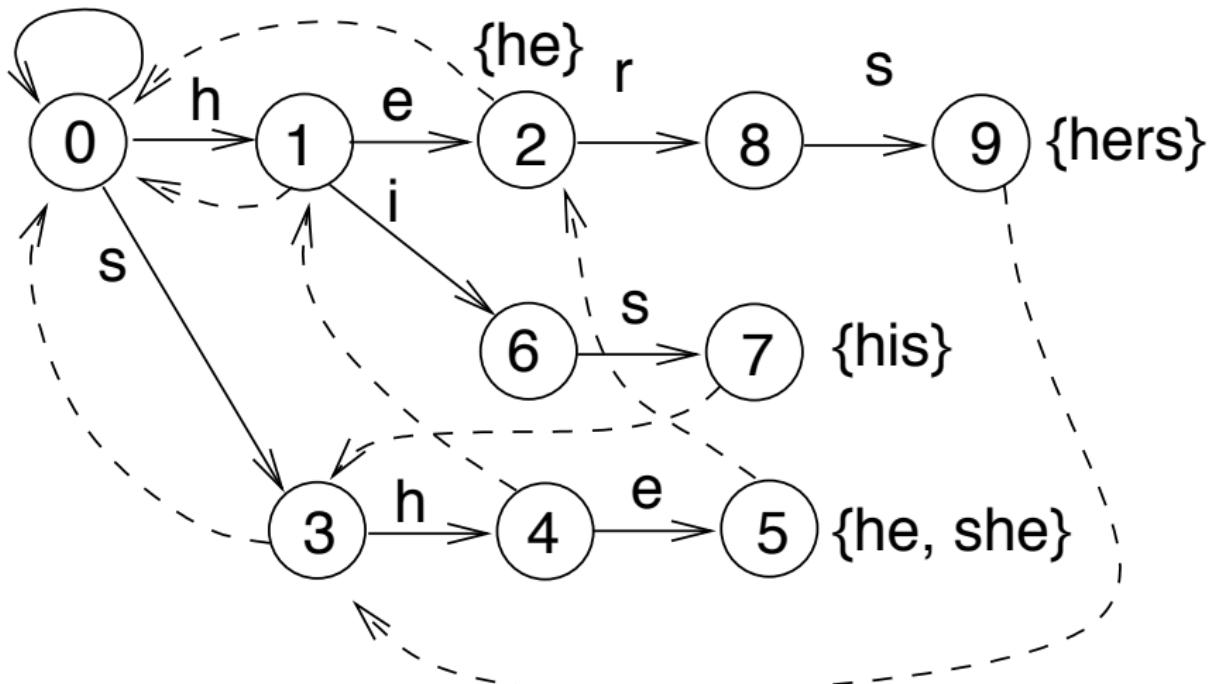
Alternatives

- Rabin-Karp, Wu-Mamber, etc.
- Peu robustes : cf attaques en complexité algorithmiques
 - Exemple : Wu-Mamber (par défaut dans les anciennes versions de Snort)
 - Repose sur des décalages (cf Boyer-Moore) → très sensible à la taille du motif
 - Recherche limitée par la taille du motif le plus petit (→ petits décalages)
 - Algorithme plus efficace sur un ensemble de motifs de grandes tailles
- Les expression régulières : NFA/DFA

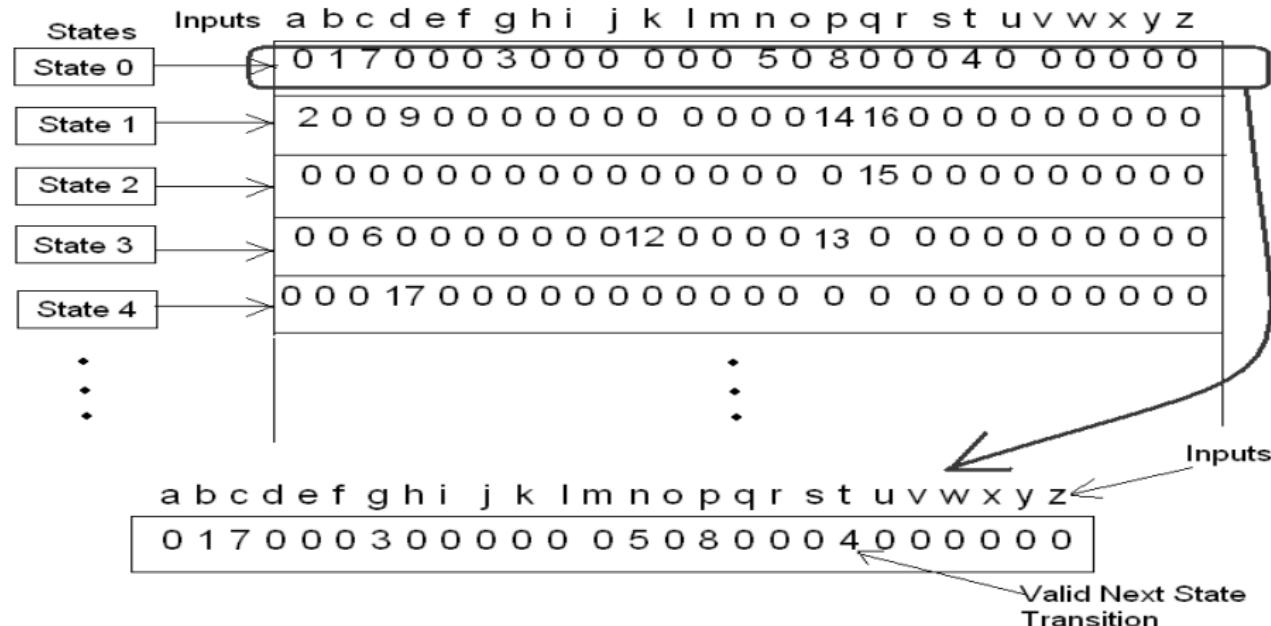
$/=\{h, s\}$



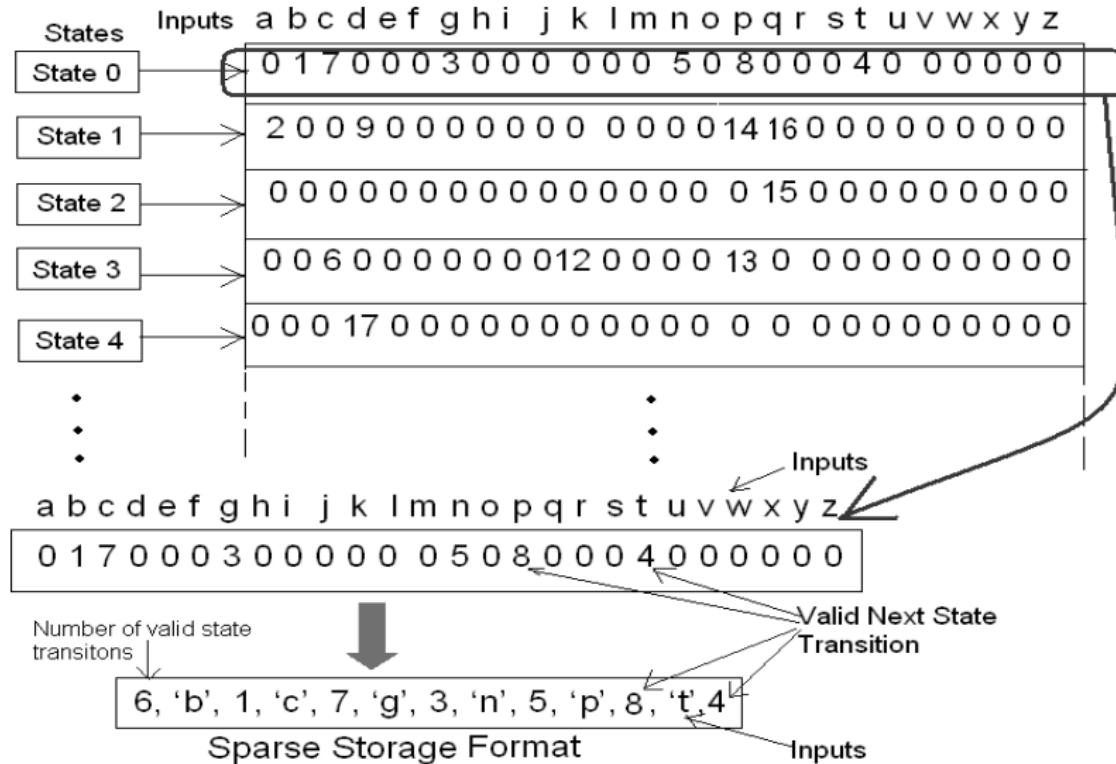
$/=\{h, s\}$



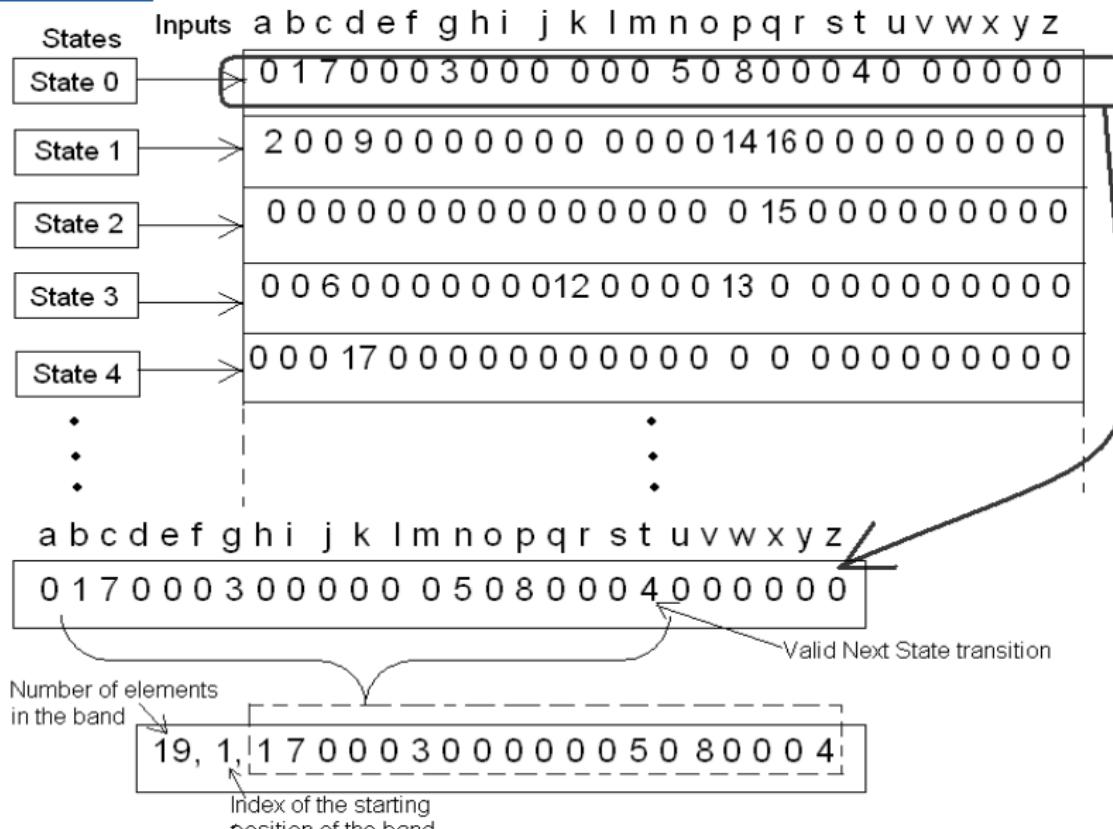
Aho-Corasick : Full-Matrix



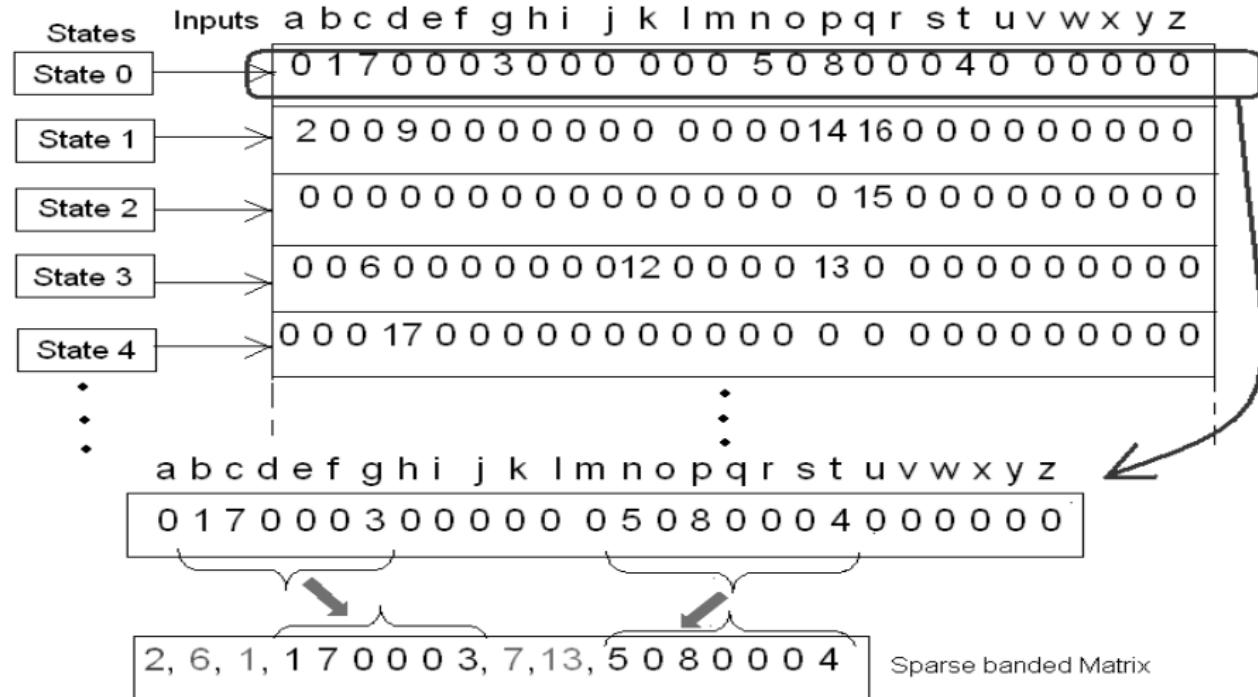
Aho-Corasick : Sparse



Aho-Corasick : Banded



Aho-Corasick : Sparse-Banded



- Plusieurs possibilité de détection : préprocesseur (existant ou ad-hoc), pattern fixe (content), PCRE, etc.
- Modules d'extension :
 - pré-processeurs dynamiques
 - règles dynamiques (so rules)
 - règles écrites en C
 - comprises dans les règles distribuées par Sourcefire
 - permettent des formes plus évoluées de détection
 - ... mais aussi d'obfuscuer les règles !
- Chargés dynamiquement par Snort au démarrage
- Configuration de Snort

- Option --dynamic-detection-lib
- Création de « méta règles » qui gèrent le filtrage sur les entêtes pour déclencher les règles dynamiques :

```
alert tcp any any -> any any (msg:"Mon Alerté"; ...;
metadata:engine shared, soid 3|2000001;
sid:2000001;gid:3;rev:1;...)
```

- Règles dynamiques
 - Snort plugin API
 - Inclusion de `sf_snort_plugin_api.h`
 - Structures permettant d'exprimer des règles « classiques »
 - Possibilité de détecter des motifs sur les contextes supportés par Snort
 - Tableau d'options de règles : permet de déclarer des règles
 - Règles Snort écrite en C : intérêt ? (obfuscation....)
 - Plus intéressant, possibilité d'installer un *callback* vers une fonction de détection *ad hoc*
 - en paramètre : le paquet (ou le méta paquet)
 - valeur de retour : RULE_MATCH ou RULE_NOMATCH
- Pré-processeur dynamiques
 - Inclusion de `sf_dynamic_preprocessor.h`
 - Possibilité de vérifier des signatures !
 - Et même de ne pas générer la forêts de règles...
- Détection d'anomalies, approches *ad hoc*, etc.
- Assez complexes et pas toujours bien documenté
- S'inspirer du code source des modules libres existants

- Pré-processeur de normalisation pour le mode inline
- Efforts pour améliorer le décodage applicatif :
 - Fournir au moteur de détection des champs utilisables dans les règles (contexte de détection) : cf Bro...
 - Facilite l'écriture de règles (moins de règles)
 - Exemple : DCE/RPC.
 - Complexifie le langage de règles
 - Attention aux performances si tous les décodeurs sont activés...
- Adaptive (target-based) profile.
 - Repose sur un fichier décrivant la configuration des serveurs.
 - Informations utilisées notamment par les préprocesseurs
- Razorback (détection attaque poste client/malware)
- Support Intel Quick Assist (support matériel pour la recherche de motif)
- Continuer la détection pendant le changement ou la mise à jour de la politique (sans arrêter l'IDS)
- LibDAQ : abstraction des différents modules d'acquisition

Snort : conclusion

- Une solution mature de NIDS/NIPS open-source
- Soutenue par Sourcefire qui fournit des services et des produits (notamment des signatures, équipe VRT)
- Une architecture pas toujours « élégante »
 - Pas de séparation claire décodage protocolaire / détection
 - Les pré-processeurs font à la fois du décodage et de la détection (attention à l'ordre)
 - ... les règles aussi ! → complexifie les règles et le travail des équipes de sécurité
 - + seul les décodages nécessaires sont réalisés (cf coût du décodage)
- Un moteur de détection complexe, utilisant différents modules de détection : efficacité globale ?
- Un seul fil d'exécution → sous-exploitation des processeurs « modernes »
 - Architecture à plusieurs fils d'exécution : cf Snort 3, Suricata
 - Plusieurs processus : utilisation en cluster
 - Problème de synchronisation et de corrélation des événements

1 Supervision de sécurité

2 Sonde NIDS

Snort

Bro

3 Architecture de supervision

4 Outils de détection d'intrusions

Sondes NIDS

Sondes HIDS

Exemple d'architecture : Bro

- Sonde IDS libre issue des travaux académiques de Vern Paxson (98)
- NIDS réseau destiné à la détection dans le cœur de réseau (Gbps)
- Développement assuré en grande partie par le International Computer Science Institute (ICSI) du LBNL
- Financement par le National Science Foundation ?s Strategic Technologies for the Internet program (3M\$!)
- Minimiser pertes paquets et les faux positifs (alertes peu pertinentes)
- Architecture modulaire facilitant l'ajout de fonctionnalité
 - nouveaux protocoles
 - nouvelle forme de détection
- Dernière version stable : v 1.5
- Compatibilité UNIX (licence BSD), utilisation recommandée sur FreeBSD
- Pas de paquets Debian (c'est en cours...)

- Plutôt destiné à un public d'expert (ou d'intégrateur)
- Module de réaction limité : logger ou exécuter un script
- Pas réellement un NIPS (pas de mode inline par défaut)
- Points forts :
 - décodage protocolaire (*application level semantics*)
 - liens entre les informations (forme de corrélation, détection multi-événementielle)
 - prise en compte du contexte (signatures robustes)
- Décodage protocolaire évolué :
 - Fournit des événements qui seront analysés par les règles de Bro
 - Exemple : "nouvelle connexion TCP, HTTP GET, etc.
 - Chargement dynamiques des analyseurs (DPD)
- Règles évoluées :
 - Spécifiées dans un langage *ad hoc* (Bro language)
 - Forme de détection d'anomalies (ou de policy-based)
 - Quelles activités réseaux semblent suspectes
 - Plus que de la simple reconnaissance de motifs
 - Un peu complexe, demande du temps pour maîtriser

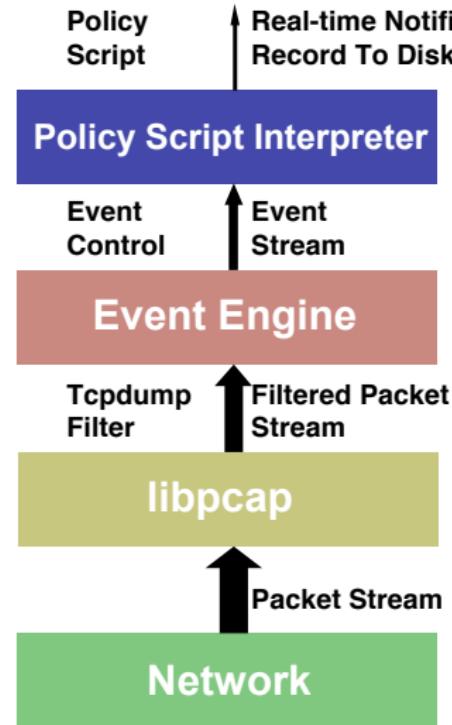
- Un mécanisme de détection de motifs « à la Snort »
- Utiliser les expressions rationnelles, pas de distinction regexp / fixed string
- Mécanisme additionnel, surtout utilisé pour le DPD (cf dpd.sig)
- Un certain niveau de compatibilité avec les signatures de Snort :
 - Utilisation d'un outil de conversion
 - Ne supporte pas les dernières versions de signatures Snort
 - Certaines options récentes ne sont pas reconnues
 - Mappings pas toujours possibles entre les deux langages
 - Complexification du langage de Snort, différence d'approche
- Un ensemble de signatures fourni
 - Obsolète (pas de mise à jour)
 - Surtout à titre d'exemples
- Génération d'événements (qui peuvent être pris en compte par des scripts de politique)

Exemple de signature

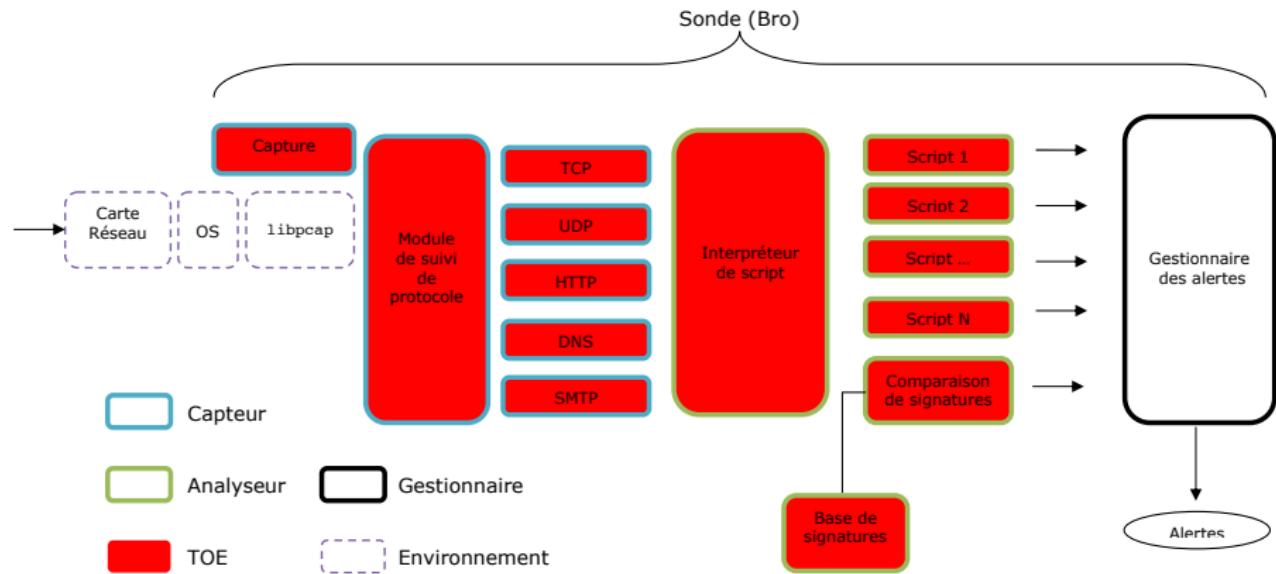
Détecter les accès au fichier test.php sur un serveur Web

```
signature sid-2152 {  
    ip-proto == tcp  
    src-ip != local_nets  
    dst-ip == http_servers  
    dst-port == http_ports  
    event "WEB-PHP test.php access"  
    http /.*[\\/\\]test\\.php/  
    tcp-state established,originator  
}
```

Architecture de Bro



Architecture de Bro (cf CSPN)



- Capture à l'aide de libpcap
- Event manager (décodage protocolaire et génération d'événements)
 - Contrôles intégrité (en tête IP), défragmentation IP
 - Détection d'anomalies protocolaire
 - Création de contexte de session (@, port destination et sources, protocoles)
 - cf décodeur et pré-processeurs Snort
 - Possibilité d'activer un ou plusieurs analyseurs
 - Attention : activer analyseur → impact sur les performances
- Module de décodage protocolaire (transport)
 - TCP, UDP
 - Contrôle intégrité, détection anomalies protocolaires
 - Suivi état, reconstruction de flux pour la détection de motif
 - cf Snort stream5
 - Génération d'événements :
 - TCP : connection_attempt, connection_established, connection_rejected, connection_finished
 - UDP : udp_request, udp_reply

- Evénements nom + liste de paramètres typés

```
new_connection(c: connection)
```

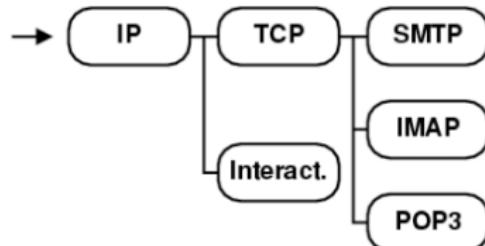
```
new_packet(c: connection, p: pkt_hdr)
```

```
tcp_contents(c: connection, is_orig: bool, seq: cou
```

- Architecture extensible

- Possibilité de définir ses propres analyseurs (en C++)
- Outil BinPAC permet d'automatiser la génération en fonction des spécification du protocole)
- Binary Protocol Analyzer Compiler, style déclaratif
- Possibilité de définir ses propres événements

- Arbre de décodeurs



Exemple de déclaration de protocole (DNS)

```
type DNS_message = record {  
header: DNS_header;  
question: DNS_question(this) [header.qdcount];  
answer: DNS_rr(this, DNS_ANSWER) [header.ancount];  
authority: DNS_rr(this, DNS_AUTHORITY) [header.nscount];  
additional: DNS_rr(this, DNS_ADDITIONAL) [header.arcoun]  
} &byteorder = big endian, &exportsourcedata;  
  
type DNS_rr(msg: DNS_message, ans: DNS_answer_type) =  
rr_name: DNS_name(msg);  
rr_type: uint16;  
rr_class: uint16;  
rr_ttl: uint32;  
rr_rdlength: uint16;  
rr_rdata: DNS_rdata(msg, rr_type, rr_class) &length =  
};
```

- Dernier module : analyse des événements (au sens IDS)
- Traitement asynchrone des événements qui sont mis en file (pour un paquet), appelé après chaque traitement de paquet
- Interprétation de scripts (règles) permettant la détection/réaction en fonction des événements capturés
- Scripts Bro : détection d'anomalies réseau, vérification de politiques (cf Snort so-rules)
- Spécification de gestionnaires d'événements (event handler)
- Langage Bro
 - procédurale, type spécialisé pour l'analyse réseau
 - fortement typé, typage implicite
- Variables globales : suivi à état
 - aussi utilisées pour le paramétrage
 - définies par l'utilisateur, différents types (tableaux, liste, etc.)
 - cf Snort, mais plus souple
- Gestion de timer
- Scripts Bro permettent également le paramétrage, la gestion, etc.

50 requêtes sur le port 135 en moins de 5 minutes

```
[...]
global w32b_scanned: table[addr] of set[addr] &write_expire = 5min;
global w32b_reported: set[addr] &persistent;

const W32B_port = 135/tcp;
const W32B_MIN_ATTEMPTS = 50 &redef;

redef enum Notice += W32B;

event connection_attempt(c: connection) {
    if (c$id$resp_p != W32B_port)
        return;

    local ip = c$id$orig_h;

    if (ip in w32b_reported)
        return;

    if (ip in w32b_scanned) {
        add (w32b_scanned[ip])[c$id$resp_h];

        if (length(w32b_scanned[ip]) >= W32B_MIN_ATTEMPTS) {
            NOTICE([$note=W32B, $conn=c,
                    $msg=fmt("W32.Blaster source: %s", ip)]);
            add w32b_reported[ip];
        }
    } else
        w32b_scanned[ip] = set(ip) &mergeable;
}
```

- Deux types de sortie : notification(notice) et alertes
- Par défaut, émission d'une information :

```
NOTICE ([ $note=OutboundTFTP, $conn=u,  
$msg=fmt ("outbound TFTP: %s -> %s", src, dst) ]);
```

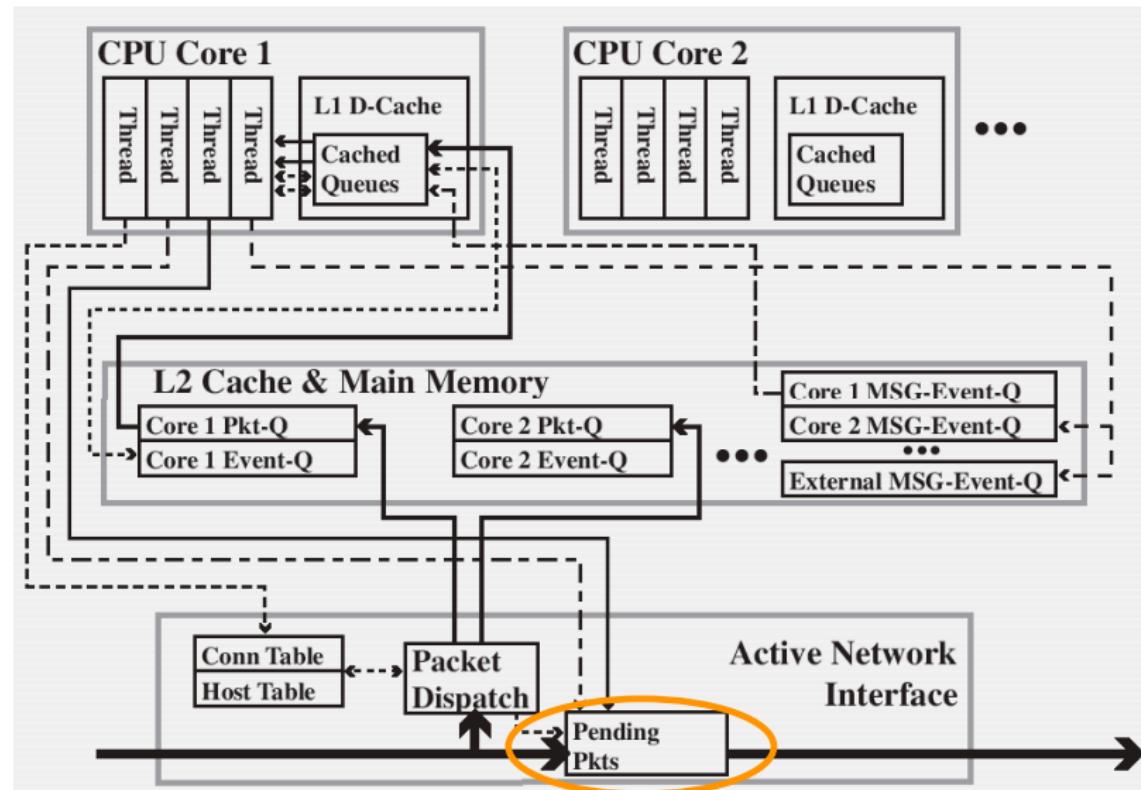
- Des scripts permettent de traiter les notifications et de définir la politique de gestion des notifications
 - règles prédéfinies : transformation en alertes (par défaut), log, etc.
 - possibilité de définir ses propres fonctions de traitement

```
redef notice_policy += {  
    [$pred(a: notice_info) =  
     {  
         # Do not report this notice for remote hosts.  
         return a$note == ProtocolDetector::ServerFound &&  
                ! is_local_addr(a$src);  
     },  
     $result = NOTICE_FILE,  
    ] };
```

- (analyse paquet, détection motif) → événements → (script politique) → notification → (script) → alertes
- Riche mais un peu complexe
- Permet une forme de corrélation
- Un type d'événement particulier : activité « bizarre » (*weird activity*)
 - Anomalie protocolaire
 - Fait l'objet d'un script dédié (qui peut être modifié)
 - Typage des événement permet un traitement approprié
 - Eviter les alertes peu pertinente

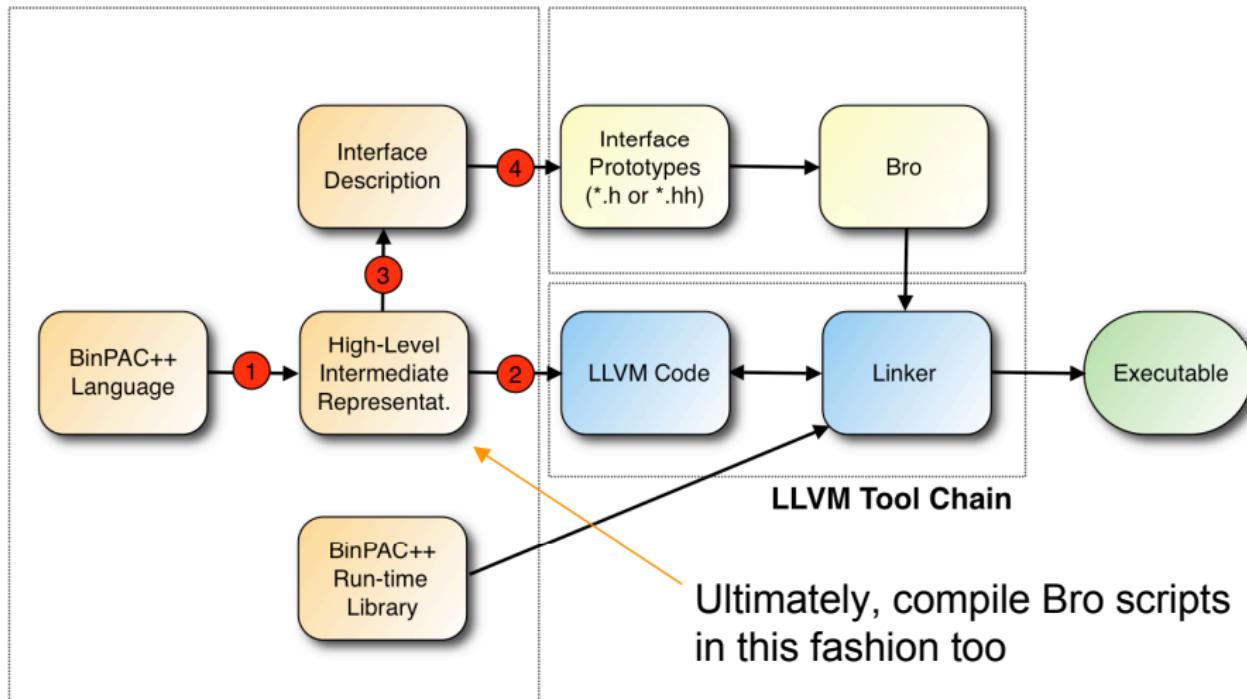
Travaux en cours

- Support matériel (FPGA, CUDA, etc.)
- Amélioration du fonctionnement en cluster, gestion des processeurs multi cœur
- Amélioration du décodage protocolaire (et de BinPAC)
- Interprétation → compilation



BinPAC++

Host Application



Conclusion sur Bro

- Permet surtout l'analyse du trafic en mémorisant les connexions, les messages échangées, etc.
- Peu de faux + (mais le spectre des attaques visé n'est pas le même)
- Adapté à la détection d'anomalies, de violation de politique, détection de comportement réseau
- Très modulaire : analyseurs, script, signatures modifiables
- Bibliothèque pour échanger avec Bro : Broccoli
- Quelques scripts sont fournis (détection DDOS, etc.) : peuvent servir d'exemple
- Demande un effort non négligeable pour maîtriser l'outil, écrire ses propres règles
- Documentation incomplète, pas toujours à jour
- Pas d'interface graphique, manque d'outils facilitant l'intégration
- Evalué CSPN par AMOSSYS, cf site ANSSI (cible sécurité, documentation, rapport évaluation)

1 Supervision de sécurité

2 Sonde NIDS

Snort

Bro

3 Architecture de supervision

4 Outils de détection d'intrusions

Sondes NIDS

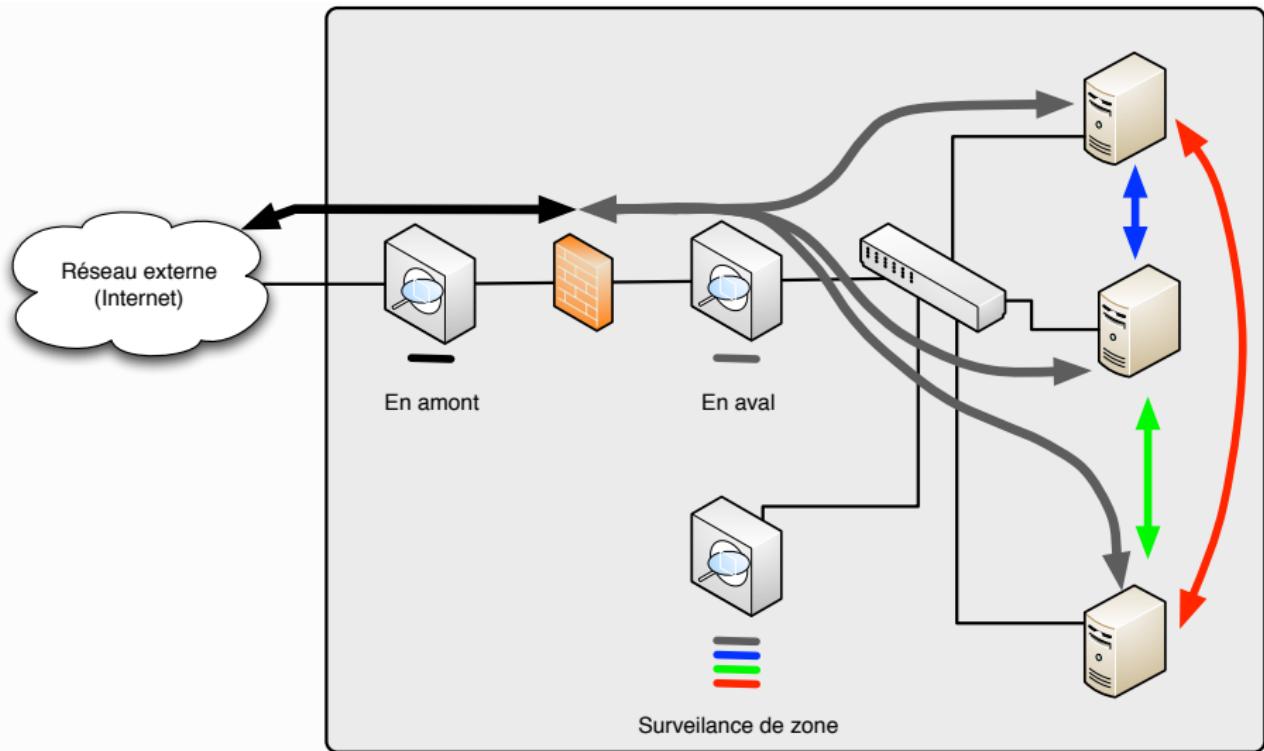
Sondes HIDS

- D'abord, définir ses besoins (politique de sécurité, analyse de risques / d'impacts)
- Choix des sondes :
 - Capacité de détection/prévention
 - Robustesse (attaques contre l'IDS, sûreté de fonctionnement)
 - Performances (paquets/s, connexions simultanées, etc.)
 - Facilité de gestion (documentation, support technique, services, mise à jour, interopérabilité)
 - Coût durant tout le cycle de vie (initial + maintenance)
- Le choix de l'emplacement est crucial
- Interface de capture
 - Capture passive = mode *promiscuous* (pas d'adresse pour l'interface de capture)
 - Mode *inline*, routeur (adresse pour l'interface de capture)
- Au moins 2 interfaces (capture + administration)
 - Garantir la furtivité (capture passive)
 - Vulnérabilités de l'interface d'administration

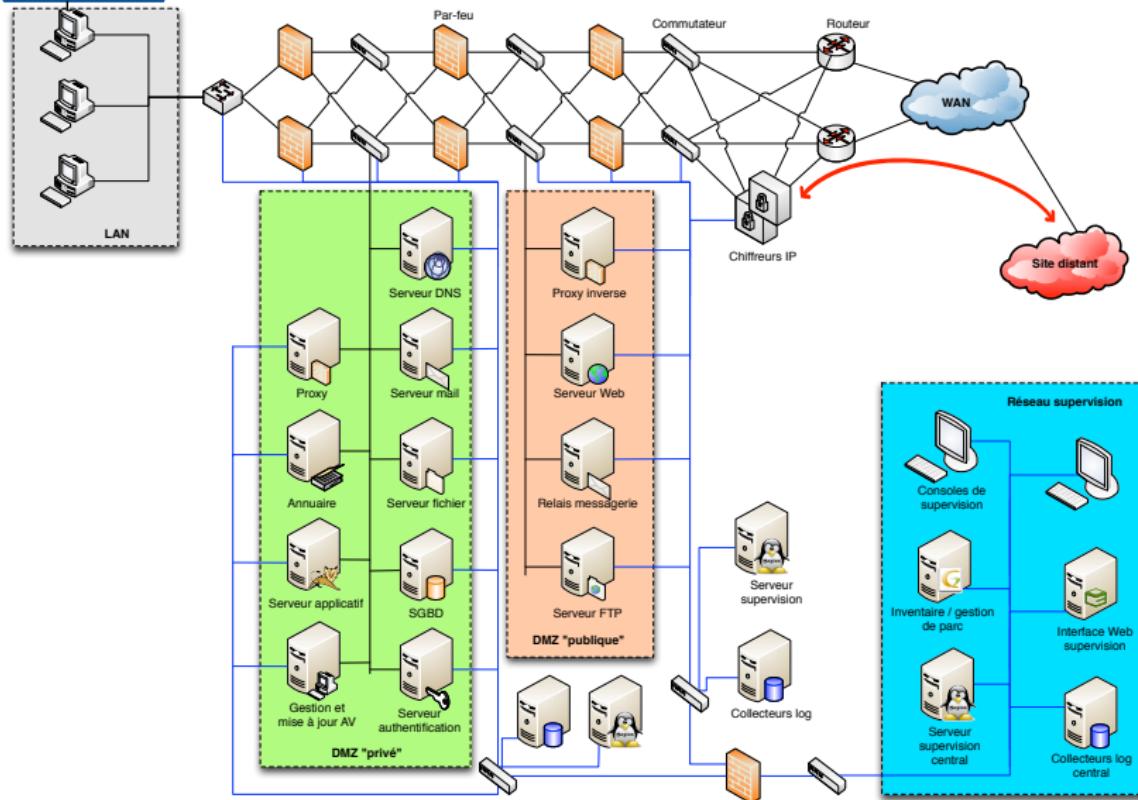
- Capture passive
 - + furtivité, surveillance au sein d'une zone (via *hub* ou *switch*), non-intrusive (perturbation limitée du réseau surveillé)
 - évasion, réaction moins facile
- Capture *inline*
 - + facilite la réaction (IPS), normalisation
 - non-furtif, impact les performances, trafic entre zones
- Capteur passif :
 - *hub* : uniquement pour petit réseau/faible débit
 - *switch* : miroir de port (*spanning / replication port*)
 - + pas de matériel supplémentaire, surveillance de zone
 - limité en débit, risque de perte, risque de mauvaise configuration
 - TAP
 - + pas de problème de débit, pas de perte sur la transmission
 - coût, utilisation sur un brin (surveillance inter-zone, cf mode *inline*)
- Remarque : *inline* \neq blocage, possibilité de ne bloquer que certaines alertes (impact de faux +)

- IPS \simeq pare-feu : cloisonner des zones de confiance \rightarrow positionnement aux interfaces, en coupure
- IDS = surveiller une zone
- Définition des zones d'intérêt à surveiller
 - Intérêt : $I = \frac{V}{C}$
 - Valeur : $V = f(\text{impact}_{CID})$
 - Confiance : $C = f(\text{niveau protection, vulnérabilité}) = ?$
- Autre approche (complémentaire) : cloisonner en fonction des applications (pour spécialiser les sondes)
- Placement / pare-feu
 - ① En amont :
 - + visibilité maximale, connaissance de la menace
 - beaucoup de trafic à analyser, risque d'alertes peu pertinentes, risque d'évasions
 - ② En aval :
 - + moins de trafic, protection, vérification de la politique du FW
 - certaines attaques sont filtrées
 - ③ A la place (mode *inline*) : revient au cas 1 ou 2 selon configurations

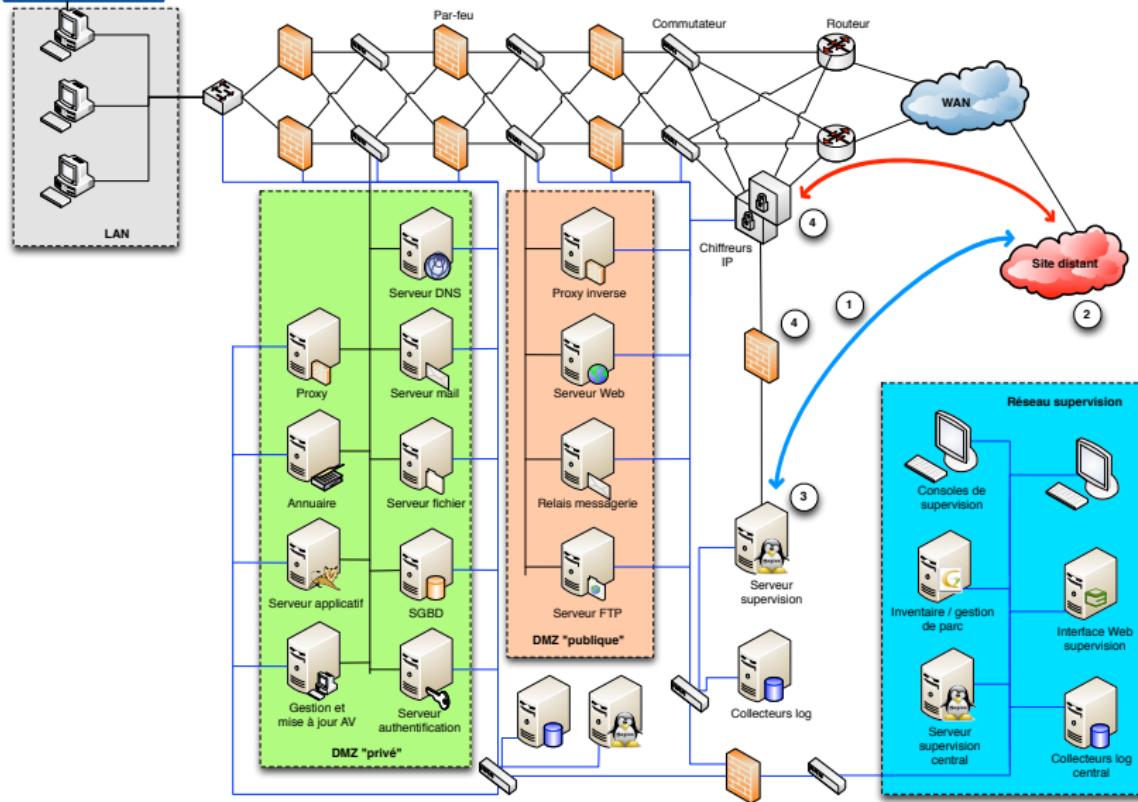
Les différents placements



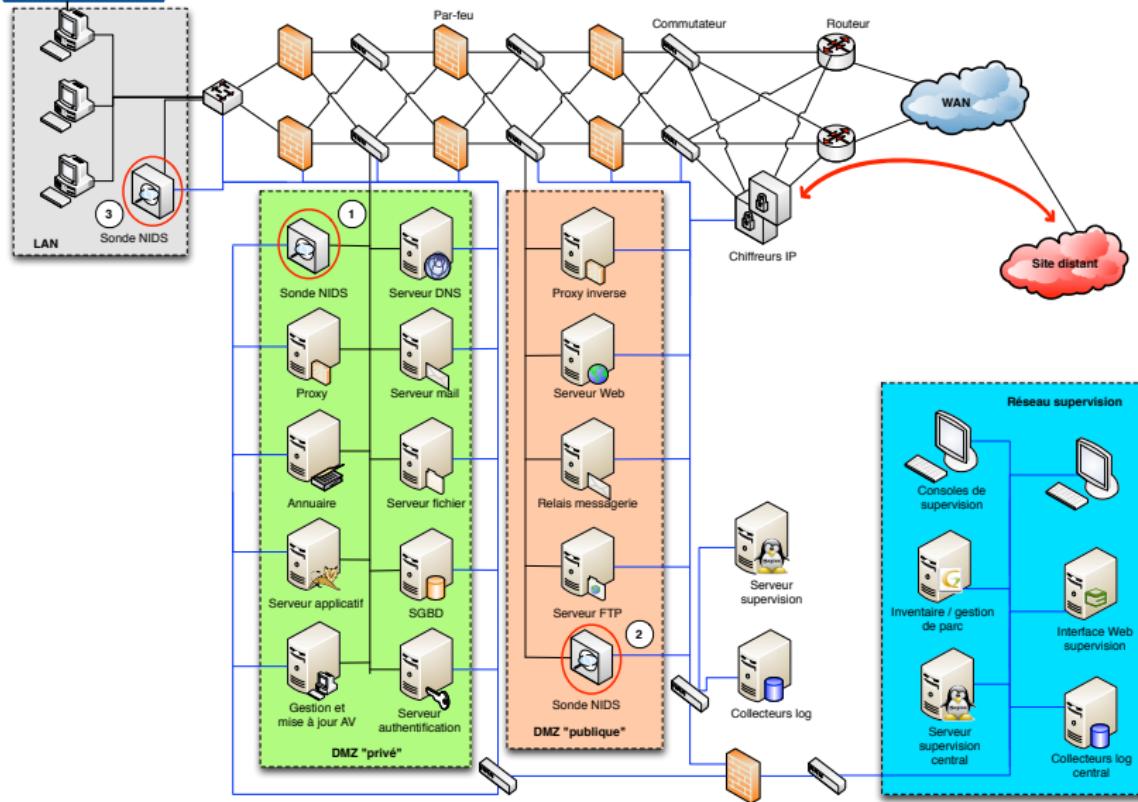
Exemple : architecture initiale



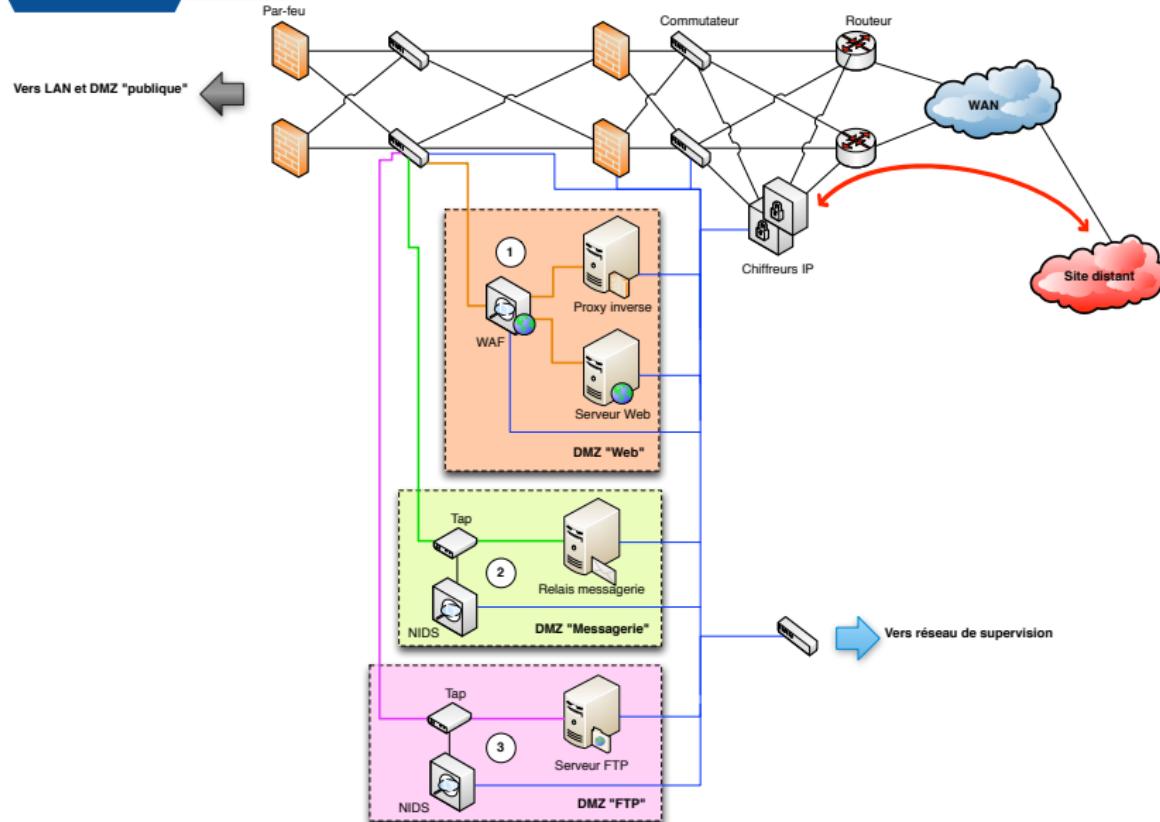
Exemple : liaison inter-sites



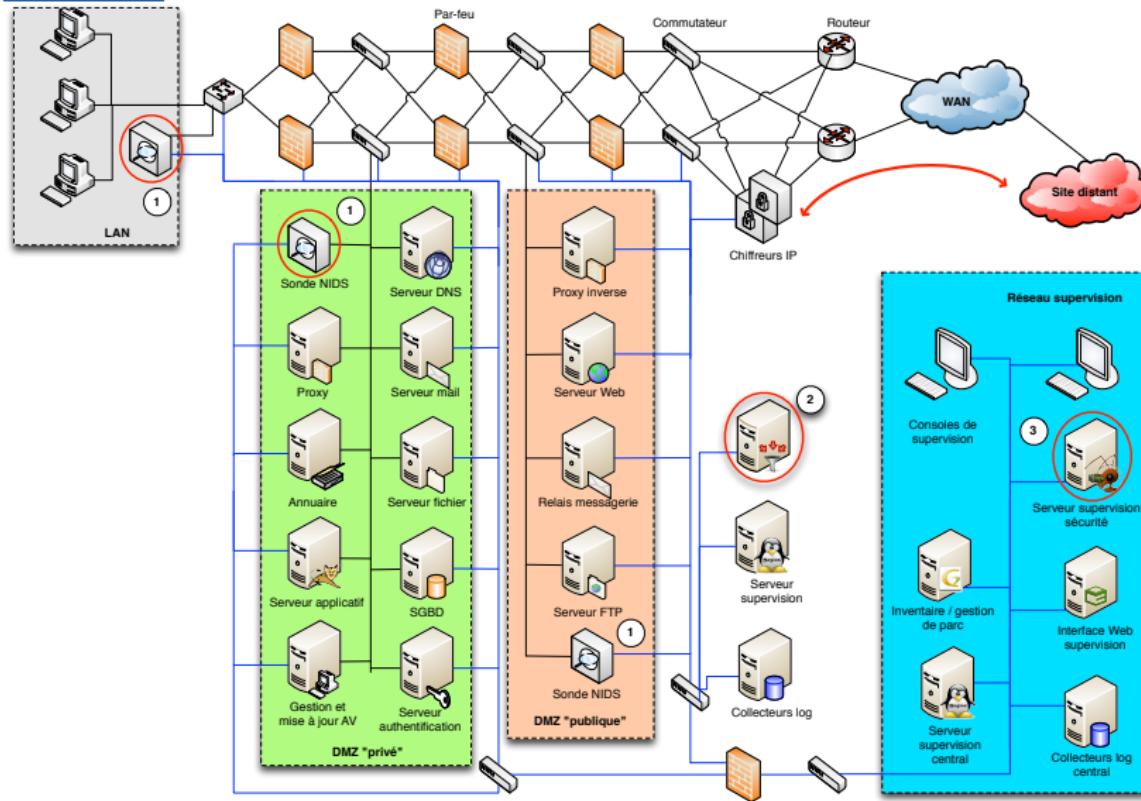
Exemple : sondes NIDS passives



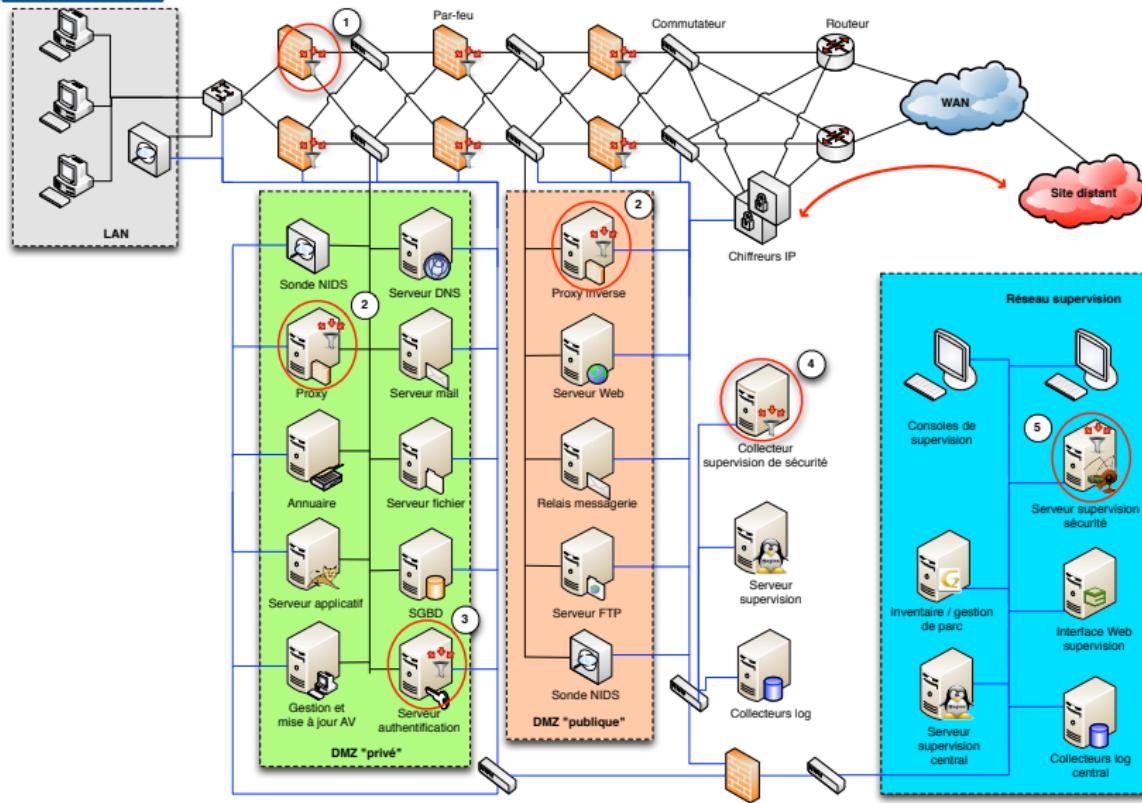
Exemple : cloisonnement horizontal



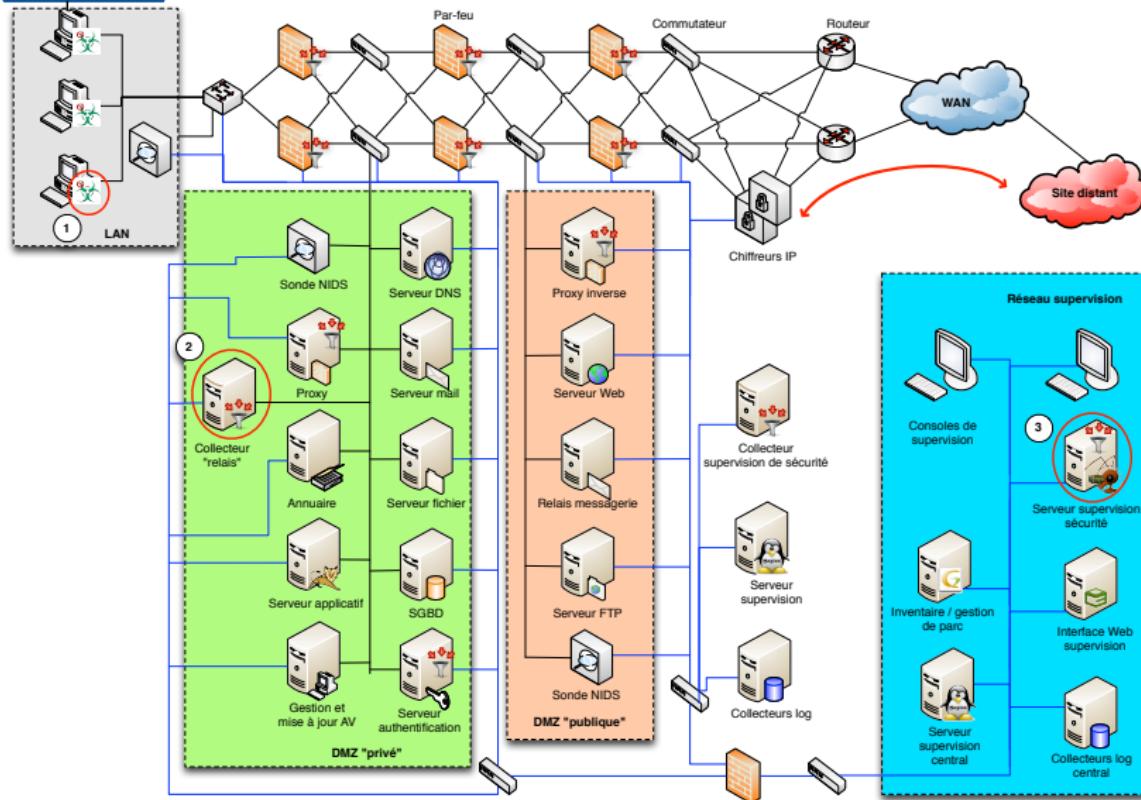
Exemple : collecte des alertes (SIEM)



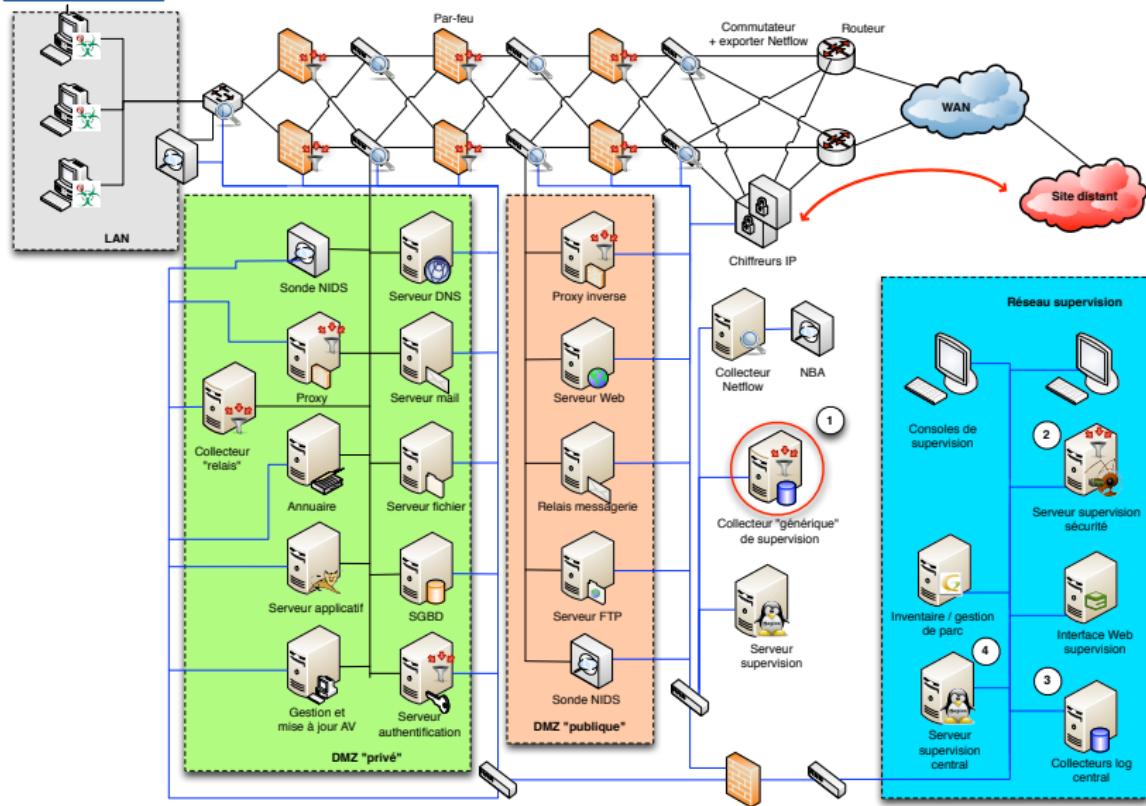
Exemple : autres événements de sécurité



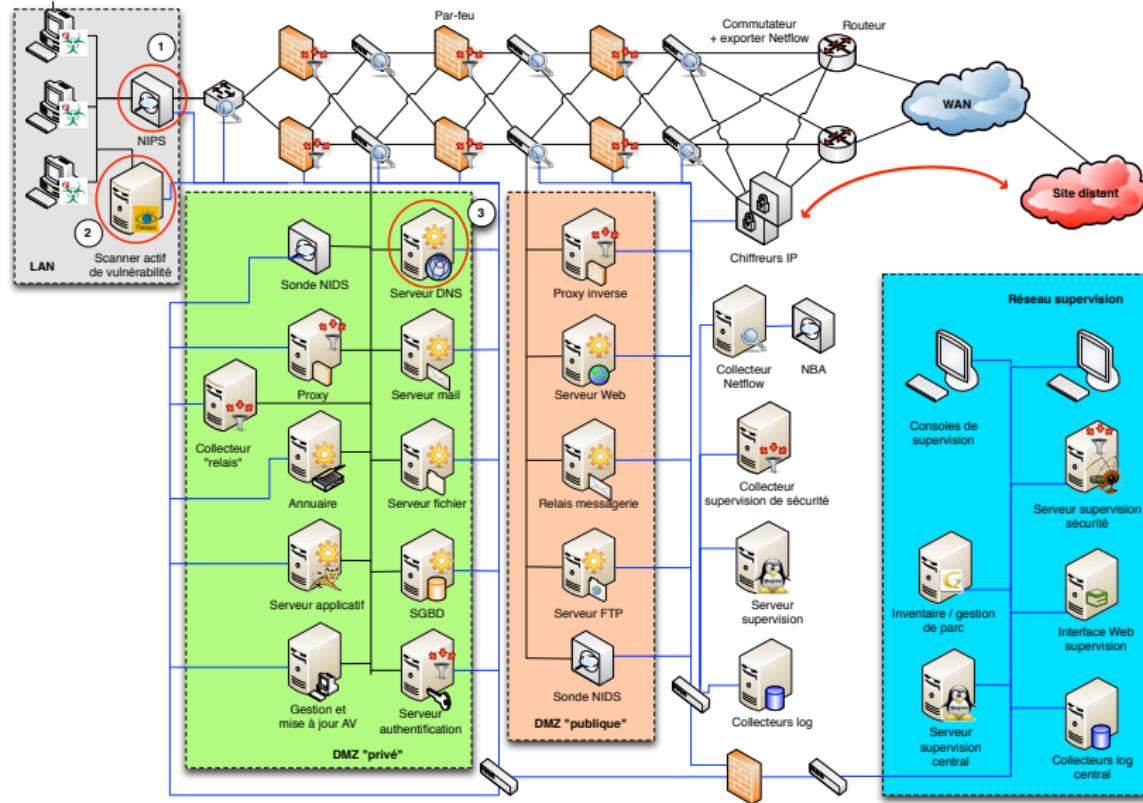
Exemple : collecte out-of-band



Exemple : intégration avec la supervision fonctionnelle



Exemple : HIDS, NIPS et scanner de vulnérabilité



1 Supervision de sécurité

2 Sonde NIDS

Snort

Bro

3 Architecture de supervision

4 Outils de détection d'intrusions

Sondes NIDS

Sondes HIDS

Reconnaissance de motifs

- Technique la plus courante (Snort, etc.) et la plus « mature »
- Importance des signatures
 - Qui fournit les signatures ? Abonnement ?
 - Les signatures sont-elles modifiables, redistribuables ?
 - Différence d'expressivité entre les signatures propriétaires et celles que l'utilisateur peut définir.
- Sur quoi porte la recherche :
 - Des paquets ?
 - Des flux TCP ?
 - Des contexts applicatifs (URI, etc.) ?
- Complexité : reconnaissance mono ou multi-événementielle, prise en compte de la réponse (*cf. corrélation*)
- Efficace pour détecter un exploit particulier → détecteur d'attaques
- Peu robuste à l'évasion

Détection d'anomalies de protocoles

- Détection de non-conformités à la spécification (RFC), valeurs suspectes ou incohérentes de certains champs
- Nécessite un décodage précis (couteux)
- Limites : performance, hétérogénéité des implémentations des piles protocolaires, protocoles propriétaires

Détection d'anomalies réseau

- Détection de l'importance anormale de certains flux
- Simple mais assez efficace pour détecter certaines attaques :
 - scans (*one to many*), DDOS (*many to one*)
 - propagation de vers, *backdoors*, etc.
- Spectre limité mais ne dépend pas d'un exploit particulier
- Limites : apprentissage, faux +, difficulté du diagnostic

Blacklistage d'adresses IP

Solutions dédiées : plateformes matérielles dédiées (*appliance*)

- Souvent PC avec un IDS logiciel (Snort ou autre)
- Parfois logique câblée (ASIC, FPGA), carte d'acquisition optimisée
- Plus simple à déployer, composants dédiés et optimisés

Solutions intégrées

- Modules matériels ou logiciels (*blade*) pour switch ou routeur
- Universal Threat Management (UTM) : pare-feu + VPN + IPS +...
- Solutions « tout-en-un » mais des ressources à partager
- Selon NSS Labs, performances proches des *appliances* dédiées

Solutions "ad hoc"

- Network Behavior Analysis (NBA) : détection d'anomalies réseau
- Wireless IPS (WIPS) : détection d'anomalies protocolaires (WIFI)
- Web Application Firewall (WAF) : HTTP et applications Web

Les produits commerciaux (*appliance* et UTM)

- McAfee, Cisco, IBM (ISS), Sourcefire, CheckPoint (NFR), HP (TippingPoint), Palo Alto, Stonesoft, Enterasys (Dragon), Radware, TopLayer, iPolicy
- Des *appliances* à base de Snort : Fortinet, Nitro Security, e-Cop Cyclops, StillSecurev, Endace, autres (pas toujours affiché...) ?
- Les leaders selon Gartner : McAfee, HP, Sourcefire
- Les meilleurs selon NSS Labs : McAfee (Intel), IBM, Sourcefire

Les logiciels open-source

- Snort (Sourcefire)
- Bro (LBNL)
- Suricata (OISF, Homeland Security)

1 Supervision de sécurité

2 Sonde NIDS

Snort

Bro

3 Architecture de supervision

4 Outils de détection d'intrusions

Sondes NIDS

Sondes HIDS

- Contrôle d'intégrité (fichiers, base de registre, noyau, etc.)
 - Vérification (trop) simple
 - Prise en compte des changements légitimes (mise-à-jour, etc.)
- Recherche de motifs dans les journaux (*cf. limites des NIPS par signatures*)
- Heuristique : détection de malware, rootkit
 - Lien avec les Anti-Virus
- Analyse du comportement des applications (*spec-based, policy-based*)
 - Lien avec certains mécanismes de contrôle d'accès mandataire (SE-Linux, AppArmor, Blare, etc.)
 - Nécessite d'avoir des politiques ou des profils corrects et complets
 - Attention aux faux + si profils ou politiques trop restrictifs
- Analyse du comportement des utilisateurs

Produits commerciaux

- Contrôle d'intégrité
 - Tripwire
 - NetIQ Change Guardian
- Protection serveur
 - Trend Micro (Third Brigade) Deep Security (recommandé par Cisco depuis fin CSA...)
 - eEye Blink Server Edition
- Protection du poste client
 - cf. « suites de sécurité » (*Endpoint Protection*)
 - Anti-Virus, Firewall personnel, HIPS, Anti-malware, etc.
 - AVG, Avira, Kaspersky, Symantec, ESEC, Sophos, McAfee, etc.
 - « Firewall personnel ++ » :
 - SoftSphere DefenseWall
 - Online-Armor
 - Emsi Software Mamutu

Logiciels libres

- Détection de motifs simples dans les journaux
 - Prelude LML (CS)
- Détection motifs + contrôle intégrité + détection rootkit
 - Samhain
 - OSSEC (TrendMicro)
 - OSIRIS
- Contrôle intégrité simple
 - AIDE

Etat art technique

Guide to Intrusion Detection and Prevention Systems (IDPS). Karen A. Scarfone, Peter M. Mell. NIST SP - 800-94.

Corrélation d'alertes

Christopher Kruegel, Fredrik Valeur and Giovanni Vigna. Intrusion detection and correlation. Springer. ISBN 0-387-23398-9. 2005.

Article général sur les IDS

H.Debar, M.Dacier and A.Wespi. A Revised Taxonomy for Intrusion-Detection Systems. Annales des Télécommunications, Vol.55, No7-8, 2000.

En français ...

Marc Dacier. Chapitre « La détection d'intrusions » du traité IC2 sur la sécurité des systèmes d'informations. Hermes. ISBN 2-7462-1259-5. 2006.

Bibliographie

-  World intrusion detection and prevention systems markets, #N22B-74 - Frost & Sullivan, 2007.
-  David Bizeul and Olivier Badet, Systèmes de détection d'intrusion réseau, architecture et paramétrage, 2005.
-  Kathleen A. Jackson, Intrusion detection system (ids) product survey, Tech. Report LA-UR-99-3883, Los Alamos National Laboratory, June 1999.
-  NSS Labs, Network ips group test q4 2010, 2010.
-  Karen Scarfone and Peter Mell, Guide to intrusion detection and prevention systems (idps), Recommendations of the National Institute of Standards and Technology (NIST Special Publication 800-94), February 2007.

Bibliographie (cont.)



Greg Young and John Pescatore, Magic quadrant for network intrusion prevention systems, Gartner RAS Core Research Note G00208628, décembre 2010.

Remerciements - crédits

- Ludovic Mé, *détection d'intrusions* (transparents), Supélec, 2011.
- Steve Sturges, *Snort 2.8.5 Overview*, SourceFire
- Steve Sturges, *Snort 2.8.4 Overview*, SourceFire
- Steve Sturges, *Performance Tuning Snort*, SourceFire
- Soumya Sen, *Performance Characterization and Improvements of SNORT* Lucent Technologies, July 2006