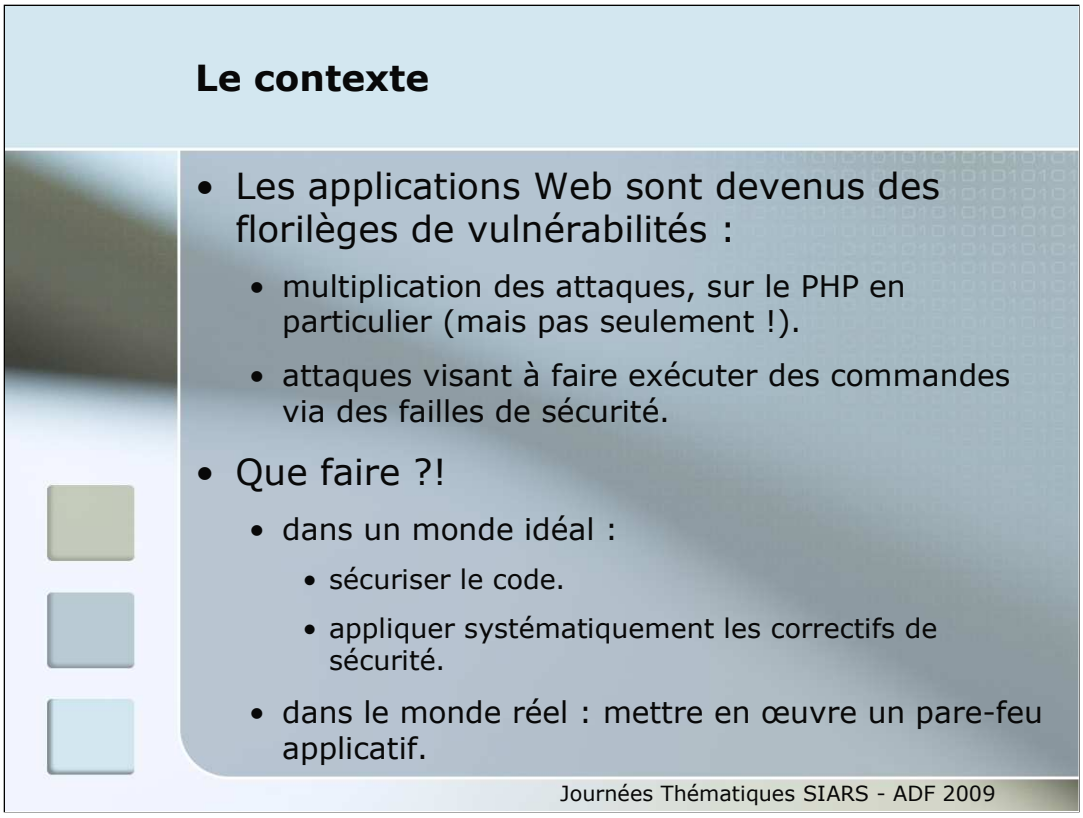
ADF 2009
Apache : ModSecurity 2.x

Thierry DOSTES
tdostes@ifr88.cnrs-mrs.fr

Plan

Le module ModSecurity :

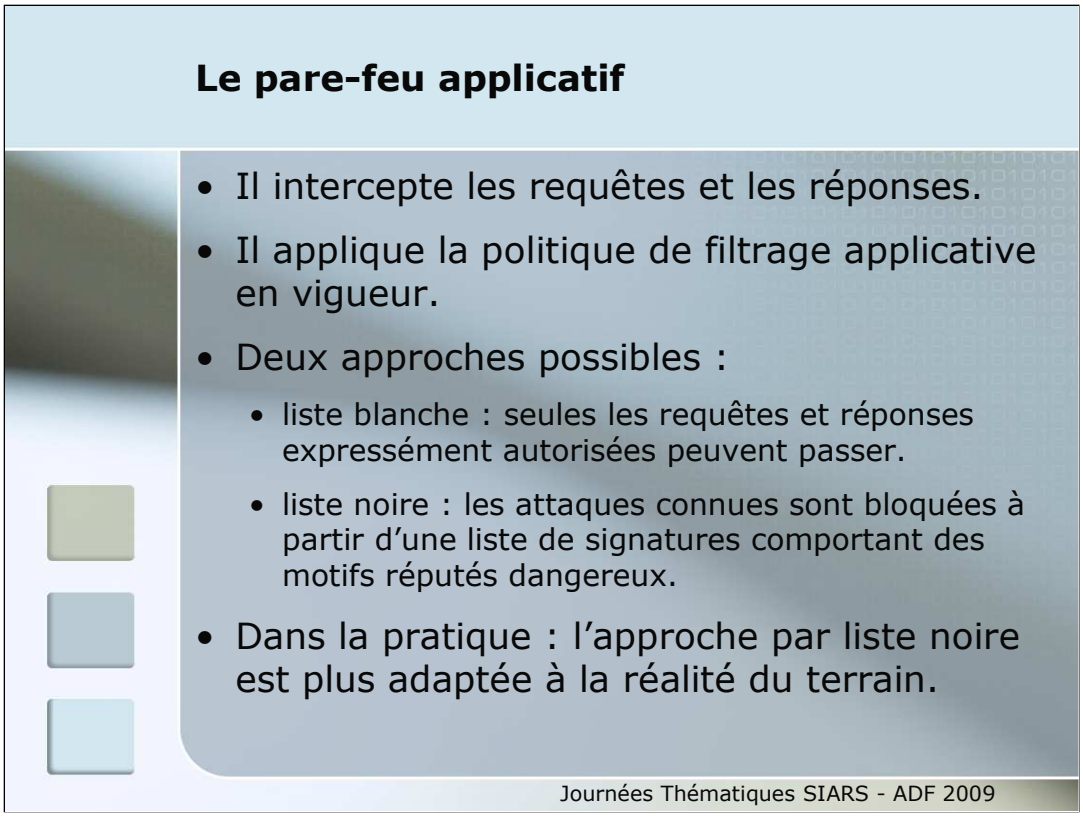
- Présentation.
- Les 5 phases d'intervention.
- Les règles de filtrage.
- Configuration.
- Les « ModSecurity Core Rules ».
- Conclusions.



Le pare-feu applicatif apportera un début de réponse à nos problèmes de sécurité. Il s'agit d'une rustine qui va apporter seulement une réponse partielle.

Elle ne dispense pas de sécuriser correctement le code. Elle présente toutefois l'avantage de protéger dans une certaine mesure des applications Web sur lesquelles nous ne pouvons intervenir.

Les pratiques de développement doivent veiller à minimiser la surface d'attaque des applications. Un pare-feu applicatif ne se substituera jamais aux développeurs d'une application. Seuls ces derniers en comprennent tous les rouages et sont en mesure de palier à ces failles.

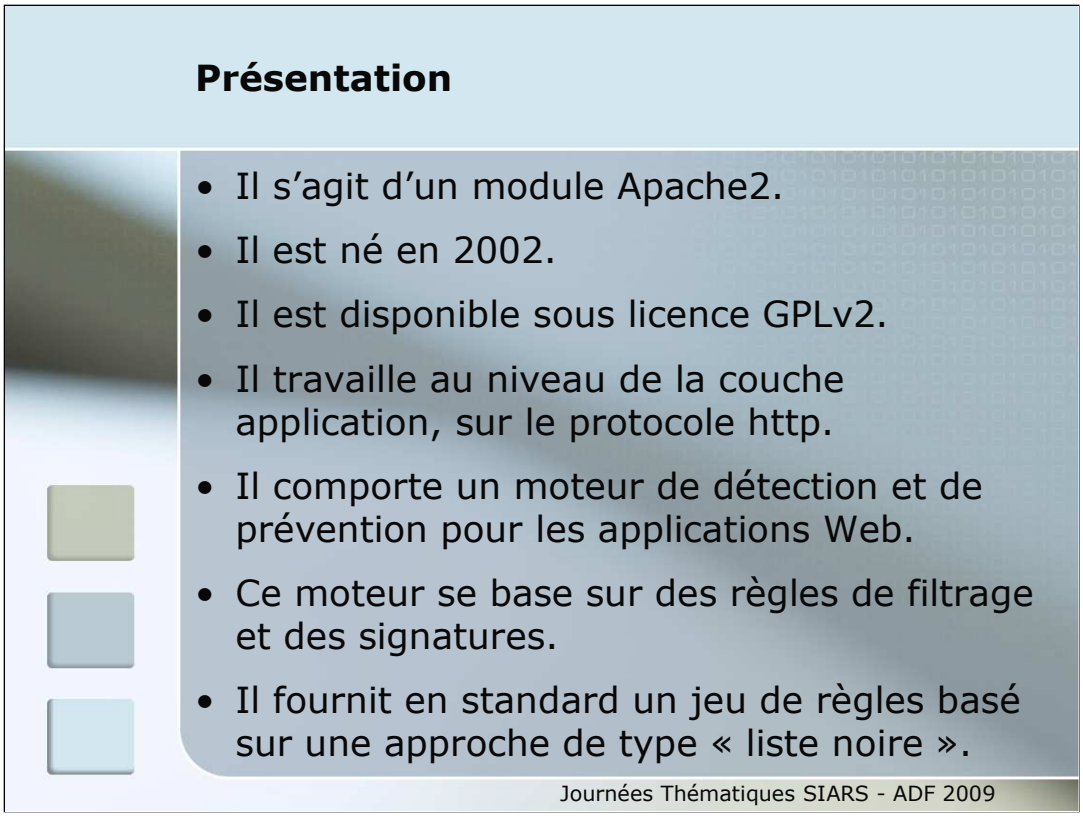


L' approche de type liste blanche est à réserver à des applications dont on maîtrise tous les rouages et qui évoluent très peu dans le temps.

Faute de quoi, on aboutit très vite à des règles de faible granularité, donc trop permissives.

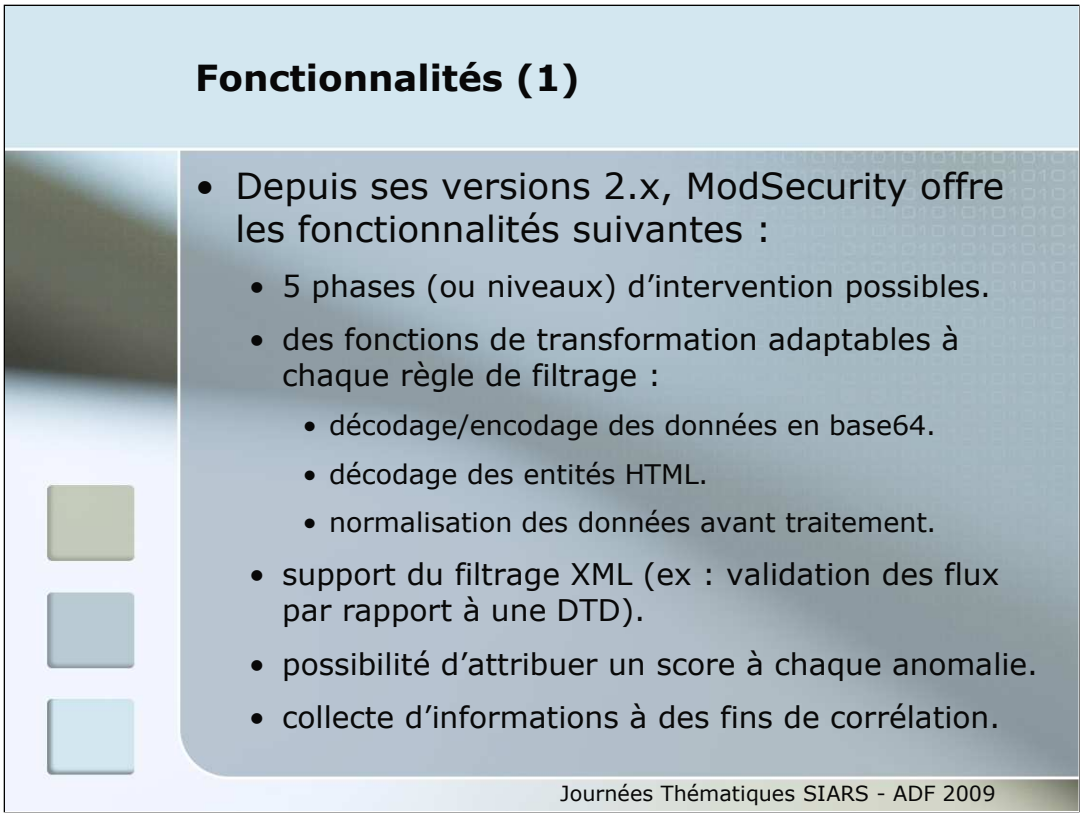
L'approche de type liste noire est plus adaptée à la réalité du terrain, même si, de prime abord, elle peut paraître inadaptée pour les plus paranoïaques d'entre nous.


Présentation de
ModSecurity
cnrs
dépasser les frontières



ModSecurity a connu un succès grandissant, relayé par une communauté d'utilisateurs compétente. Il a fini par supplanter certaines solutions commerciales.

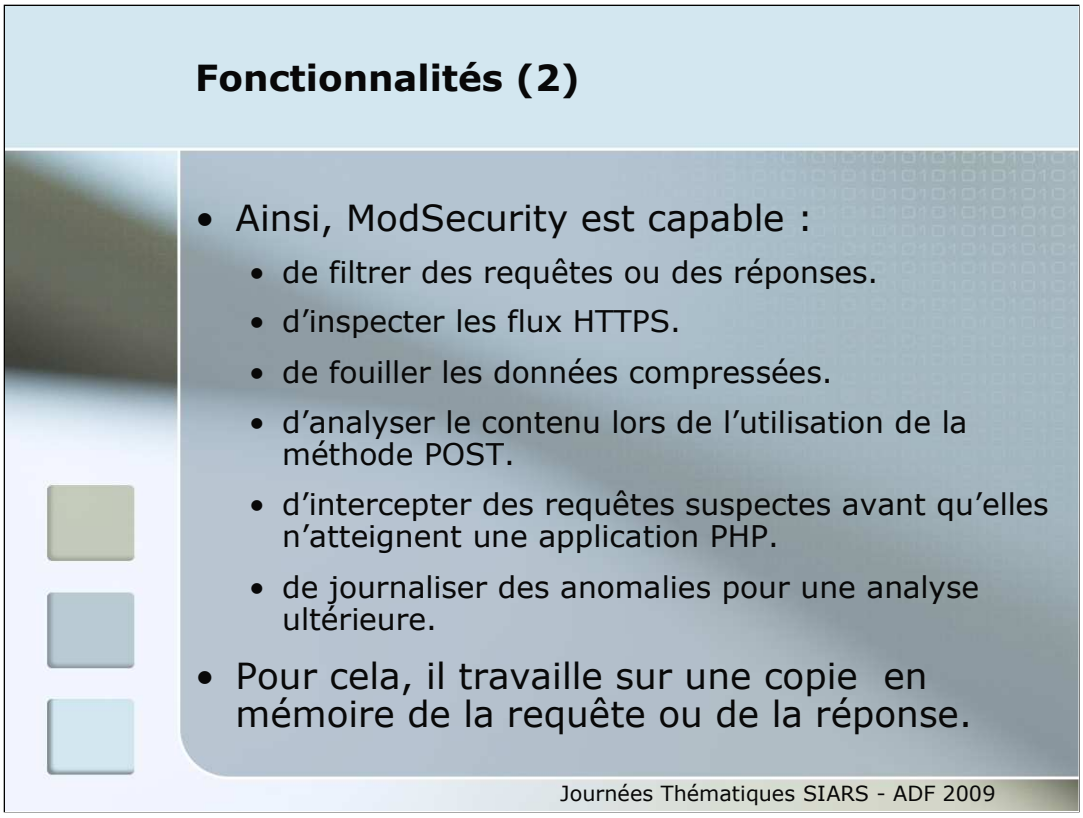
Il est disponible sous licence GPLv2 ainsi que sous licence commerciale, notamment pour fournir un support à ces utilisateurs.



Nous allons étudier plus en détails ces 5 phases dans les prochaines diapos.

Le score attribué à une anomalie pourra servir de déclencheur pour journaliser et/ou rejeter une requête/réponse.

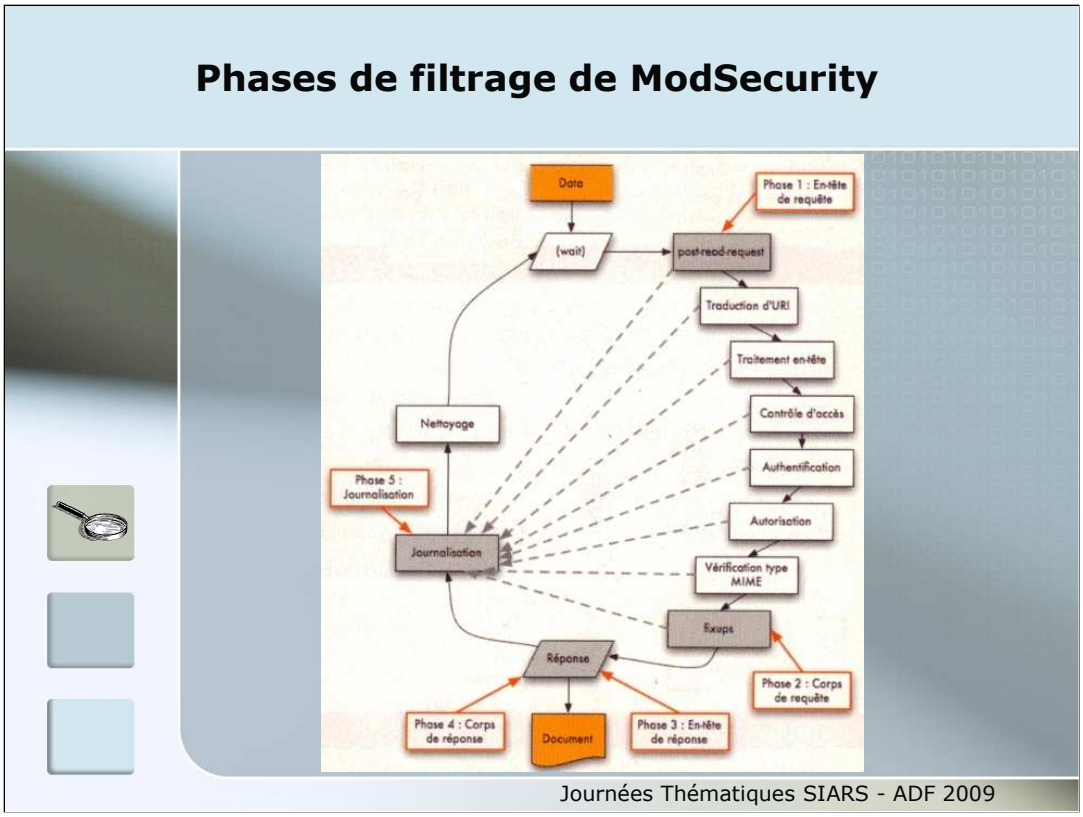
ModSecurity peut désormais collecter des données de manière persistante pour corréler les événements générés depuis certaines adresses IP, des sessions ou par des utilisateurs.



ModSecurity effectue une copie en mémoire de la requête ou de la réponse. Il travaille sur cette copie grâce aux différentes fonctions de transformations (cf diapositive précédente).

ModSecurity « transforme » les valeurs des variables avant de les tester à l'aide des règles.


Les 5 phases de
ModSecurity
cnrs
dépasser les frontières



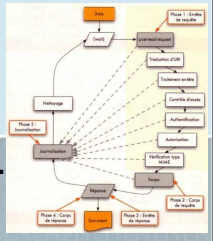
Le filtrage peut intervenir non seulement sur les entêtes et corps de requête, mais aussi sur les entêtes et les corps des réponses.

Toutes ces « interventions » peuvent être journalisées.

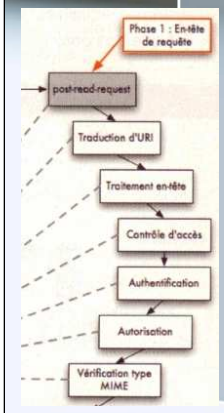
L'administrateur peut créer des règles et les faire s'appliquer au niveau qu'il souhaite.

La priorité d'application d'une règle est fonction d'une phase. Une règle de phase *n* est prioritaire par rapport à une règle de phase *n+1*.

Phase 1 : entête de la requête



- Apache vient de lire l'entête de la requête.
- Il n'a pas encore lu son contenu.
- Il ne connaît pas encore les arguments contenus dans la requête.
- Toute règle s'exécutant en phase 1 intervient avant qu'Apache soit en mesure de faire quelque chose.



Exemples de règles en phase 1:

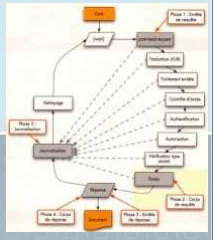
- décider si le corps de la requête doit être traité plus en détail.
- définir comment le corps doit être traité (en XML ?)

Attention : les règles de phase 1 ne peuvent s'appuyer sur les directives de portée d'Apache (Directory, Location, etc.).

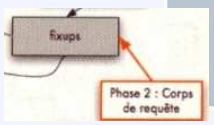
Journées Thématiques SIARS - ADF 2009

Pour utiliser des règles ModSecurity en corrélation avec des espaces Web définis par Apache, celles-ci doivent être définies en phase 2.

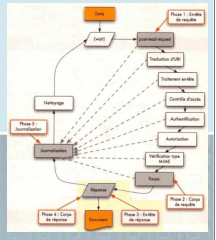
Phase 2 : corps de la requête



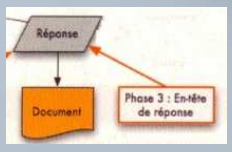
- Nous disposons désormais des arguments de la requête.
- Les règles de filtrage ou de rejets liées à des applications doivent intervenir à ce niveau .
- ModSecurity connaît trois types d'encodage pour analyser le contenu du corps d'une requête.
  - application/x-www-form-urlencoded
  - multipart/form-data
  - text/xml
- Les autres types sont rarement utilisés par les applications Web.



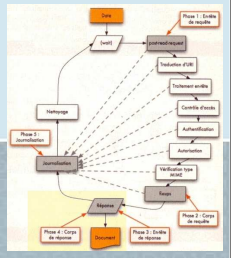
Phase 3 : entête de la réponse



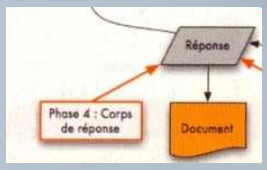
- Cette phase intervient juste avant que que les entêtes des réponses parviennent au client.
- Les règles de filtrage permettent de définir ce qu'il doit advenir de la « future » réponse.
- On décide si l'on veut analyser le contenu/corps de la réponse ou la bloquer.
- A ce niveau, nous ne savons pas encore ce que le serveur Apache va retourner.



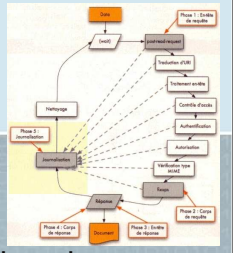
Phase 4 : corps de la réponse



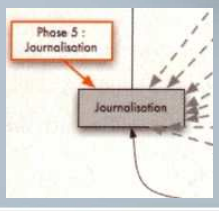
- A ce niveau, ModSecurity analyse les informations renvoyées à destination du client.
- Le contenu HTML est analysé pour détecter :
  - des fuites d'informations.
  - des messages d'erreurs.
  - des traces d'authentifications ayant échouées.
  - etc.



Phase 5 : journalisation



- Les règles déclarées à ce niveau interviennent seulement sur la manière dont la journalisation doit s'opérer.
- Cette phase peut être utilisée pour analyser les messages d'erreur enregistrés par Apache.
- Elle permet également d'inspecter des entêtes de réponse qui n'étaient pas accessibles en phases 3 et 4.
- Il est trop tard pour interdire ou bloquer des connexions.




Les règles selon
ModSecurity
cnrs
dépasser les frontières

Le principe

- Nous pouvons créer des règles et définir à quelle phase elles interviendront.
- Il est également possible de définir des comportements par défaut.

  ex : définir l'action par défaut si aucune règle ne s'applique

```
SecDefaultAction "phase:2,log,pass,status:500"
```

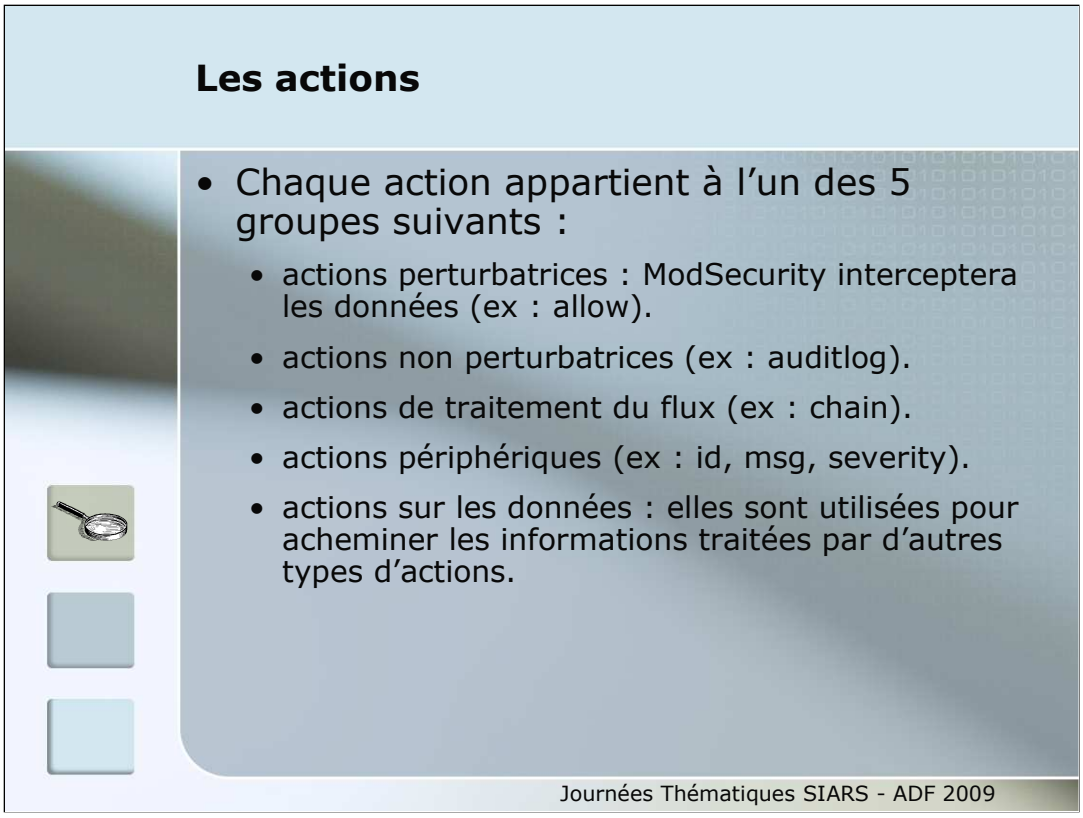
- Pour exprimer ces règles, nous disposons de variables et d'actions.

  ex : mettre une machine cliente en liste blanche.

```
SecRule REMOTE_ADDR "^10\.126\.100\.85$" phase:1,log,allow,ctl:ruleEngine=Off
```

Quelques exemples de variables

- **ARGS** : traite l'ensemble des arguments, ou l'argument spécifié en paramètre, ou tous les arguments sauf un.
- **AUTH_TYPE** : renvoie le type d'authentification utilisé.
- **HTTP_REFERER** : indique le site de provenance d'une requête.
- **REMOTE_ADDR** : l'adresse IP de la machine cliente.
- **REMOTE_HOST** : retourne le nom DNS de la machine cliente si l'option HostnameLookUp est activée.
- **REQUEST_HEADERS** : permet de travailler sur l'entête de la requête (ex : champ **Host**).
- **REQUEST_BODY** : accède aux données contenues dans le corps de la requête.
- **REQUEST_FILENAME** : retourne le nom du fichier appelé, sans les éventuels paramètres.
- **REQUEST_URI** : retourne le chemin d'accès au fichier appelé et les éventuels paramètres.



chain permet de concaténer plusieurs règles afin d'être en mesure de créer des filtres complexes prenant en compte plusieurs variables.

id permet d'attribuer un identifiant numérique à une règle.

msg lui attribue un message qui sera affiché dans les fichiers de traces.

Quelques exemples d'actions (1)

- allow : interrompt le traitement et autorise la transaction lorsque une règle s'applique.

```
SecRule REMOTE_ADDR "^10\.126\.100\.85$" phase:1,log,allow,ctl:ruleEngine=Off
```

- auditlog : journalise la transaction dans le fichier d'audit.

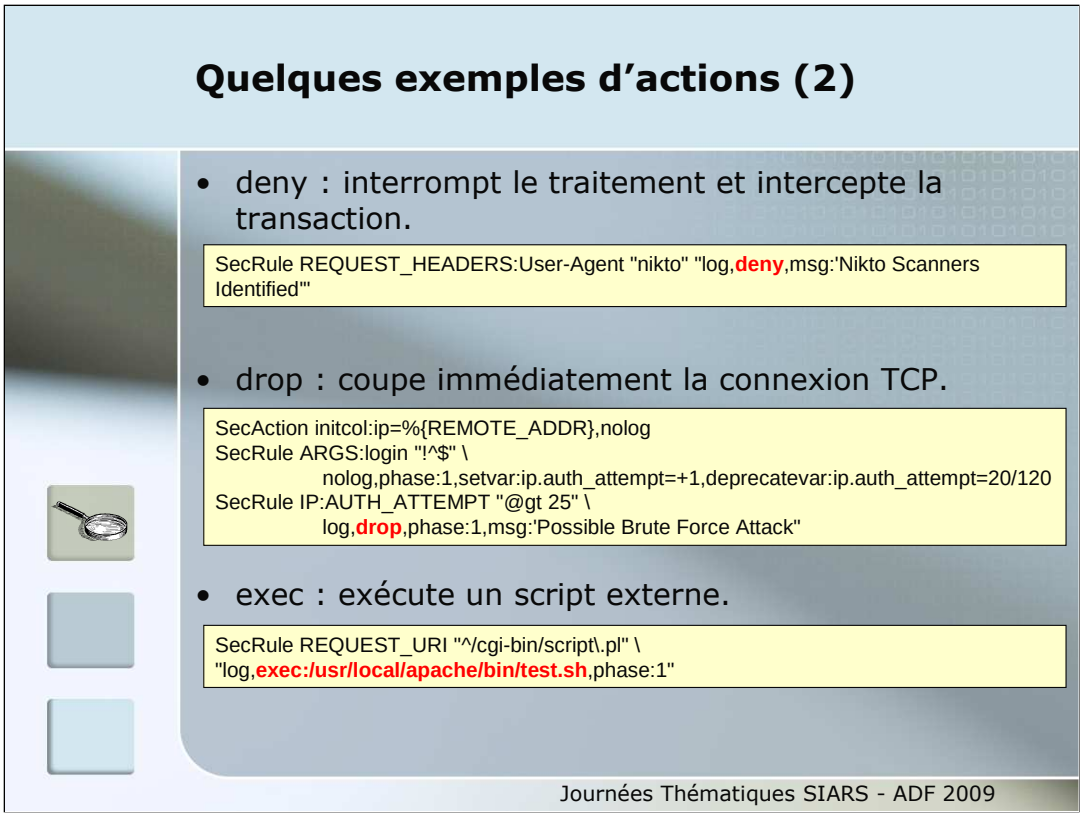
```
SecRule REMOTE_ADDR "^192\.168\.1\.100$" auditlog,phase:1,allow
```

- capture : déclenche la sauvegarde d'une transaction lorsqu'un modèle défini apparaît (10 captures possibles).

```
SecRule REQUEST_BODY "^username=(\w{25,})" phase:2,capture,t:none,chain SecRule
TX:1 "(?:(?:a(dmin|nonymous)))"
```

- ctl : permet de surcharger la configuration par défaut pour la règle courante :

```
SecRule REMOTE_ADDR "^10\.126\.100\.85$" phase:1,log,allow,ctl:ruleEngine=Off
```



Voici un exemple d'utilisation de la commande drop pour se protéger d'attaque en force brute. Nous enregistrons les tentatives d'authentifications auprès de notre serveur. Si le client distant essaye de se connecter plus de 25 en fois en moins de 2 minutes, ModSecurity rejettera toute nouvelle tentative de connexion.

Il est impossible de passer des paramètres aux scripts appelés par l'intermédiaire de la fonction exec. En revanche, l'ensemble des variables d'environnement CGI seront accessibles au programme externe. Celui-ci doit retourner quelque chose sur la sortie standard (stdout), faute de quoi ModSecurity considèrera que l'exécution a échoué.

Quelques exemples d'actions (3)

- log : journalise la transaction dans le fichier de traces.
- phase : précise la phase pour laquelle la règle devra s'appliquer.

```
SecRule REMOTE_ADDR "^192\.168\.1\.100$" phase:1,log,allow,ctl:ruleEngine=Off
```

- proxy : redirige la requête vers un autre serveur grâce au module proxy d'Apache.

```
SecRule REQUEST_HEADERS:User-Agent "Test" log,proxy:http://www.nowhere.com/
```

- severity : définit la gravité d'une alerte.

```
SecRule REQUEST_METHOD "^PUT$" "id:340002,rev:1,severity:2,msg:'Restricted HTTP function'"
```

- status : spécifie le status HTTP retourné au client.

```
SecDefaultAction log,deny,status:403,phase:1
```

Journées Thématiques SIARS - ADF 2009

Les niveaux de gravité possibles pour severity sont :

0 = EMERGENCY

1 = ALERT

2 = CRITICAL

3 = ERROR

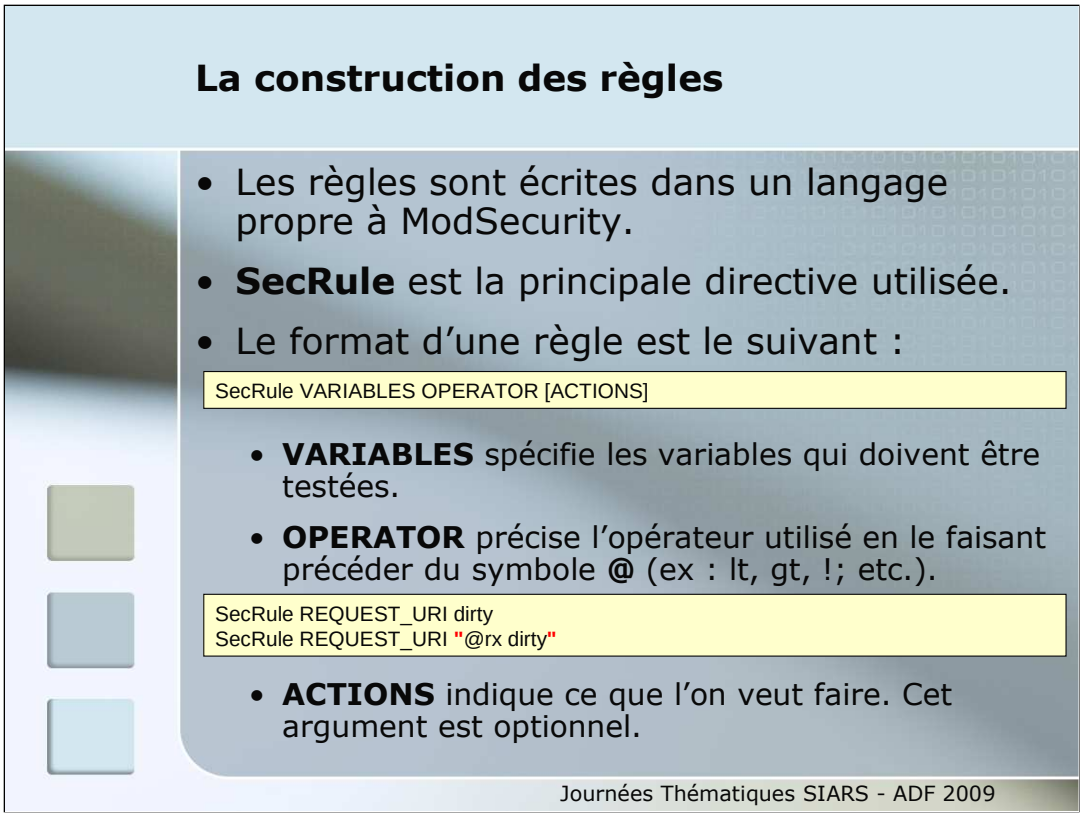
4 = WARNING

5 = NOTICE

6 = INFO

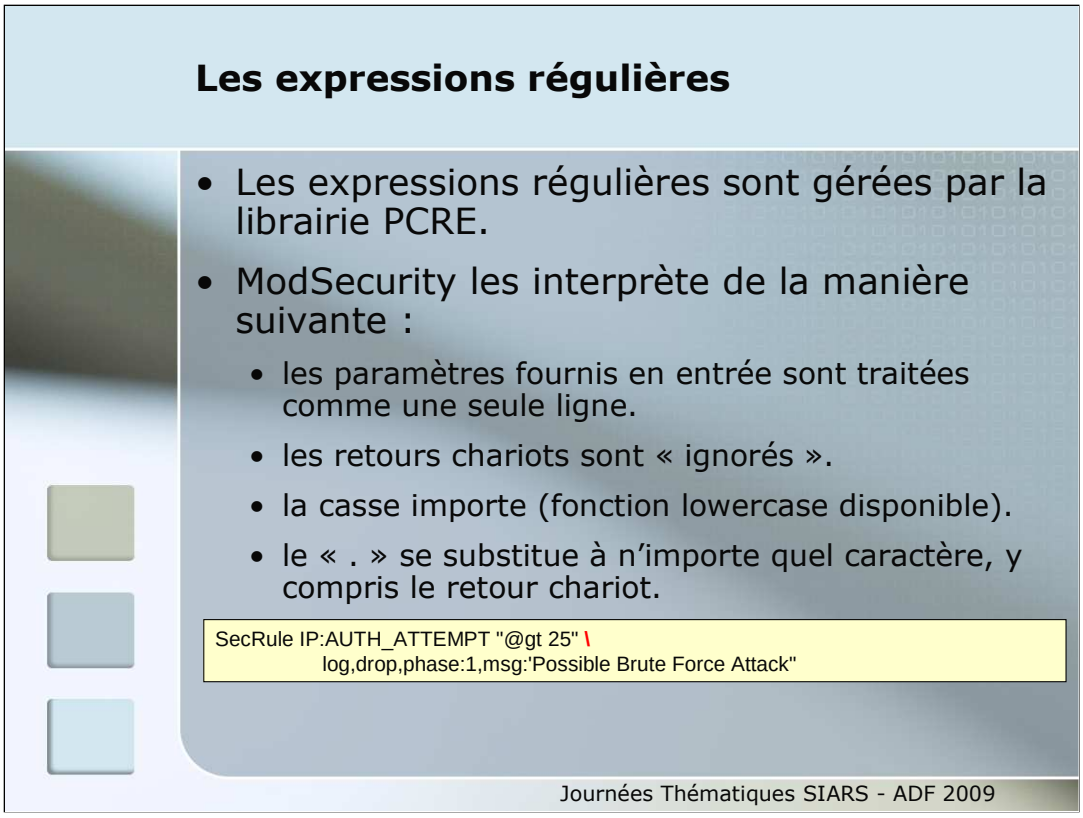
7 = DEBUG

Le code erreur HTTP 403 correspond à une action interdite. Le serveur HTTP a compris la requête, mais refuse de la traiter.
Ce code est généralement utilisé lorsqu'un serveur ne souhaite pas indiquer pourquoi la requête a été rejetée.



Si aucun opérateur n'est spécifié, c'est l'opérateur **rx** qui est utilisé par défaut. Il permet de déclarer une expression régulière.

Dans l'exemple ci-dessus, nous avons utilisé des guillemets pour la déclaration du second paramètre (OPERATOR) car il contient un espace. Dans le cas contraire, ModSecurity considèrerait que « dirty » est le troisième argument de la directive SecRule, donc une action.



PCRE considère que le « . » se substitue à tout caractère, à l'exception du retour chariot.


La configuration
de ModSecurity
cnrs
dépasser les frontières

Configuration de ModSecurity (1)

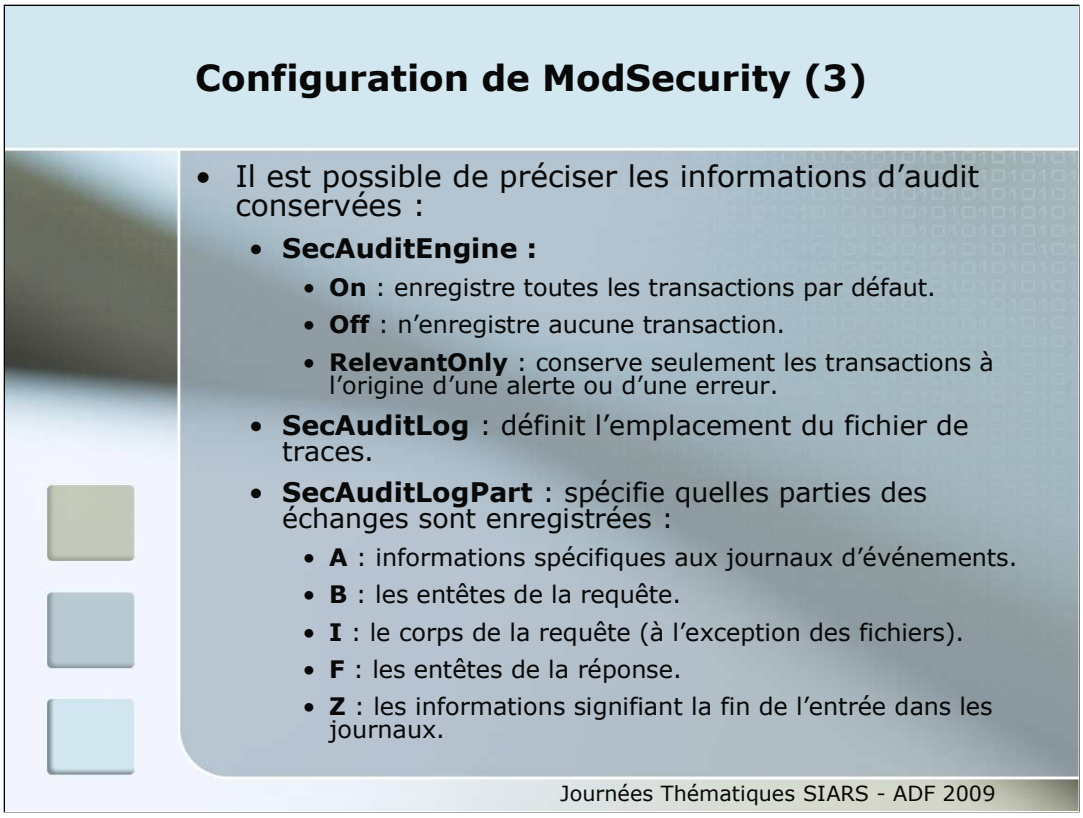
- Le fichier **modsecurity_crs_10_config.conf** définit le comportement par défaut de l'outil :
  - **SecRuleEngine :**
    - **On** : interception des échanges et application des règles de filtrage.
    - **DetectionOnly** : application des règles sans interception.
    - **Off** : désactivation de la surveillance.
  - **SecRequestBodyAccess On** : activation des règles permettant l'inspection du corps des requêtes.
  - **SecResponseBodyAccess On** : même chose pour le corps des réponses.
  - **SetDefaultAction** : permet de déterminer l'action par défaut si aucune règle ne s'applique.

Configuration de ModSecurity (2)

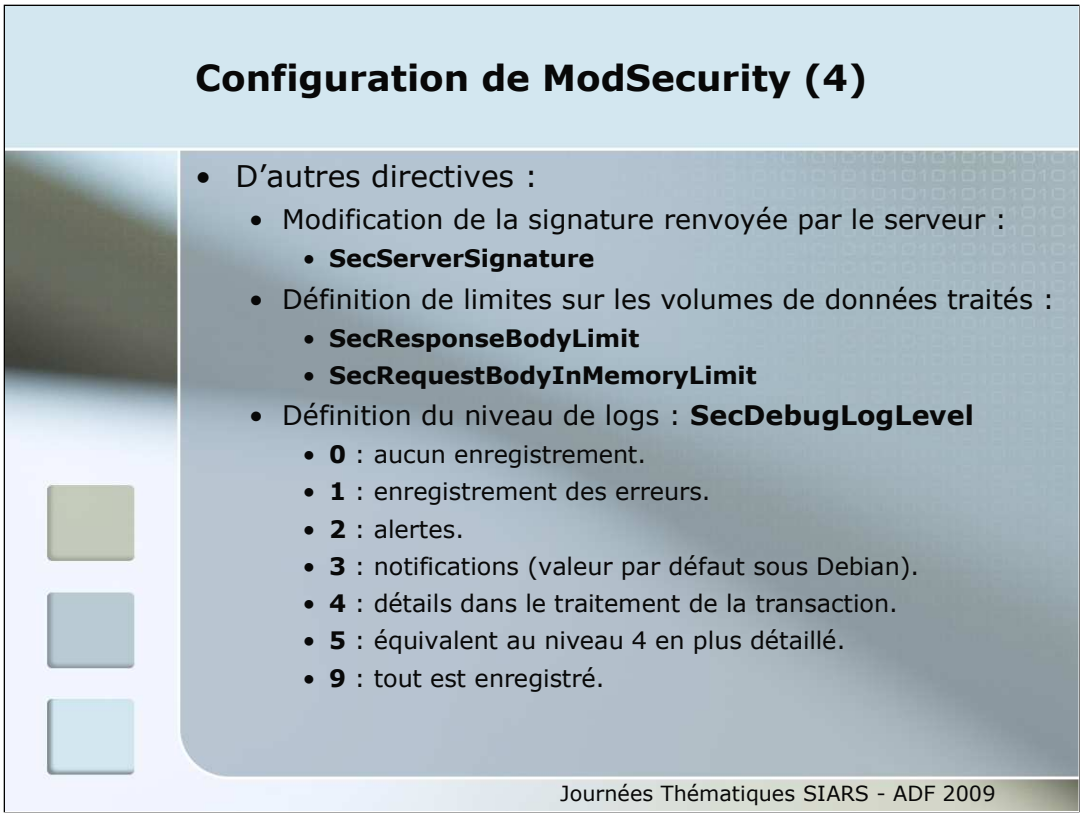
- Nous pouvons spécifier plusieurs actions par défaut :

```
SecDefaultAction log,auditlog,deny,status:403,phase:2,t:lowercase
```

  - **log** : toute transaction est enregistrée dans le fichier de traces.
  - **auditlog** : toute transaction est conservée dans le fichier d'audit de ModSecurity.
  - **deny** : intercepte la transaction.
  - **status:403** : précise le code HTTP retourné au client dès qu'une règle correspond.
  - **phase:2** : indique la phase lors de laquelle s'applique cette règle par défaut.
  - **t:lowercase** : active la transformation de tous les caractères d'une requête en minuscules.



ModSecurity permet d'enregistrer beaucoup plus d'informations que le serveur Apache dans les fichiers de traces. Nous disposons ainsi d'informations d'audit beaucoup plus détaillées. Voyons comment définir ce que nous souhaitons conserver.



Un outil de prise d'empreinte n'aura aucun mal à retrouver la véritable signature du serveur.

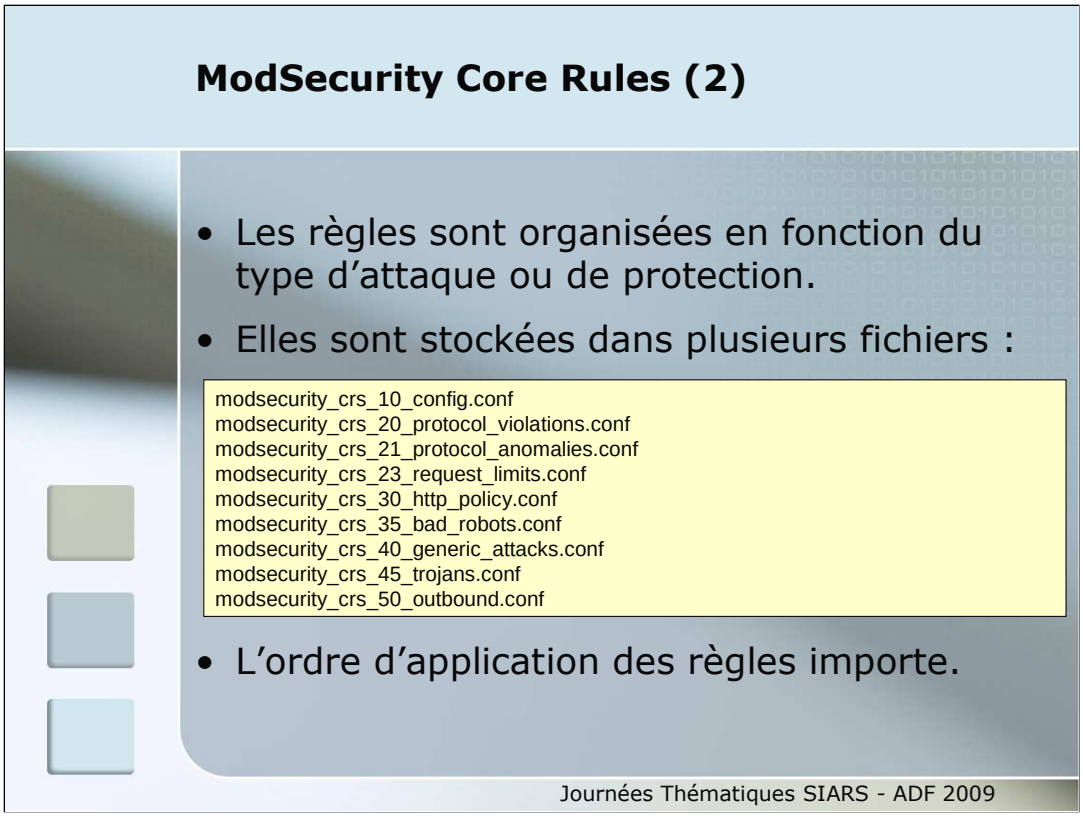
SecResponseBodyLimit : taille maximale autorisée (en bytes) pour le corps d'une réponse.

SecResponseBodyInMemoryLimit : taille maximale autorisée pour stocker en mémoire le corps d'une requête.

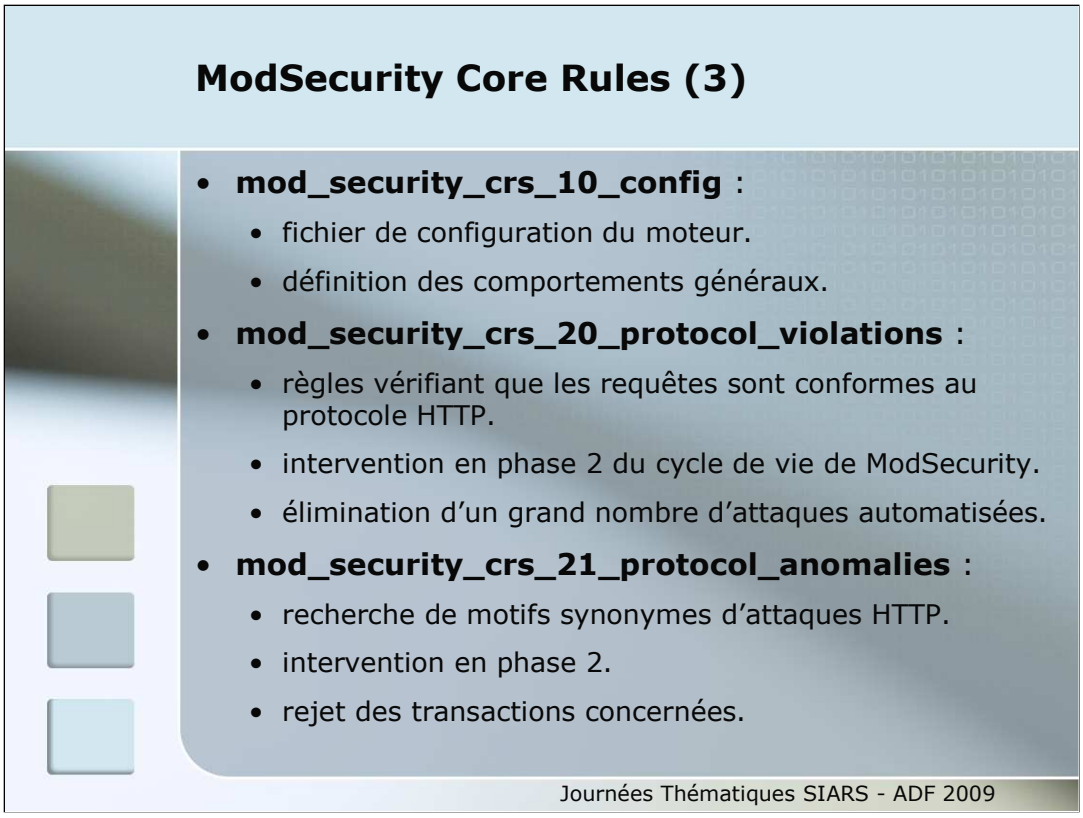
ModSecurity est désormais prêt à protéger nos applications Web. Pour cela, un kit de règles de base (ModSecurity Core Rules) nous est fourni. Il est régulièrement mis à jour et testé par l'éditeur.

ModSecurity Core Rules (1)

- ModSecurity propose des règles par défaut.
- Elles sont basées sur une approche de type liste noire.
- Elles offrent une protection honorable contre les attaques les plus connues :
  - détection de l'activité de certains scanners ou bots.
  - interception de tentatives d'accès à des cheveaux de Troie.
  - recherche les irrégularités dans l'utilisation du protocole HTTP (en entrée et en sortie).
- Elles permettent de modifier les messages d'erreurs renvoyés par le serveur.



Passons en revue la fonction de chacun de ces fichiers.



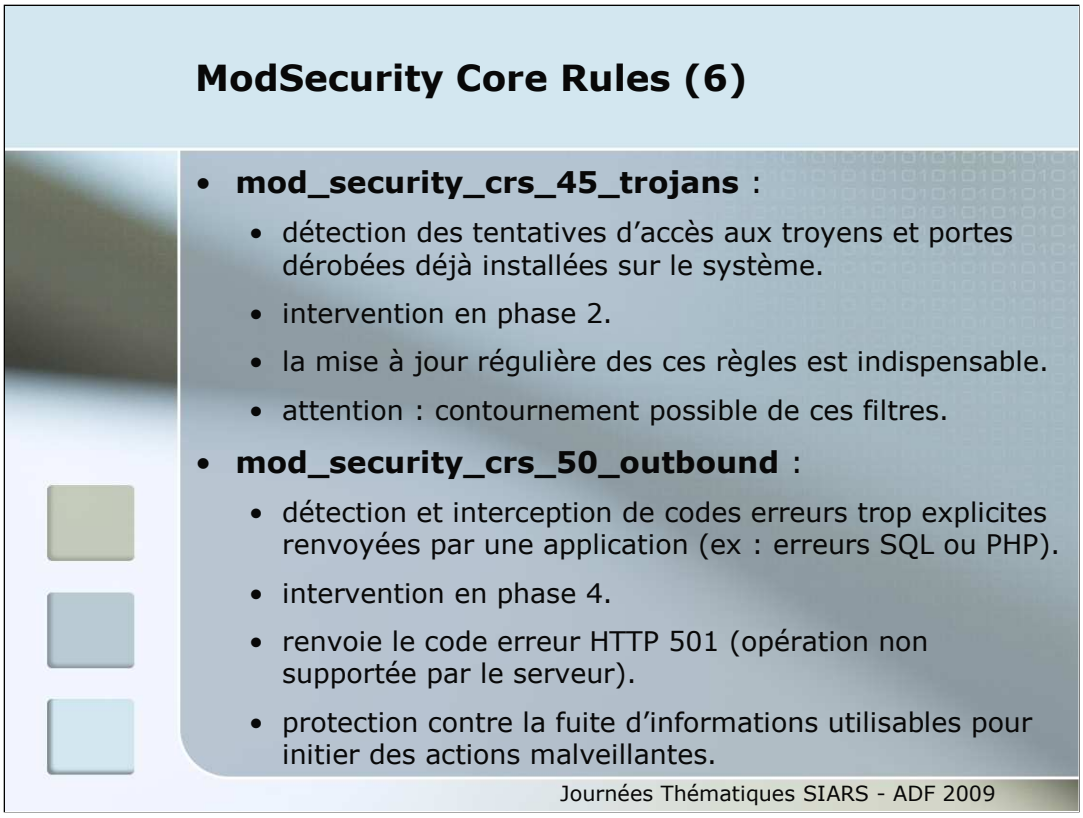
Exemple de motif synonyme d'une attaque : des requêtes à destination du serveur « localhost ».

ModSecurity Core Rules (4)

- **mod_security_crs_23_request_limits** :
  - limitation du nombre et de la taille des arguments soumis lors d'une requête HTTP.
  - intervention en phase 2.
  - protection contre les attaques de type *buffer overflow* ou manipulation des paramètres.
- **mod_security_crs_30_http_policy** :
  - règles restreignant l'utilisation du protocole HTTP par les clients :
    - blocage de méthodes (CONNECT, DEBUG, TRACE).
    - refus du protocole HTTP 0.9.
    - interdiction de fichiers dont l'extension est synonyme de contenus dangereux (.ini, .key, .old).
  - intervention en phase 2.

ModSecurity Core Rules (5)

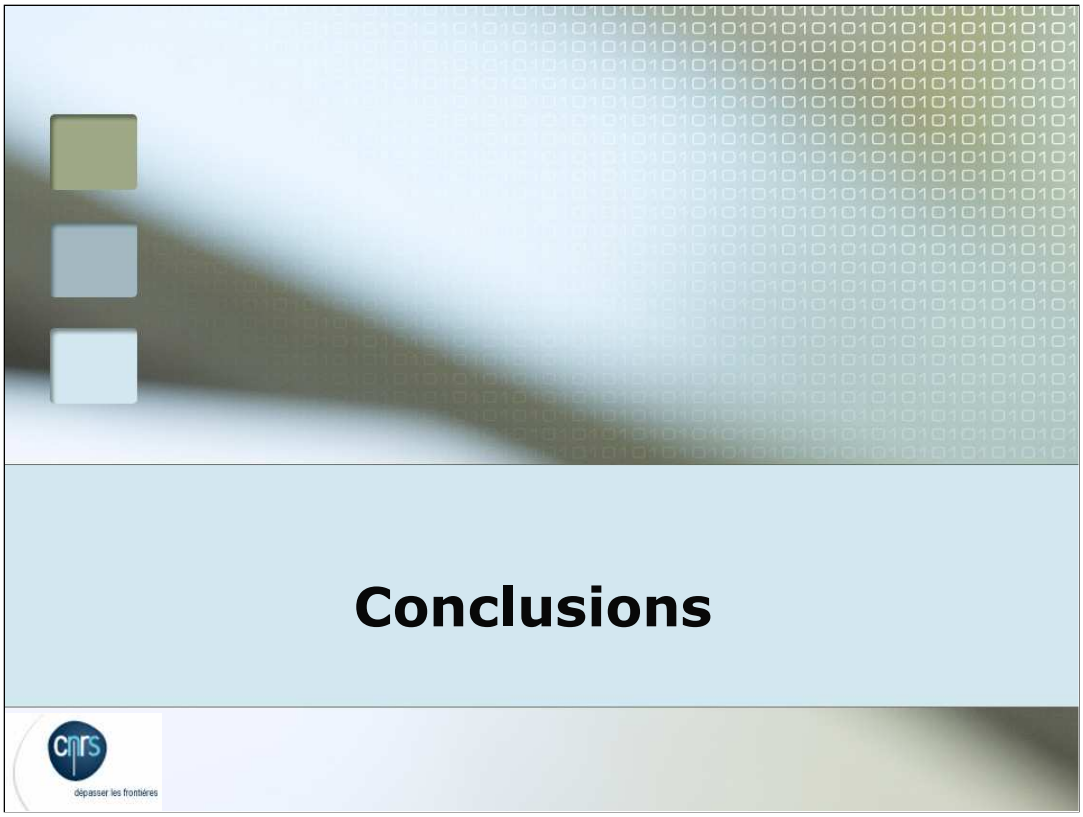
- **mod_security_crs_35_bad_robots** :
  - limitation des nuisances générées par des robots d'indexation qui n'en seraient pas.
  - enregistrements des transactions, mais pas de blocage.
  - intervention en phase 2.
  - détection des scanners de sécurité.
- **mod_security_crs_40_generic_attacks** :
  - mise en place de règles contre des attaques connues :
    - protection des sessions.
    - injections de code SQL.
    - attaques de type Cross Site Scripting (XSS).
    - accès ou injection de commandes.
    - injections diverses (PHP, LDAP, SSI, etc.).
  - intervention en phase 2.

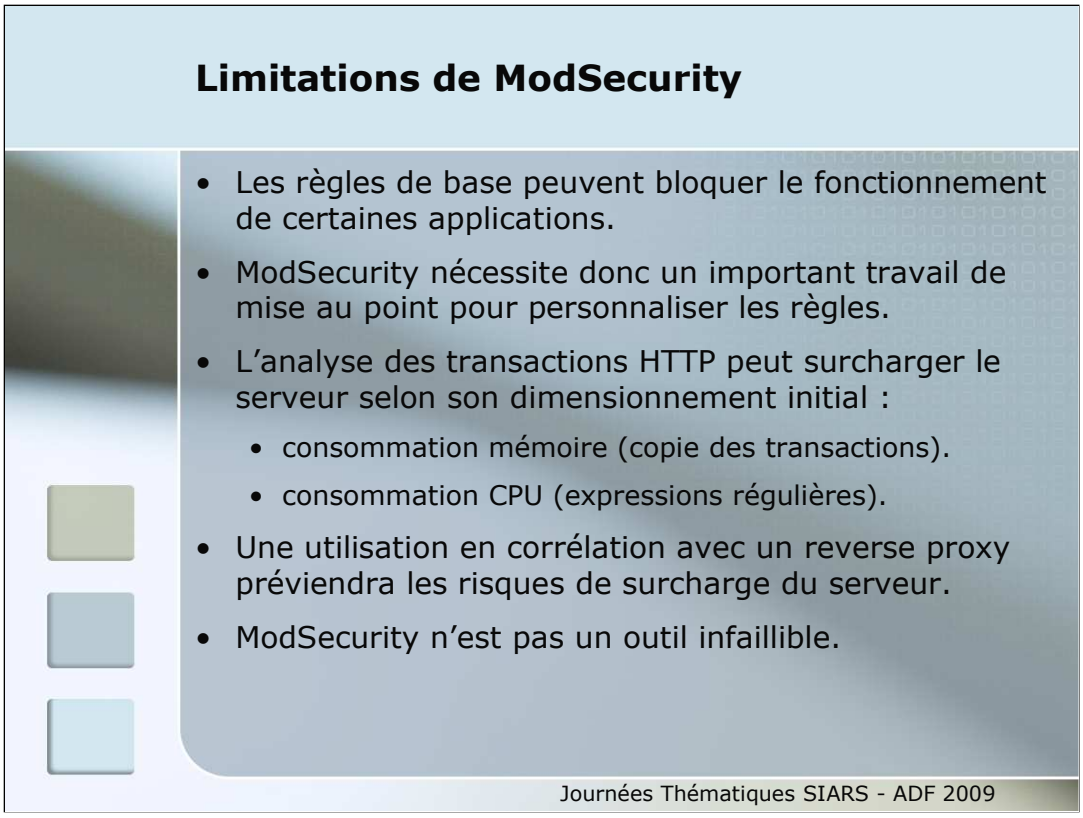


Puisque ModSecurity est un produit libre, les pirates et autres personnes mal intentionnées ont accès au contenu des règles. Il leur est donc possible de mettre en œuvre de nouvelles attaques pour les contourner.

ModSecurity Core Rules (7)

- **mod_security_crs_55_marketing** :
  - journalisation des transactions initiées par les moteurs de recherches.
  - intervention en phase 2.
  - utilisation à des fins statistiques.
- **mod_security_crs_60_custom_rules** :
  - définition de nouvelles règles utilisateur.
  - cf TP.

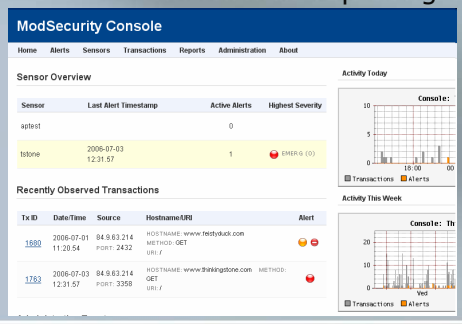

Conclusions
cnrs
dépasser les frontières



ModSecurity bénéficie d'un très bon support de la part des équipes de développement. Si vous rencontrez des problèmes de type « faux positifs » avec les règles de base, vous pouvez soumettre ce cas, à l'adresse security@modsecurity.org, pour qu'il soit résolu.

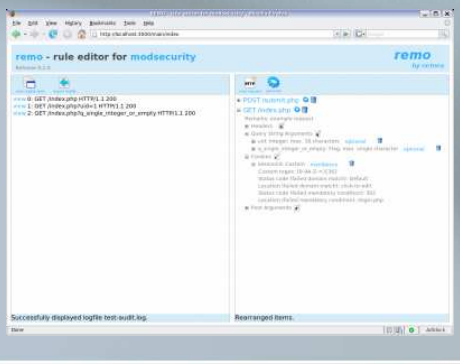
Outils complémentaires (1)

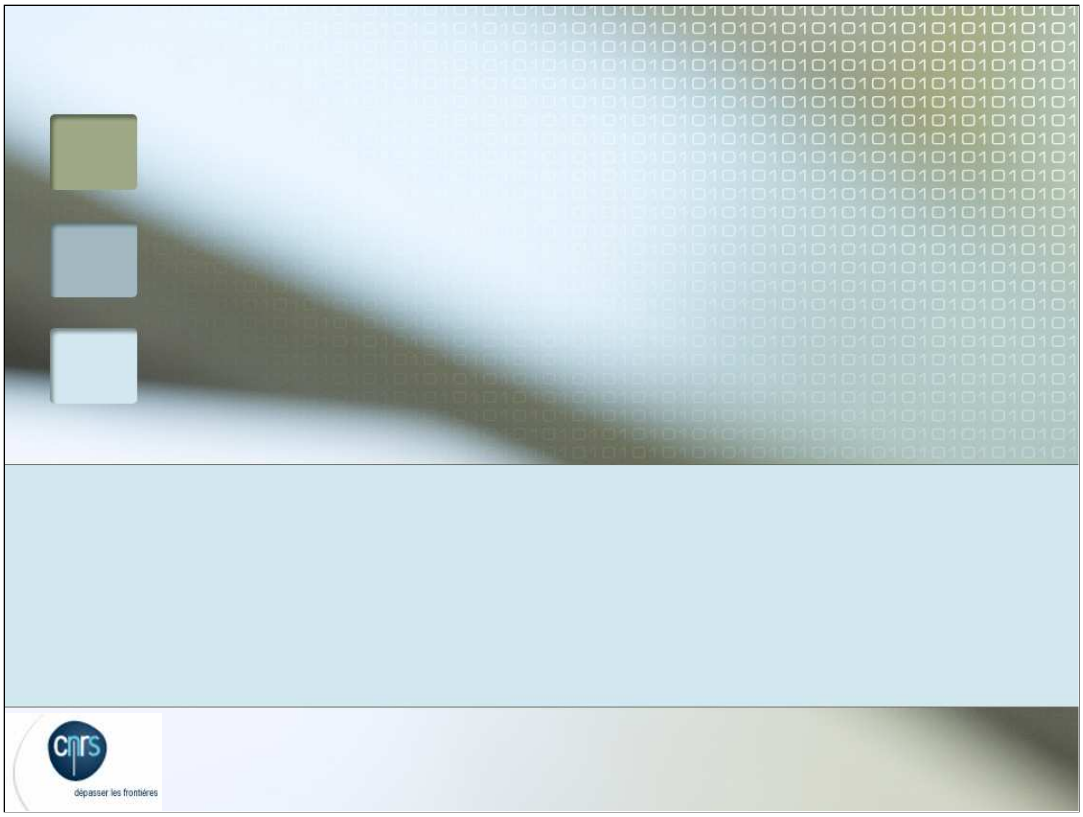
- ModSecurity Console permet de centraliser et de surveiller les événements de ModSecurity en réel :
  - elle est développée en Java.
  - elle peut être installée sur une machine indépendante.
  - une version gratuite permet d'analyser les traces de 3 instances de ModSecurity.
  - elle propose des fonctions de « reporting ».



Outils complémentaires (2)

- REMO est un outil graphique de création de règles de filtrage pour ModSecurity :
  - REMO : Rule Editor for MOdSecurity.
  - il est développé en langage Ruby et il utilise le framework Rails.
  - l'outil est en cours de développement.




cnrs
dépasser les frontières